Universidade Federal do Ceará
Centro de Ciências
Departamento de Computação
Mestrado em Ciência da Computação

Implementação de um Gateway Transacional para o Ambiente de Integração Web Banco de Dados baseado nos princípios das Transações Encadeadas e Compensatórias.

por
Marcelo Bezerra de Alcântara
(marcelo@ufc.br)

Dissertação submetida ao Departamento de Computação da Universidade Federal do Ceará, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: *Javam de Castro Machado*

Fortaleza, dezembro de 2000

Dedico este trabalho
a meus pais, Domingos Urbano e Maria Beserra,
a meus irmãos, Sandra, Cleidson e Domingos J.r.,
e a minha esposa, Liliane

Agradecimentos

Agradeço inicialmente a Deus por me iluminar neste trabalho. Agradeço também a meus pais – Domingos Urbano e Maria Beserra, a meus irmãos - Sandra, Cleidson e Domingos Jr. e a minha esposa Liliane. Todos eles forneceram um grande incentivo para o desenvolvimento deste trabalho.

Ao professor e orientador Javam Machado, pelas leituras de textos, reuniões, esclarecimento de dúvidas e pelo envio de importantes dicas para esta dissertação.

Aos amigos e colegas mestrandos, em especial aos membros da turma de 1997 (Luís Cláudio, Adriano, Miguel, João Fernando, Aragão e Valéria), deixo meus agradecimentos, pela convivência e troca de experiências.

Gostaria de agradecer também não somente aos professores do Departamento de Computação, pelos conhecimentos que me foram passados, bem como à equipe de suporte técnico, e aos demais funcionários do centro pela eficiência e prestatividade.

Agradeço a meus amigos de trabalho do POP-CE, Marcos Frota, Adriana, Wellington, Rogério, Crisamon e Michele, pelas grandes experiências.

Ao CNPq, que financiou minha bolsa de mestrado e ao Centro de Informática que proporcionou a estrutura necessária para a execução deste trabalho.

A todos vocês, mais uma vez, só tenho a dizer: muito obrigado.

Resumo

Desde a sua criação até os dias atuais a Web tem sofrido contínuas alterações, vem passando rapidamente da condição de troca irrestrita de documentos, conforme projetada inicialmente, para tornar-se uma plataforma de desenvolvimento de inúmeras aplicações. Seguindo esta tendência, foi inevitável a integração da Web com os Sistemas Gerenciadores de banco de dados (SGBD),visto que estes são tradicionalmente componentes indispensáveis para o gerenciamento dos dados das aplicações. Para fazer tal integração foi necessário a criação de um novo componente de software, o Gateway de banco de dados, com a função de fazer o mapeamento entre as requisições do cliente Web (browser Web) e do SGBD. Apesar de já existirem gateways bastante eficientes, o desenvolvimento de aplicações Web banco de dados ainda apresenta muitos problemas funcionais cujas soluções não são triviais. Dentre eles está a garantia da corretude das transações. Isto é, as características das transações (atomicidade, consistência, isolamento e durabilidade) devem ser amplamente suportadas no ambiente de integração e o estado da Web deve corresponder ao do banco de dados. Já existem gateways que garantem em grande parte esta corretude, porém é possível identificar diversos elementos que apontam o modelo de transações planas dos atuais SGBDs como um dos fatores limitantes para a solução dos problemas transacionais. Portanto, é necessário que façamos um estudo mais aprofundado sobre as características das aplicações deste novo ambiente a fim de determinarmos um modelo de transações satisfatório. Neste trabalho apresentaremos vários conceitos envolvidos na integração Web banco de dados, dando uma maior ênfase aos problemas transacionais e a escolha de um modelo de transações que melhor satisfaça as necessidades da Web.

Palavras Chaves: Web, Sistemas Gerenciadores de Banco de Dados, transações, modelos de transações.

SUMÁRIO

Lista de Figuras e Tabelas

Capítulo 1

Introdução

1.1 Motivação

Desde a sua criação até os dias atuais a Web tem sofrido contínuas alterações, vem passando rapidamente da condição de plataforma para troca irrestrita de documentos, conforme projetado inicialmente, para tornar-se uma plataforma de desenvolvimento de inúmeras aplicações.

Seguindo esta tendência, foi inevitável a integração da Web com os Sistemas Gerenciadores de Banco de Dados (SGBD), visto que estes são tradicionalmente componentes indispensáveis para o gerenciamento dos dados das aplicações.

Para fazer tal integração, foi necessário criar um novo componente de software, o Gateway de Banco de Dados, com a função de fazer o mapeamento entre as requisições do cliente Web (navegador Web) e do SGBD, sendo que este pode ser implementado na forma de extensões da Web ou do SGBD, ou ainda como um componente de software independente.

Hoje já existem vários gateways comerciais, podendo ser agrupados segundo vários critérios. O critério mais utilizado é aquele baseado na localização do software integrador: no lado cliente ou no lado servidor Web.

Apesar de já existirem gateways bastante eficientes, o desenvolvimento de aplicações Web Banco de Dados ainda possui muitos problemas funcionais cujas soluções não são triviais [Lim97]. Dentre eles está a garantia da corretude das transações. Isto é, as propriedades das transações (atomicidade, consistência, isolamento e durabilidade) devem

ser amplamente suportadas no ambiente de integração e o estado da Web deve corresponder ao do banco de dados. Já existem gateways que garantem em grande parte esta corretude, porém é possível identificar diversos elementos que apontam o modelo de transações planas dos atuais SGBDs como um dos fatores limitantes para a solução dos problemas transacionais. Portanto, é necessário que façamos um estudo mais aprofundado sobre as características das aplicações deste novo ambiente a fim de determinarmos as características de um modelo de transações satisfatório.

Nesta dissertação, apresentaremos vários conceitos envolvidos na integração Web Banco de Dados, dando maior ênfase aos problemas transacionais e à definição de um modelo de transações que melhor satisfaça as necessidades da Web. Por fim, será implementado um gateway que suporte o modelo definido.

1.2 Problema

Durante alguns anos o principal problema para integração Web Banco de Dados esteve relacionado em como mapear as requisições do ambiente Web baseado em operações *"stateless"*, para as requisições do ambiente dos Bancos de Dados baseado em operações *"statefull"*. Ou seja, como mapear as operações do ambiente Web, baseado no protocolo HTTP, onde para cada nova interação entre o cliente Web e o servidor Web é necessário abrir uma nova conexão, não permitindo assim que duas ou mais operações entre o mesmo cliente e servidor pertençam ao mesmo contexto, para as operações do ambiente dos Bancos de Dados, onde a conexão entre o cliente e o servidor é mantida enquanto durar a sessão, permitindo assim que duas ou mais interações entre o mesmo cliente e servidor pertençam ao mesmo contexto.

Já existem arquiteturas que garantem este mapeamento [Lim97], como por exemplo, a arquitetura do Gerenciador de Aplicação e Gerenciador de Transação. Porém com a possibilidade de se desenvolver aplicações cada vez mais complexas para este ambiente, começou-se a observar que o modelo de transações tradicional (planas) não satisfazia muita das necessidades das aplicações, como por exemplo, referência a resultados intermediários e transações de longa duração.

Este fato sugere que é necessário fazer um estudo mais aprofundado das

características deste novo ambiente para se determinar um modelo de transações que se adapte as suas necessidades.

1.3 Objetivo

O objetivo principal desta dissertação é analisar os vários modelos de transações existentes e definir um modelo que venha a satisfazer as necessidades do ambiente de integração Web Banco de Dados.

Como objetivos secundários, temos:

- Determinar as características relevantes do ambiente de integração Web Banco de Dados.
- Estudar e compreender os modelos de transações.
- Implementar um gateway que suporte o modelo de transações proposto.
- Desenvolver uma aplicação protótipo com o objetivo de validar os resultados da dissertação.

1.4 Trabalhos relacionados

As pesquisas na área Web Banco de Dados são bastante recentes e vêm abrangendo vários tópicos distintos. Em linhas gerais, três tópicos de pesquisa têm se destacado: a visão da Web como um enorme Banco de Dados distribuído e multiplataforma, o acesso a Bancos de Dados através da Web e o desenvolvimento de técnicas e ferramentas para desenvolvimento de aplicações Web integradas a SGBDs. Este trabalho está voltado basicamente para as duas últimas áreas.

Os trabalhos relacionados com o acesso a Bancos de Dados através da Web incluem entre outros, implementação de arquiteturas de integração das tecnologias [YK96, HM97], estudo das arquiteturas existentes [Per95, Kim96], análise de desempenho de arquiteturas [NF96, HM97] e o processamento de transações [LSC+97].

Devido o estudo dos problemas transacionais no Ambiente Web Banco de Dados ser bastante recente, poucos trabalhos foram publicados sobre este assunto [Bil98, LEK97].

Tais trabalhos estão mais preocupados com a integração dos vários gerenciadores de transação do que com os aspectos navegacionais do navegador e do modelo de transações utilizado no Banco de Dados. Entretanto deve-se ressaltar que já existem referências [Lim97] que ressaltam a importância de um estudo mais detalhado do assunto desta dissertação.

1.5 Metodologia

A metodologia desta dissertação é dividida em quatro fases:

- A primeira fase consistiu em fazer um levantamento mais detalhado das principais características do ambiente de integração Web Banco de Dados. No capítulo 2 serão apresentadas as principais características da Web, dos Bancos de Dados e do Ambiente de Integração Web Banco de Dados. No capitulo 3, será analisados os aspectos transacionais no ambiente de integração.
- A segunda fase propõe em realizar um estudo detalhado sobre os vários modelos de transações existentes, e fez um paralelo entre as características destes modelos com as do ambiente Web Banco de Dados, para se determinar um modelo mais adequado para este ambiente. No capitulo 4 será apresentado os principais modelos de transações existentes, e no capítulo 5 será definido o modelo de transações.
- A terceira fase consistiu em escolher uma arquitetura de integração, possivelmente a arquitetura de Gerenciador de Aplicação ou Gerenciador de Transação, e indicar possíveis mudanças nesta, para a utilização do modelo adotado na fase anterior, e analisar as vantagens e desvantagens obtidas com a implementação do gateway. No capitulo 6 será apresentado à implementação do gateway.
- A quarta fase consistiu em implementar uma aplicação para validar de uma maneira concreta os resultados obtidos na dissertação e ilustrar como desenvolver uma aplicação com o gateway implementado. Este processo é definido no capitulo 7.

Capítulo 2

Tecnologias envolvidas

2.1 Introdução

Este tópico apresenta uma visão geral sobre as tecnologias e a nomenclatura envolvidas na integração da Web e SGBDs. Na seção 2.2 são apresentados conceitos básicos envolvidos no ambiente Web que serão utilizados ao longo do trabalho. E na seção 2.3 os conceitos básicos envolvidos no ambiente dos Bancos de Dados, que serão utilizados no decorrer deste trabalho.

2.2 World Wide Web

A World Wide Web [BCL+94], ou simplesmente Web, é um sistema de informação hipermídia[1] de larga escala baseado na Internet. Desde seu desenvolvimento no CERN (*Conseil Européen pour la Recherche Nucléaire*), Suiça, em 1991, a tecnologia tem crescido rapidamente no ambiente da Internet, e agora está sendo extensivamente utilizada em aplicações comerciais e acadêmicas. Ao contrário de outros sistemas hipermídia que se concentram nas interfaces de visualização e na estrutura de armazenamento, a Web focaliza a criação de um formato que seja acessível de forma semelhante em diferentes ambientes computacionais.

[1] *Hipermídia* aqui é a reunião de hipertexto e multimídia num único documento Web. Hipertexto é a combinação de textos não lineares, de forma que se pode saltar de texto para outro dinamicamente por meio de vínculos (links).Multimídia é a combinação de textos e outras mídias como imagens estáticas ou em movimento, sons, vídeos e gráficos, entre outros.

2.2.1 Arquitetura da Web

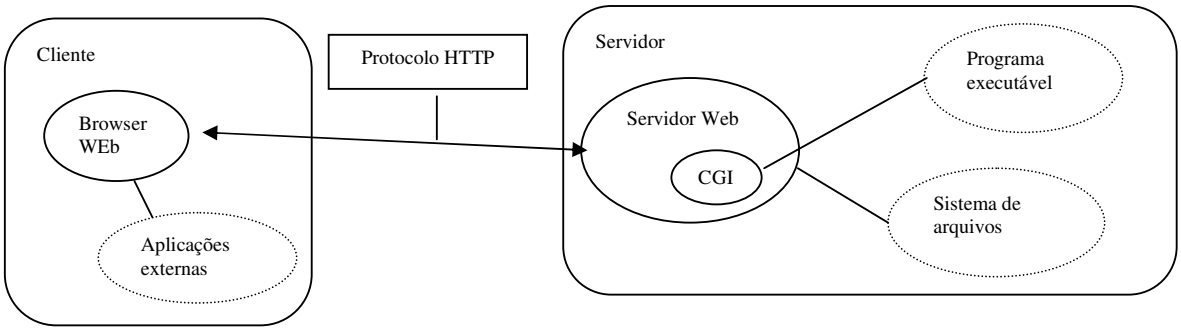

FIGURA 2.2.1 – ARQUITETURA WEB SIMPLIFICADA

A figura 2.2.1 mostra a arquitetura simplificada da Web. Como se pode observar, é uma típica arquitetura cliente/servidor. O lado cliente é composto por navegadores, chamados de browsers Web, que possuem a capacidade de solicitar e exibir documentos sobre a rede. Os browsers podem acoplar aplicações externas, permitindo a visualização de alguns tipos de dados que não podem ser interpretados por eles, como sons ou imagens em movimento. Atualmente, os browsers estão sendo estendidos para permitir a execução de lógica de programação, como por exemplo, a inserção da Máquina Virtual Java no browser. No lado servidor, temos o servidor Web, que tem a função de atender aos pedidos dos clientes Web por documentos armazenados no sistema de arquivos da plataforma onde se encontra instalado. Possui também a capacidade de disparar uma aplicação externa como, por exemplo, a execução de um programa via interface padrão CGI (Common Gateway Interface). Os servidores Web também estão sendo estendidos para suportarem a execução de lógica de programação, como por exemplo, a inserção da Máquina Virtual Java no servidor, no caso da tecnologia de "*serverlets"* [Hal00].

A base desta arquitetura compreende quatro padrões abertos, o URL para localizar as informações, o HTML para descrever a formatação da informação, o HTTP para transferir a informação e o CGI para permitir criação de informações dinâmicas.

2.3 Sistemas de Banco de Dados

A tecnologia de Banco de Dados tem causado um grande impacto no

desenvolvimento das aplicações. Podemos dizer que os Bancos de Dados terão um papel crucial em quase todas as áreas onde os computadores são usados.

2.3.1 Arquitetura de Sistema de Banco de Dados

Atualmente, as aplicações de Banco de Dados são, em geral, baseadas no ambiente cliente/servidor. As aplicações cliente/servidor de banco de dados consistem de três componentes de software [Lat94]. O primeiro componente é o módulo de gerenciamento de dados (SGBD) que suporta o armazenamento e a recuperação dos dados. Este pode ser subdividido em dois sub-módulos: um para o processamento de consultas e outro para o acesso aos dados. O segundo componente é o módulo que implementa a lógica da aplicação utilizando a interface do módulo de gerenciamento de dados. O último é o módulo de apresentação usado na interação com os usuários.

Existem várias variações de sistemas cliente/servidor de bancos de dados. A mais comum, atualmente, é a arquitetura de dois níveis, como ilustra a figura 2.3.1.

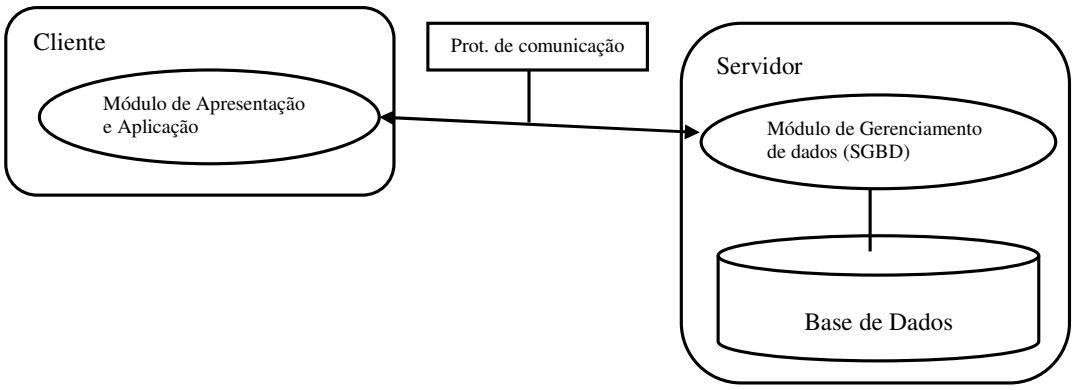

FIGURA 2.3.1 – ARQUITETURA DE DOIS NÍVEIS CLIENTE SERVIDOR DE BANCO DE DADOS.

Na arquitetura de dois níveis, os módulos de aplicação e apresentação são implementados como um único módulo.

Porém, existem previsões que a arquitetura de três níveis venha a ser predominante

no futuro [Sch94a, Sch94b]. Nesta o módulo de aplicação é separado do módulo de apresentação, que ficará agora no servidor, de onde é compartilhado entre os vários clientes. A principal vantagem desta arquitetura é a maior flexibilidade e escalabilidade para sistemas de grande porte.

2.3.2 SQL

O acesso ao banco de dados relacional é feito usando SQL (Structured Query Language) [EN94] que é a linguagem de banco de dados padrão da ISO. SQL é uma linguagem não-procedural, podendo ser efetivamente usada para formular consultas sem conhecer a representação interna dos dados. Possui comandos para definição dos dados, consultas e atualizações. Assim, é ao mesmo tempo uma DDL (Data Definition Language) e uma DML (Data Manipulation Language). Além disso, tem facilidades para definições de visões dos bancos de dados e para criação e exclusão de índices.

2.3.3 Transação

Uma transação é a unidade de consistência e reabilitação computacional [OV91]. Intuitivamente, uma transação sobre um banco de dados executa uma ação sobre ele, e gera uma nova versão do banco de dados, causando uma mudança de estado. Isto é similar a execução de uma consulta, exceto que se o banco de dados estava consistente antes da execução da transação, podemos garantir a consistência depois de sua execução, salientado o fato de que uma transação pode ser executada concorrentemente com outras, e que falhas podem ocorrer durante sua execução. Em geral uma transação é constituída de uma seqüência de operações de leitura e escrita no banco de dados.

Os aspectos de consistência e reabilitação das transações são decorrentes de quatro propriedades: atomicidade, consistência, isolamento, e durabilidade.

- Atomicidade refere-se ao fato de que uma transação é tratada como uma única operação, isto é, todas as operações da transação são executadas ou nenhuma delas.
- A consistência de uma transação é simplesmente sua corretude. Ou seja, uma

transação é um programa correto que faz o mapeamento de um estado consistente para outro.

- O isolamento requer que cada transação veja o banco de dados consistente a todo instante. Em outras palavras, a execução de uma transação não pode interferir no resultado de outras transações concorrentes antes que estas validem.
- A durabilidade garante que o resultado de toda transação validada é permanente e não pode ser perdido pelo banco de dados.

2.3.4 Modelo de transações

Um modelo de transações [Elm90] define quando uma transação começa, quando ela finaliza, e que ações são tomadas no caso de falhas. As transações gerenciadas pela maioria dos SGBDs existentes seguem o modelo de transações planas, assim chamado porque existe uma única camada de controle na aplicação. As principais restrições com relação às transações planas são a impossibilidade de se fazer referência a resultados intermediários e aceitar ou cancelar parte das ações. Este modelo é adequado para transações de curta duração. Porém como as aplicações vêm se tornando cada vez mais complexas, o critério de corretude que constitui a aceitabilidade para a aplicação, precisa de critérios mais complexos do que os oferecidos no modelo de transações planas [RC94] . Devido a estes fatos, têm surgido extensões do modelo de transações planas para atender as necessidades dos novos ambientes (distribuídos, cooperativos e heterogêneos). Exemplos destes novos modelos são as transações aninhadas e o modelo SAGAS, que serão detalhadamente apresentados no capítulo 3.

2.4 Ambiente de Integração

Desde sua criação até os dias atuais a Web tem sofrido contínuas alterações, vem passando rapidamente da condição de troca irrestrita de documentos, conforme projetada inicialmente, para tornar-se uma plataforma de desenvolvimento de inúmeras aplicações. Seguindo esta tendência, foi inevitável a integração da Web com os sistemas gerenciadores de banco de dados (SGBD), visto que estes são, tradicionalmente, componentes indispensáveis para o gerenciamento dos dados das aplicações.

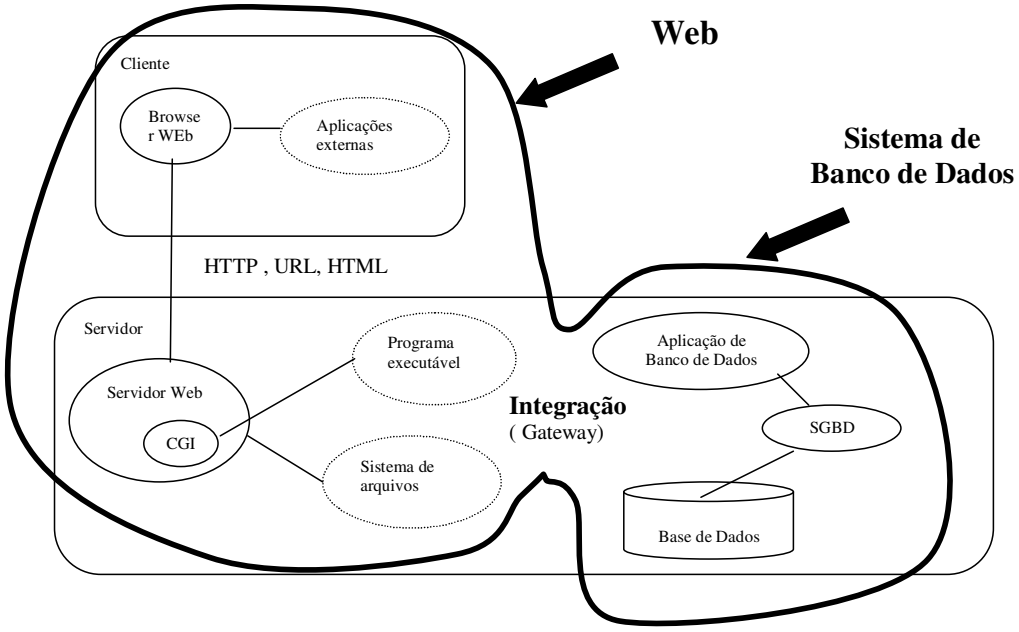

FIGURA 2.4 – INTEGRAÇÃO WEB BANCO DE DADOS

A integração destas duas tecnologias ocasionou o surgimento de uma nova área de pesquisa, o ambiente de Integração Web Banco de Dados. A figura 2.4. ilustra este novo ambiente.

Neste ambiente, o usuário utiliza o cliente Web para fazer suas requisições e visualizar os resultados, o SGBD para o armazenamento dos dados e suas manipulações, e o Gateway de Banco de Dados para fazer o mapeamento entre as requisições do cliente Web e do SGBD. A comunicação entre o cliente Web e o SGBD pode ser direta, ou através do servidor Web.

2.4.1 Áreas de pesquisa

As pesquisas na área Web Banco de Dados são recentes e abordam vários tópicos distintos. Segundo [Lim97], três tópicos de pesquisa têm se destacado atualmente: a visão da Web como um enorme banco de dados distribuído e multiplataforma, o desenvolvimento de técnicas e ferramentas para construção de aplicações Web integradas a SGBDs e o acesso a bancos de dados através da Web.

Os trabalhos relacionados com a visão da Web como um enorme banco de dados distribuído, inclui dentre outros: técnicas de visualização da Web [MAD96]; utilização de

tabelas para armazenamento de documentos da Web [RA97] e propostas de linguagens de consulta declarativas para recuperação de informações na Web [KS95, MMM96, MMM97].

Em relação ao desenvolvimento de técnicas e ferramentas para construção de aplicações Web integradas a SGBDs, têm surgido vários estudos para a criação de metodologias de desenvolvimento de aplicações [FP97,Kes95, DI*95, SR95]. Outros tópicos de pesquisa nesta área são: requisitos para o desenvolvimento de aplicações [PHMS96, PF98]; propostas e/ou análise de software que ocupam uma camada intermediária na integração das tecnologias [HM96, Pla96, Tel97] e linguagens de programação na Web integradas a banco de dados [Dua96, Sha96].

Os trabalhos relacionados com o acesso a bancos de dados através da Web compreendem entre outros: a implementação de arquiteturas de integração [ORA97, JG96, SP96, YK96, HM97]; estudo das arquiteturas existentes [Per95, Kim96, GGS97, CSJ+98], análise de desempenho das arquiteturas [NF96, HM97] e o estudo dos aspectos transacionais [LSC+97, Bil98, LEK97, PJP+91].

Esta dissertação não aborda a Web como um enorme banco de dados distribuído, mas analisa aspectos funcionais da integração SGBD e Web para o desenvolvimento de aplicações Intranet/Internet. Neste contexto, enfatiza os aspectos transacionais referentes ao ambiente de integração.

2.4.2 Requisitos de integração

Assim como em todas as áreas de pesquisa envolvidas na integração de duas ou mais tecnologias, é importante definirmos que requisitos são necessários à integração da Web com os SGBDs, pois estes servirão de base para comparações (vantagens e desvantagens) entre as soluções apresentadas.

Os requisitos podem variar de autor para autor, entretanto, em grande maioria o critério de escolha está relacionado com a questão de não limitar ou alterar a natureza do ambiente da Web e do banco de dados.

A satisfação ou não de um requisito depende do tipo de aplicação a ser desenvolvida. Por exemplo, um simples quiosque de informações não requer um alto controle transacional (requisito de consistência e integridade), já num Shopping Virtual o controle transacional é de suma importância.

Os requisitos adotados nesta dissertação foram obtidos da junção dos propostos em [Lim97] e [EKR97], pois estes apresentam critérios de alto nível, bastante abrangentes.

Os requisitos identificados nesta dissertação são:

- *Interface do usuário harmoniosa*: significa que a interface do usuário deve permitir a integração de diferentes tipos de mídia. Por exemplo, o resultado de uma consulta ao banco de dados pode apresentar, junto no mesmo browser, tanto texto quanto imagens. Esta propriedade está relacionada com a característica multimídia da Web.
- *Modelo interativo da Web*: a interatividade das páginas Web corresponde a possibilidade de mudar o contexto com a reação da entrada do usuário. Esta característica refere-se a capacidade navegacional da Web (hipertexto).
- *Consistência e integridade*: corresponde a restrição de que os dados visualizados no navegador Web correspondam aos dados armazenados no banco de dados. Tal requisito está intimamente relacionado ao suporte das características transacionais neste ambiente.
- *Desempenho:* corresponde a mensuração da velocidade da resposta de uma requisição do cliente Web (usuário) ao banco de dados.
- *Escalabilidade:* pode ser mensurada pelo tamanho do banco de dados bem como o número de usuários simultaneamente capazes de fazer acesso ao banco de dados. Esta requisição está relacionada com o desempenho.
- *Segurança*: corresponde a inviolabilidade dos dados.
- *Desenvolvimento*: permitir a criação de aplicações Web Banco de Dados de forma mais transparente e abrangente.

2.4.3 Problemas da integração Web e Banco de Dados

A integração da Web e banco de dados proporcionou a criação de um novo ambiente, capaz de combinar a interatividade e simplicidade da Web com os mecanismos avançados de gerenciamento de dados dos SGBDs.

Várias são as vantagens obtidas com esta integração, tais como:

- Interface do usuário harmoniosa;
- Suporte das características transacionais;

- Facilidade para modelar de tipos complexos e seus relacionamentos;
- Controle de falhas, entre outras.

Entretanto, para obtermos essas vantagens, é necessário que sejam solucionados alguns problemas.

Esta seção pretende discutir alguns dos problemas, tendo como referencial os requisitos especificados anteriormente, e apresentar algumas das soluções propostas na literatura para estes problemas.

2.4.3.1 Problemas Transacionais

A capacidade transacional é umas das principais características que diferencia um SGBD de outros sistemas de armazenamento de dados (gerenciadores de arquivos, arquivos simples). Esta característica permite aos usuários executarem operações sobre o banco de dados de forma mais transparente, pois se abstraem de questões relacionadas com o controle de concorrência e falhas computacionais. Assim, qualquer tecnologia que se integre aos bancos de dados deve ser capaz de suportá-la.

É nesta questão que surge um dos principais problemas na integração Web e Banco de Dados: como integrar um ambiente baseado em transações "*stateles"* (Web) com um outro baseado em transações "*statefull"* (SGBDs)?.

Os atuais SGBDs foram projetados para atender pedidos de usuários que se conectam ao banco de dados em sessões contínuas, ou seja, o usuário fica conectado ao SGBD enquanto durar a sua sessão [GR93]. A orientação à sessão contínua do usuário é a base para o suporte das propriedades transacionais, pois permite ao SGBD gerenciar eficientemente o processamento de transações, controlar a concorrência aos dados, realizar otimização de consultas, garantir a integridade transacional, manter os mecanismos de autenticação e autorização de acesso aos objetos do banco, entre outras funcionalidades.

Assim, a mudança deste ambiente para um fundamentado em transações "*stateless*" como da Web, onde a próxima ação do servidor Web não depende da ação tomada anteriormente[PF95], implica em vários questionamentos relacionados com as características transacionais, por exemplo: como manter o contexto transacional? E que modelo de transação melhor se adapta ao ambiente de integração?

O problema da manutenção do contexto transacional é devido a descontinuidade entre as interações do usuário, ou seja, o usuário não pode fazer referência a informações

referentes a outras interações (outras páginas). Isto decorre da natureza "*stateless*" do protocolo HTTP utilizado na comunicação entre o cliente e o servidor Web.

Tal característica inviabiliza muito dos mecanismos existentes nos SGBDs, como por exemplo:

1. *Uso de cursores:* como as informações das interações passadas são perdidas, então é necessário criar um novo cursor a cada interação, ocasionado a perda da posição da tupla apontada na interação anterior.
2. *Otimização de consultas:* como a cada interação é necessário criar um novo contexto, então os dados armazenados na memória principal e os planos de consultas pré-compilados, base para os mecanismos de otimização, são perdidos, não podendo ser utilizados nas próximas interações.
3. *Mecanismo de autenticação:* como a cada interação é perdido o contexto, torna-se necessário que o usuário, a cada interação, informe qual o seu login e senha.
4. *Mecanismos de bloqueio*: como uma transação pode envolver mais de uma interação e os bloqueios a cada interação são perdidos, então não é possível preservar, neste caso, a consistência da transação.

Portanto, para qualquer solução que vislumbre garantir a capacidade transacional neste ambiente, é necessário manter o estado da aplicação. Já existem algumas técnicas consagradas para manutenção do estado, estas vão desde soluções mais simples, como a utilização de *cookies* [Net97] e variáveis escondidas no documento HTML, até soluções mais complexas, onde são criados componentes de software intermediários, responsáveis em manter a conexão com o banco de dados aberta até a aplicação finalizar. [Lim97] apresenta detalhes destas soluções, ressaltando suas vantagens e desvantagens.

Com relação a definição de um modelo de transações para o ambiente de integração Web Banco de Dados, podemos citar a proposta de [SDB+91] que propõem o suporte do conceito de transações para Hipertexto, e a de [JGS96] que descreve uma pesquisa para prover o serviço de transações de longa duração e cooperativas do sistema Oz para Web (OzWeb).

Deve-se ressaltar, entretanto, que nenhum artigo obtido a partir do levantamento bibliográfico deste trabalho propõe-se a especificar um modelo de transações que permita eliminar as limitações impostas pelo modelo de transações planas às aplicações Web Banco de Dados, tema desta dissertação, pois os trabalhos que utilizam outros modelos de

transações (transações distribuídas [LEK97, Bil98], transações de longa duração cooperativas [JGS96]) pretendem apenas dar suporte a certas características de determinadas aplicações.

Outro problema transacional ocasionado pela característica distribuída da Web e a inserção de diferentes SGBDs na Internet, refere-se a capacidade de executar transações distribuídas na Web, ou seja, permitir que a execução de uma transação envolva a execução de outras transações em diferentes SGBDs, ou Gerenciadores de Transação. Como trabalhos relacionados a esta questão, temos os referentes a especificação do TIP (Transaction Internet Protocol)[Bil98], padrão desenvolvido pela IETF (Internet Engineering Task Force) para a execução de transações distribuídas na Internet. O TIP é baseado em um simples protocolo de validação em duas fases, que permite a comunicação entre os gerenciadores de transação para a execução de transações distribuídas. O ponto chave deste protocolo, é que a comunicação entre os gerenciadores de transação pode ser feita com qualquer protocolo de comunicação da Internet.

[Bil98] cria um novo modelo de transações, I-transaction, baseado em ações atômicas autogerenciáveis, com mecanismos de tolerância à falhas e segurança. Esta solução é destinada a execução de transações distribuídas, através de bancos de dados heterogêneos, não conhecidos a priore, situados na Internet, acessados através da interface da Web. Este modelo pode ser implementado sobre o TIP.

2.4.3.2 Desenvolvimento de Aplicações Web Banco de Dados

Com relação ao ambiente de desenvolvimento de aplicações Web baseado em banco de dados vários problemas merecem destaque.

Um dos problemas está relacionado com dificuldade na escolha da tecnologia ideal para o desenvolvimento da aplicação, pois existe uma grande variedade de tecnologias para o desenvolvimento, sendo que cada uma delas tem suas próprias particularidades e finalidades (Java, JavaScript, CORBA, HTML/ CGI, PL/SQL, VbScript, Perl, Python, Juice, Tcl/TK, ServerLet, ASP). Requerendo, assim, por parte do projetista da aplicação um amplo conhecimento e contínua reciclagem para poder escolher e utilizar a melhor tecnologia. Além disso, a escolha de uma determinada solução pode ocasionar sérios problemas de portabilidade, pois estas são na maioria dos casos proprietárias e específicas a um determinado servidor Web, ou cliente Web, ou SGBD.

O reuso é outro problema, pois se o desenvolvedor resolver mudar de tecnologia é possível que seja necessário rescrever tudo novamente, visto que a interoperabilização das ferramentas é geralmente uma tarefa muito onerosa.

As contínuas atualizações das tecnologias utilizadas no desenvolvimento das aplicações podem gerar problemas de manutenção, caso o projetista resolva atualizar a aplicação para uma versão mais recente, ainda não suportada amplamente pelo browser, ou Servidor Web ou SGBD, requerendo assim, que haja duas versões correntes da aplicação, uma baseada na nova tecnologia e a outra na tecnologia anterior.

A falta de metodologias para o desenvolvimento destas aplicações é outro sério problema, pois a grande maioria das ferramentas é desenvolvida somente para trabalhar ao nível de implementação, não disponibilizando um modelo lógico, através do qual o projetista possa abstrair-se de detalhes da implementação. Assim, essas ferramentas não cobrem o ciclo de vida completo (análise, desenvolvimento, implementação e manutenção) de uma aplicação, conforme definido pela Engenharia de Software.

Apesar de todos estes problemas, já estão surgindo várias ferramentas que facilitam o desenvolvimento destas aplicações. [Fra98] faz um estudo detalhado destas ferramentas, dividindo-as segundo uma taxonomia baseada em critério da engenharia de software, arquitetura e qualidade da expectativa do usuário. Ele considera o desenvolvimento da aplicação Web sobre três perspectivas:

- Estrutural: que objetos constituem a aplicação.
- Navegacional: como navegar de objeto para objeto.
- Apresentação: como os objetos serão visualizados.

Segundo [Fra98], estas ferramentas podem ser agrupadas em cinco categorias básicas:

- *Editor Visual de HTML e gerenciador de site*: Esta categoria contém as ferramentas que são diretamente uma evolução dos editores de HTML. Estas não suportam realmente complexas aplicações Web Banco de Dados, mas não são menos importantes, pois foram as pioneiras em muitos conceitos. Exemplos delas são: Adobe SiteMill and PageMill, NetObject Inc.'s Fusion, SoftQuad's HotMetal, Claris Home Page, Macromedia's backstage designer e Microsoft FrontPage.

- *Ferramenta de editoração hipermídia para Web*: Essas ferramentas compartilham o mesmo foco no editoramento como nos editores de HTML, mas têm uma origem diferente, porque foram inicialmente concebidas para o desenvolvimento de aplicações hipermídia off-line, e têm sido estendidas recentemente para suportarem aplicações Web. Exemplos destas ferramentas são: Asymetrix's Toolbook II Assistant/Instructor, Macromedia's Director and Authorware, Formula Graphics Multimedia 97, Aimtech Iconauthor e Allen Communication Quest.
- *Integradores de HTML-DBPL*: essa categoria de produtos está especificamente relacionada com a integração dos bancos de dados e aplicações Web, e persegue esta integração ao nível de linguagem. A integração pode-se dar de duas formas: através da extensão do HTML e da extensão das linguagens de banco de dados. Exemplos destas linguagens são: Cold Fusion Web Database Construction Kit by Allaine Inc, Microsoft's Active Server Pages (ASP) and Internet Database Connector (IDC), StoryServer by Vignette Corporation, HAHT Software's HahtSite, Oracle's PL/SQL Web Toolkit, Sysbase's PowerBuilder Web.PB class library e Borland's Web interface for the Delphi Client Server Suite.
- *Editores de formulários, geradores de relatórios e database publishing wizards para a Web*: Esta categoria compreende todos os produtos que têm a característica comum de suportar os conceitos e ferramentas tradicionais para o projeto de banco de dados no desenvolvimento de aplicações Web. Exemplos destas ferramentas são: Visual Basic 5, Access97 , Oracle Developer 2000, Borland's IntraBuilder, Sysbase's PowerBuilder e Apple's WebObjects.
- *Gerador de aplicações dirigido a modelos:* As ferramentas desta categoria são topo, pois permitem um alto nível de automação e completo ciclo de vida, pela utilização de um modelo conceitual e técnicas de geração de código para o desenvolvimento de aplicações Web. Exemplo desta ferramenta, já comercial, é o Oracle Web Development. Outros exemplos são os protótipos: Autoweb [FP97], HSDL [Kes95], RMC [DI*95] e o OOHDM [SR95].

Por fim, podemos concluir que muitos dos problemas envolvidos no desenvolvimento de aplicações Web Banco de Dados serão superados com a utilização das

ferramentas de última geração. Entretanto, ainda são necessárias várias pesquisas nesta área.

2.4.3.3 Desempenho

A questão do desempenho é um ponto muito importante na escolha ou não de determinada solução para o ambiente de integração. O desempenho neste ambiente corresponde ao tempo de resposta de uma requisição do usuário ao SGBD.

Grande parte dos problemas de desempenho no ambiente de integração Web Banco de Dados, surge devido ao fato de que a Web foi projetada para visualizar documentos e não para ser uma plataforma de desenvolvimento de aplicações, pois mesmo os mecanismos utilizados para incrementar o desempenho da Web, como o cache e conexão *stateless*, tendem a degradar o desempenho neste ambiente, já que são gastos muitos recursos para a manutenção da integridade do cache, ou seja, o sistema tem que monitorar o cache para evitar inconsistência entre o estado da Web e do banco de dados, e a para manutenção do estado da aplicação, pois como a Web perde o estado entre interações subsequentes, então se deve criar mecanismos para que as informações sejam disponíveis a cada interação.

A existência de muitos componentes na mensuração do desempenho da Web é outro problema. O seu cálculo é feito a partir de três componentes: o desempenho do cliente Web, o tráfego na rede e o desempenho do servidor Web.

O desempenho no cliente Web refere-se à capacidade de se exibir os dados enviados pelo servidor Web, o gerenciamento do cache local, a execução de códigos associados aos dados enviados, como por exemplo, *applets* Java, e a ativação dos *plug-in*.

O tráfego na rede refere-se ao tempo necessário para prover acesso remoto, escolha da melhor rota para conexão com o servidor Web e transmissão dos dados na rede. Este é um problema potencial do ambiente Web principalmente se o contexto da aplicação for a Internet e se executada em horários de intenso tráfego na rede.

O servidor Web é um dos componentes determinantes no tempo de resposta de uma operação do cliente, pois ele é afetado pela carga de pedidos, pela plataforma de hardware e pelo ambiente de software (sistema operacional, gerenciador de arquivos e execução de programas CGI, entre outros), como analisado por [MAA96].

Além desses fatores, pode-se identificar outros que afetam o desempenho do ambiente de integração intimamente relacionado com os SGBDs, como é o caso da otimização de consultas, número de usuários e a utilização de procedimentos armazenados e visões.

A otimização de consultas refere-se aos mecanismos presentes nos atuais SGBDs utilizados para permitir um melhor gerenciamento do acesso aos dados armazenados, sendo um fator extremamente importante no desempenho do SGBD, porque a diferença no tempo de execução entre uma boa e má estratégia pode ser enorme [KS97].

Os dois mecanismos mais utilizados na otimização de consultas são a manutenção de um cache em memória para os dados consultados frequentemente e o armazenamento dos planos de consultas compilados e otimizados da conexão corrente. Dessa forma, consultas repetidas são executadas simplesmente interpretando a estratégia estabelecida anteriormente, sem nenhum *"overhead"* de recomputação, re-otimização e acesso ao disco.

Entretanto, ao levarmos esta característica ao ambiente de integração, estes mecanismos podem degradar o desempenho, pois se a cada interação do usuário com o banco de dados for necessário criar uma nova conexão, o *"overhead"* causado pelo otimizador será subutilizado, pois serão perdidos o cache dos dados e os planos de consultas, não compensando o seu tempo gasto durante a execução da consulta. Portanto, a estratégia de manter a sessão contínua do usuário no SGBD pode afetar extremamente o desempenho deste ambiente.

Com relação ao número de usuários, deve-se levar em consideração que os SGBDs foram projetados para atender a pedidos de um número limitado de usuários, assim ao passarmos para o ambiente de integração, onde o número de usuários esperado é extremamente grande, o desempenho do SGBD pode ser bastante afetado, como no caso do servidor Web. Para eliminar este problema é necessário, por exemplo, uma arquitetura de alto desempenho, que seja bastante escalável e com a possibilidade de processamento paralelo.

Outra característica dos SGBDs que pode aumentar o desempenho do ambiente de integração é o uso de técnicas como procedimentos armazenados (stored procedures) e visões. Estas técnicas são utilizadas para otimizar o desempenho de operações e consultas mais comuns. Assim, se uma interação da aplicação Web banco de dados for utilizada com

muita freqüência, procedimentos armazenados e visões no SGBD devem ser utilizados sistematicamente.

Por fim, podemos concluir que o desempenho do ambiente de integração Web Banco de Dados envolve muitos aspectos, e a não observância de um deles pode levar ao desenvolvimento de uma solução inviável. A estratégia de manter a sessão contínua do usuário no SGBD é pré-requisito para que se desenvolva uma solução com alto desempenho.

2.4.3.4 Segurança

Mesmo com a crescente utilização da Web como plataforma de desenvolvimento de aplicações comerciais, ainda existem muitos problemas que impedem a implantação do paradigma do comércio eletrônico (e-commerce) neste ambiente.

Um dos principais fatores responsáveis por estes problemas estão relacionados com a questão da segurança, visto que em muitas aplicações da Web, como é o caso das aplicações que envolvam encomenda ou pagamento de bens e serviços eletronicamente, é requerido que as entidades comunicantes, browser e servidor Web, se autentiquem mutuamente e que seja garantida a privacidade das interações. Todavia a atual tecnologia da Web não satisfaz estas necessidades.

Em face da limitação do software da Web com relação a questão da segurança, têm sido criados inúmeros grupos de trabalho, no contexto de empresas com claros interesses comerciais na Web, e outros sobre a alçada do "*Internet Engeneering Task Force"* (IETF) para a resolução deste problema. Várias têm sido as soluções propostas documentadas essencialmente na forma de Internet Drafts, estando algumas já disponíveis comercialmente, e outros em domínio público.

Com o intuito de orientar o desenvolvimento de novos mecanismos de segurança para a Web, em particular para o protocolo HTTP, um grupo de trabalho do IEFT, com a participação de várias pessoas e entidades interessadas no processo, estabeleceu de forma clara um conjunto de requisitos a serem satisfeitos pelas soluções eventualmente propostas [BCD95]. Os requisitos considerados essenciais são:

- *Privacidade nas transações:* deve-se garantir que o conteúdo das transações HTTP efetuadas (Resquest e Response), não possam ser conhecido por terceiras entidades, mesmo tendo acesso ilícito aos canais de comunicação.

- *Autenticação de servidores e serviços:* Um servidor HTTP deve autenticar-se perante os clientes, para que se possa garantir privacidade, e para que estes possam confiar nos dados/objetos enviados pelo servidor.
- *Autenticação de clientes:* um cliente deve autenticar-se perante os servidores, ou seja, apresentar credenciais válidas, a fim destes procederem ao controle de acessos conforme especificado na sua parametrização.
- *Integridade nas transações:* deve-se garantir a integridade dos objetos hipermídia transferidos, e também dos cabeçalhos dos pedidos dos clientes e das respostas dos servidores.

Além destes requisitos básicos podemos citar:

- *Não repudiamento:* consiste em tornar tecnicamente e eventualmente legal, demonstrar que uma dada transação ocorreu de fato entre determinadas entidades, e identificar o seu conteúdo.
- *Certificação de objetos:* suporte para certificação/autenticação da identidade do objeto/documento e garantia da sua integridade.
- *Anonimato dos usuários:* Pode ser de interesse de alguns usuários manter o anonimato nas suas transações, mesmo perante a entidade que providencia o serviço.

As técnicas mais utilizadas para resolver os problemas de segurança em redes de computadores são as técnicas de criptografia. Estas podem ser divididas em: algoritmos de criptografia de chave secreta, algoritmos de criptografia de chaves publicas, funções de *"hash"* seguras e assinaturas digitais [BCD95].

Segundo [Sim95], como propostas mais relevantes para o serviço de segurança na Web, têm: autenticação básica [Luo93], servidor com chave publica [Luo93b], segurança baseada no RIPEM/PGP[NCSA??], Digest Access Authentication[HFH+95], Mediated Digest Authentication[Rag95], Segure HTTP (S-HTTP)[RS94], Shen[Hal??] e Secure Sockets Layer (SSL)[Hic95].

Com relação a questão da segurança no ambiente de integração Web banco de dados, além de levarmos em conta os mecanismos de segurança da Web, devemos considerar as questões relacionadas à segurança e ao acesso ao banco de dados.

O mecanismo de segurança mais utilizado pelos SGBDs é baseado na segurança discriminatória, onde são concedidos privilégios aos usuários a executar determinadas

operações sobre o banco de dados [EN94]. Neste caso o usuário faz acesso ao banco de dados através de um *"login"* e senha e aos objetos do banco através de privilégios (grants) que lhe são concedidos.

Caso o contexto da aplicação seja uma Intranet pressupõem-se que os usuários sejam limitados, assim os mecanismos tradicionais de segurança e acesso ao banco de dados são satisfatórios. Entretanto quando o contexto da aplicação envolve a Internet, os mecanismos tradicionais são ineficientes, pois pode ser inviável controlar o acesso ao banco de dados e aos seus objetos individualmente, devido a perda de desempenho dos mecanismos tradicionais.

Para diminuir o problema, pode-se adotar a estratégia de classificar os usuários segundo grupos com as mesmas características de acesso ao banco de dados e seus objetos, e dar acesso a eles segundo o grupo em que estão inseridos. Perdem-se, entretanto, as características de acesso individual de cada usuário, o que pode ocasionar problemas no caso de auditoria do banco de dados[2], pois fica difícil de determinar quem executadou certas operações.

[BJM+94] apresenta outra solução para este problema utilizando o mecanismo de segurança baseado em papel (role), em que ao cliente do SGBD é atribuído um ou mais papéis, que por sua vez determina as autorizações de acesso para o cliente [Loc88].

Por fim, devemos ressaltar os estudos referentes a execução de transações seguras [Bil98], onde pretende-se estender as propriedades ACID à PLACID (Privacidade, Lealdade, Atomicidade, Consistência, Integridade e Durabilidade), permitindo assim que a execução de uma transação seja confidencial (privacidade) e que haja a autenticação entre os participantes e o não repudiamento das mensagens trocadas (lealdade).

2.4.4 Gateway de Banco de Dados

A função do gateway, também chamados middleware, é realizar o mapeamento entre as requisições do cliente Web e do SGBD. Existem atualmente centenas de gateways de integração Web e banco de dados [Pla96, Int96, Row97] que visam tirar proveito do

[2] Auditoria do banco de dados - Consiste em rever o log para examinar todos os acessos e operações aplicadas no banco de dados durante certo período de tempo.

ambiente aberto e multiplataforma da Web e das facilidades de gerenciamento e mecanismo de otimização para acesso a dados presentes nos SGBDs.

2.4.4.1 Classificação dos Gateways

Os Gateways de banco de dados podem ser classificados dentro de várias categorias dependendo de sua arquitetura de integração. Nesta dissertação adotamos a taxonomia definida em [Lim97], pois é mais abrangente que as demais e permite estudar melhor como as funcionalidades dos atuais SGBDs são afetadas pela arquitetura de integração proposta. O critério desta taxonomia é baseado na localização do software integrador.

A figura 2.4.4.1 mostra o diagrama das taxonomias das arquiteturas dos gateways Web Banco de Dados.

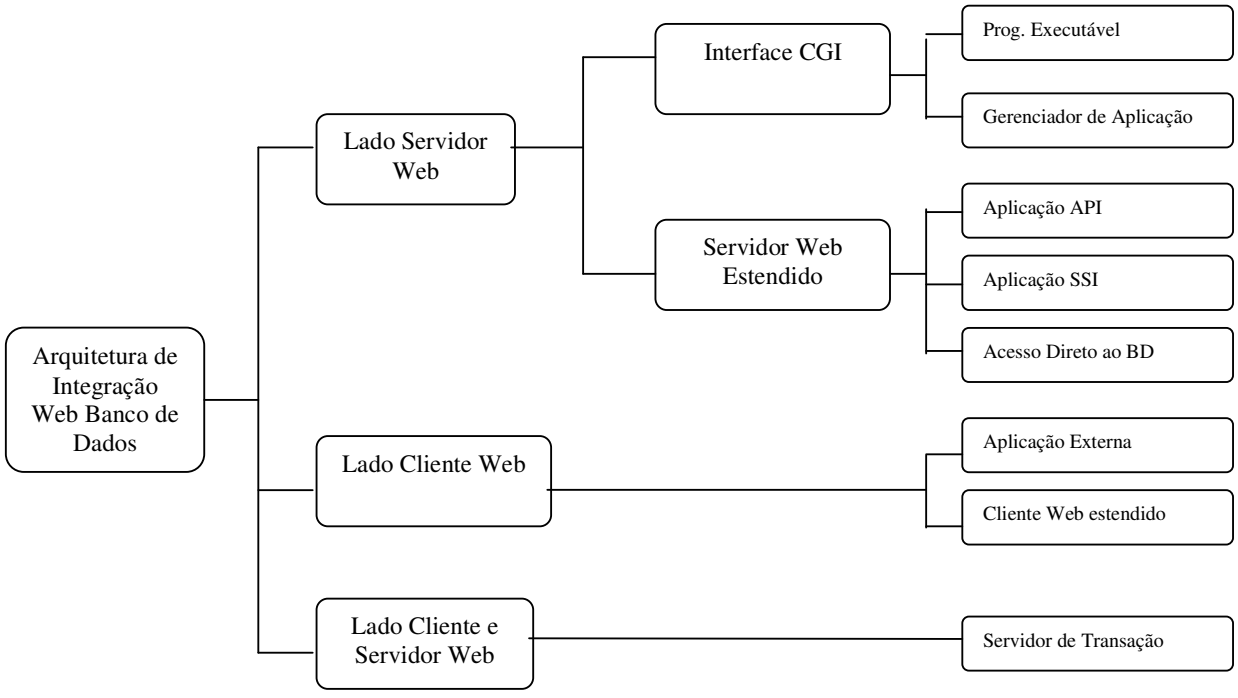

FIGURA 2.4.4.1 - TAXONOMIA DAS ARQUITETURAS DOS GATEWAYS WEB BANCO DE DADOS [LIM97]

Os gateways de integração primeiramente são divididos em três categorias: os que executam no lado servidor Web, os que executam no lado cliente Web e os que executam no lado cliente e servidor.

Os gateways que executam no lado servidor Web podem ser classificados em duas categorias: aqueles que usam a interface CGI padrão e aqueles que usam um servidor Web estendido de forma a suportar características não previstas inicialmente pelo servidor Web padrão. Os que usam a interface CGI pode-se ainda dividi-los nas categorias *Programas Executáveis*, onde o gateway é composto por um ou mais programas executávies CGI ou na categoria *Gerenciador de Aplicação* onde a interface CGI é usada somente para enviar pedidos para um gerenciador de aplicação de banco de dados. Quando o gateway usa um servidor Web estendido pode-se ainda separá-lo em três categorias: *Aplicação API,* onde se utiliza a API do servidor Web para acessar o SGBD; *Aplicação SSI*, onde se utiliza a inserção de códigos especiais (tags) em documentos HTML, que são interpretados e executados pelo próprio servidor Web quando solicitado pelo cliente Web; e *Acesso Direto* ao banco de dados, onde ao servidor Web é inserido o protocolo de comunicação do SGBD.

Os gateways que executam no lado cliente Web podem ser divididos em duas categorias: *Aplicação Externa* ao cliente Web, que se conecta diretamente ao banco de dados; *Cliente Web Estendido*, que incorpora a funcionalidade de acesso ao banco de dados.

Por fim, o gateway que executa no lado cliente e servidor Web é incluído na categoria *Servidor de Transação*, que proporciona uma estrutura de execução de aplicações Web Banco de Dados baseada na extensão das propriedades ACID ao ambiente Web.

2.5 Conclusão

Este capítulo apresentou os principais conceitos e tecnologias envolvidas na Web, nos Bancos de Dados e no ambiente de integração que serão de extrema importância para o bom entendimento desta dissertação.

Também foram discutidos vários problemas existentes no Ambiente de Integração Web Banco de Dados, ilustrando quando possível às soluções existentes.

Capítulo 3

Aspectos Transacionais do Ambiente de Integração Web Banco de Dados

3.1 Problemas transacionais

A natureza "sem estado" ("*stateless"*) das transações Web decorrente do protocolo HTTP usado na comunicação cliente Web e servidor Web é um dos grandes problemas da integração Web e SGBD. Por exemplo, quando o servidor Web responde a uma solicitação do cliente Web, este retorna um documento ou dispara uma aplicação externa via interface CGI. Uma vez que o pedido é atendido a ação é finalizada e a conexão se encerra.

Apesar deste método ser bom para distribuir documentos, pois alivia o servidor Web para o atendimento dos demais pedidos dos clientes. Este método acarreta em vários problemas no projeto de aplicação de banco de dados. Isto é decorrente do fato de que os atuais SGBDs foram desenvolvidos para atender a requisições de usuários conectados ao banco de dados em sessões contínuas, ou seja, o usuário fica conectado ao SGBD enquanto durar sua sessão. A mudança para um ambiente baseado em um protocolo "*stateless"* , como o da Web, implica em vários questionamentos, como por exemplo, a definição formal de transação Web banco de dados, o gerenciamento da execução de transações neste ambiente e a necessidade de verificar, se o modelo de transações usado nos atuais SGBDs é adequado ao ambiente Web banco de dados.

3.2 Transação Web Banco de Dados

Com relação à definição de uma transação no contexto Web, pode-se ressaltar as seguintes particularidades:

- *Inicia-se no cliente Web*, podendo estar em qualquer lugar do mundo. Assim, se usarmos os padrões estabelecidos para o ambiente Web, que neste caso é o protocolo HTTP, haverá necessariamente uma descontinuidade na sessão do usuário entre o cliente Web e o servidor de banco de dados.
- *São formuladas por meio da interação com o usuário*, podendo ter tempo "infinito" se comparadas com a duração do tempo das transações das aplicações tradicionais. Por exemplo, o usuário pode abandonar a navegação sem notificar o servidor Web ou o SGBD, ou ainda, pode navegar por outras páginas que em princípio não fazem parte do contexto transacional, podendo mais tarde retornar à transação que estava em andamento.
- *São baseadas em páginas Web*, ou seja, mesmo existindo vários comandos a realizar no banco de dados em uma única página, seria desejável que todas as operações fossem consideradas como uma unidade atômica de processamento.
- *Podem ser resultantes de várias páginas Web de interação com o usuário.* Isto indica que uma transação na Web pode ser constituída de várias páginas Web.
- *São baseadas no modelo navegacional da Web*, não havendo a priori como impor um rígido controle na ordem das operações da transação. Por exemplo, o usuário pode submeter uma transação e logo em seguida retornar à mesma transação por meio da opção *back* do navegador e re-executá-la.

Diante de todas essas particularidades, podemos observar que o conceito tradicional de transações não se adapta as características das transações Web banco de dados, já que estas são baseadas no modelo navegacional hipertexto, compostas por uma ou mais páginas Web e com uma possível interação de longa duração com o usuário.

3.3 Escopo de uma Transação Web banco de dados

A maioria dos SGBDs existentes segue o modelo de transações planas, assim chamado, por apresentar uma única camada de controle pela aplicação. A expansão desse modelo para o ambiente de integração Web banco de dados deve levar em consideração o escopo ou abrangência de uma transação nesse ambiente.

Como as páginas são as unidades básicas de interação com o usuário, então deve existir uma relação entre essas unidades e o conceito de transação no ambiente de integração. Esta relação pode ser discutida tendo em vista o escopo das transações Web banco de dados.

3.3.1 Uma página como unidade atômica

A estrutura mais simples de uma transação Web é aquela onde uma página é composta por uma ou mais operações sob o SGBD. Como uma página é considerada a unidade básica de escopo, então, caso a página executar um conjunto de operações no SGBD, é desejável que estas operações sejam consideradas como unidade atômica: ou todas as operações são executadas ou nenhuma delas.

A figura 3.3.1 ilustra uma página Web composta por várias operações a serem efetivadas pelo SGBD. Neste caso, as operações contidas na página são consideradas com a unidade básica de processamento para o SGBD.

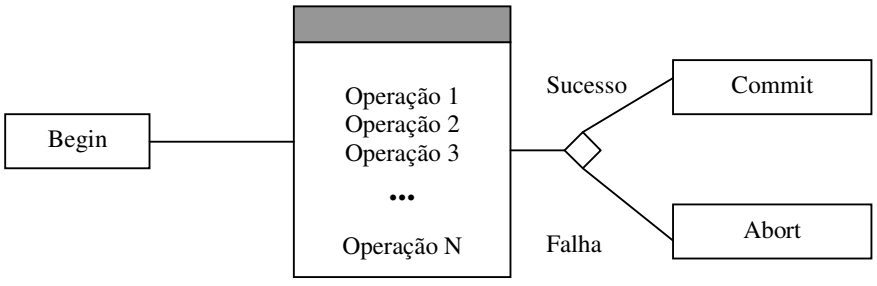

FIGURA 3.3.1 - TRANSAÇÃO COMPOSTA POR UMA PÁGINA WEB

3.3.2 Várias páginas como uma unidade atômica

Para aumentar o desempenho global é desejável limitar o escopo de uma transação Web banco de dados a uma única página Web, já que é minimizado o tempo de duração dos bloqueios aos dados no SGBD, favorecendo assim a concorrência. Entretanto, os projetistas de aplicação freqüentemente precisam que as várias interações do usuário sejam realizadas como se fosse uma única transação atômica.

Considerando que no contexto da Web cada interação corresponde a uma página, então é desejável que várias páginas façam parte uma única transação global. Assim deve-se expandir o escopo de uma transação Web banco de dados de forma a se aceitar várias páginas Web como uma unidade atômica no SGBD.

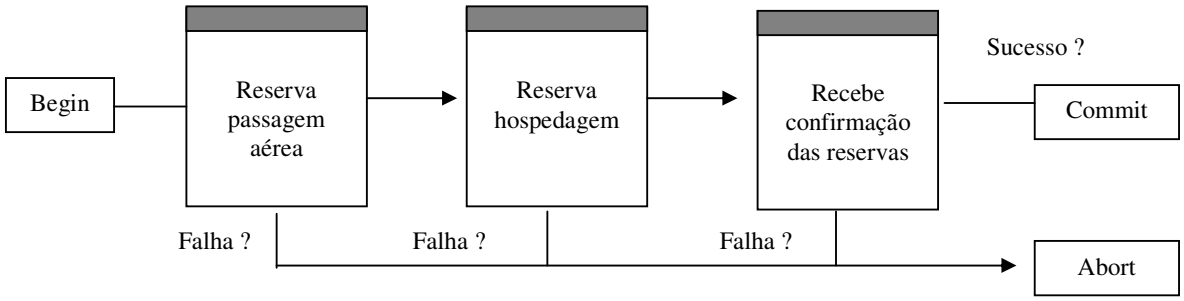

FIGURA 3.3.2 - TRANSAÇÃO COMPOSTA POR VÁRIAS PÁGINAS WEB: RESERVA DE VIAGEM

A figura 3.3.2 apresenta uma seqüência de páginas Web que devem ser manipuladas no contexto de uma transação atômica. Inicialmente, o usuário reserva uma passagem para o vôo desejado, em seguida reserva sua hospedagem e por fim recebe a confirmação de suas reservas e a ordem de pagamento com o total a ser pago, finalizando assim a transação.

3.3.3 Paginas que não fazem parte de uma transação

A possibilidade de delimitar um conjunto de páginas para compor uma transação é útil para resolver o problema de transações que exijem várias interações com o usuário, mas pode ser muito restritiva no ambiente Web.

Imagine situações em que a navegação hipertexto possibilite, durante a execução de uma transação composta por várias páginas Web, o usuário sair do escopo transacional, para realizar, por exemplo, consultas contidas em páginas que em princípio não fazem

parte do contexto da transação sendo submetida. A figura 3.3.3 mostra um exemplo de uma transação com estas características.

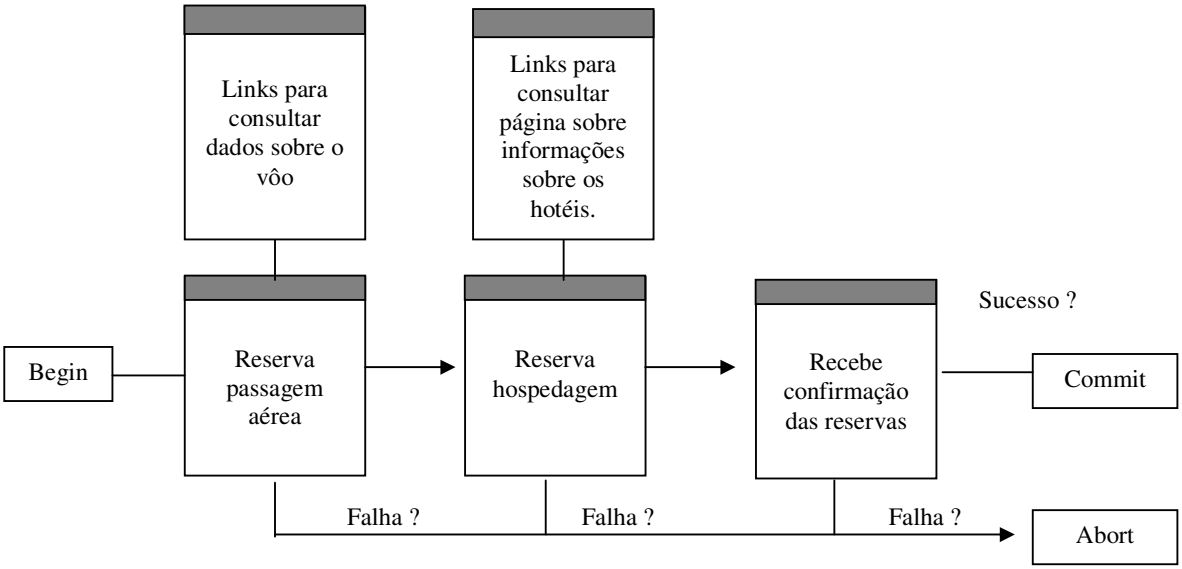

FIGURA 3.3.3 - TRANSAÇÃO COMPOSTA POR PÁGINAS TRANSACIONAIS E NÃO TRANSACIONAIS

Neste exemplo, podemos observar que além das páginas correspondentes à transação, temos outras que não pertencem ao contexto transacional, podendo inclusive ser páginas estáticas armazenadas no sistema de arquivos na plataforma do servidor Web. Então podemos identificar dois tipos de páginas no ambiente Web: aquelas que devem ser marcadas como transacionais e as que não precisam ter esta característica.

3.4 Consistência

Como vimos anteriormente, consistência de uma transação é simplesmente sua corretude, ou seja, uma transação é um programa correto que faz o mapeamento de um estado consistente para outro [OV91]. Os mecanismos presentes nos SGBDs sempre garantem o estado consistente do banco de dados ao final da transação de acordo com a modelagem do banco de dados previamente definida.

Entretanto, a orientação a páginas de dados do ambiente Web, ao contrário da orientação a conjunto de tuplas dos SGBDs, pode originar sérios problemas de consistência

das transações Web banco de dados. Isto se deve principalmente à capacidade do usuário voltar ou avançar (*back* / *forward*) pelas páginas de interação da aplicação em execução, utilizando o cache local. A seguir são apresentados alguns exemplos que ilustram o problema e suas implicações na questão da consistência das transações:

1. **Inserção de tupla:** Um exemplo típico deste caso acontece quando o usuário utiliza a opção de *back* do navegador após inserir uma tupla, e re-submete a mesma operação de inclusão com os mesmos dados. Se por acaso a chave primária da tupla for gerada automaticamente pela aplicação, pode acarretar um estado indesejado no banco de dados com duas informações iguais armazenadas no banco de dados, embora com chaves primárias distintas.
2. **Exclusão de tupla:** suponha, por exemplo, que o usuário exclua uma tupla por meio de uma interação e a re-submeta com a opção de *back* do navegador. Como não existe mais a tupla, então um erro de falha da transação vai ser retornado pelo SGBD, o que é indesejável.
3. **Atualização de tupla:** o problema da atualização é o mais crítico, pois podem existir casos onde são atualizadas várias tuplas, cujos danos podem ser difíceis de serem revertidos futuramente. Um exemplo é as aplicações que envolvam atualizações monetárias de produtos de uma empresa sob o percentual definido pelo usuário da aplicação. Se o usuário submeter a aplicação e retornar a operação via opção *back* e em seguida re-submetê-la, os dados atualizados serão modificados duas vezes ao invés de uma.
4. **Páginas inconsistentes:** este problema é causado porque o cliente Web possui um cache local para permitir o melhor desempenho do sistema. Entretanto esta característica cria um problema potencial para aplicações cujas páginas são gerados dinamicamente como nas aplicações Web banco de dados. Assim podem existir situações em que o cache local está inconsistente em relação à informação de fato disponível no banco de dados. É o caso, por exemplo, de consultas submetidas a banco de dados atualizados com muita freqüência.

Outro problema de consistência de transações Web banco de dados decorrente da navegação Web diz respeito à possibilidade de se cancelar uma operação solicitada ao servidor Web ainda em andamento, através da opção *stop* presente nos navegadores atuais.

Esta opção permite uma semântica de cancelamento de transações por parte dos usuários que é difícil de ser implementada no ambiente de integração. De fato, uma vez submetida uma transação Web banco de dados, não será possível cancelá-la através de uma ação explícita do usuário usando a opção *stop* do navegador já que estão envolvidos dois servidores na transação: o servidor Web e o servidor de banco de dados. A transação sempre será realizada se não houver nenhuma falha durante sua execução. Caso o usuário selecione a opção *stop* dos navegadores durante a execução da transação, o máximo que pode ocorrer é o não recebimento da resposta da transação enviada pelo gateway através do servidor Web, embora a transação tenha sido completada pelo servidor de banco de dados. Desta forma, a semântica proporcionada pelos navegadores Web para cancelar uma solicitação via opção *stop* não é adequada, pelo menos atualmente, sob o ponto de vista de transações Web banco de dados.

3.5 Controle de concorrência e isolamento

Como destacamos, a propriedade de isolamento das transações requer que cada transação veja o banco de dados consistente todo tempo. Em outras palavras, a execução de uma transação não pode interferir no resultado de outras transações concorrentes antes que estas validem. Para garantir esta propriedade, os SGBDs são dotados de técnicas que garantem a seriabilidade das transações concorrentes, questão já amplamente difundida na literatura [GR93, EN94]. Um dos principais métodos para garantir a seriabilidade de execução de transações concorrentes é baseado na técnica de bloqueios, onde se impede que várias transações acessem a um mesmo item de dado de forma concorrente. Esta técnica será tomada como base para a análise dos problemas de controle de concorrência no ambiente de integração aqui discutida.

No ambiente de integração, a propriedade de isolamento e o controle de concorrência estão intimamente relacionados com a quantidade de páginas que compreendem a transação Web. Assim, vamos analisar estas propriedades no caso de uma transação ser formada por apenas uma página e no caso dela ser formada por mais de uma página.

No caso da transação ser composta por apenas uma página atômica, a técnica de bloqueio aos dados implementada pelos SGBDs é válida, pois como não existe o problema da perda da conexão, que ocasionaria a perda dos bloqueios, e o tempo de execução é

curto, então estas características assemelham-se ao modelo atual implementado nos SGBDs, não precisando de qualquer mudança.

O problema surge quando várias páginas originam uma transação atômica. Imagine a situação onde não há uma sessão contínua do usuário com o SGBD e uma transação composta por várias páginas de interação. Como a conexão não é mantida, os bloqueios a cada interação são perdidos, não garantindo assim a propriedade de isolamento. Este problema é comum no caso de atualização de dados, onde em uma página o usuário seleciona o item de dado e em outras efetua a atualização. Como não é mantido o bloqueio do item de dado, outro usuário pode recuperar a mesma informação para atualização, acarretando assim em um estado inconsistente. Portanto, para que a propriedade de isolamento seja amplamente suportada no ambiente de integração, é necessário que haja a manutenção da conexão.

Entretanto, mesmo que exista uma sessão contínua do usuário com o SGBD, as técnicas atuais para controle de concorrência dos SGBDs baseadas em bloqueio podem se tornar inadequadas no ambiente de integração. Por exemplo, no caso de uma operação solicitar bloqueio exclusivo aos dados numa página Web de interação e o usuário demorar a submeter às próximas páginas da transação para a sua finalização, o mecanismo de *timeout* do SGBD, que garante bloqueio aos dados somente por um tempo determinado, vai ser acionado e a transação em andamento será desfeita, liberando os dados as outras transações concorrentes. Não só o usuário pode ter um tempo de resposta grande, mas o tráfego na rede ou uma sobrecarga no servidor Web podem ser fatores determinantes no tempo final da transação. O SGBD poderá decidir por abortar a transação tornando a operação inviável, principalmente se a aplicação estiver sob a Internet, onde o tempo de resposta tende a ser ainda maior do que numa Intranet.

Além disso, deve-se também garantir que a navegação via opção *back/forward* dos navegadores não gere situações indesejáveis do ponto de vista do controle de concorrência. Por exemplo, situações em que um usuário submete uma página que solicita um bloqueio a um item de dado e, logo em seguida, o usuário re-submete a mesma página via opção *back* do navegador.

3.6 Recuperação

A recuperação garante que, mesmo havendo falhas durante a execução das transações, o banco de dados sempre será restaurado para um estado consistente.

Uma transação pode falhar devido a vários erros [KS97]:

- erros de hardware em dos componentes do sistema;
- erros na comunicação dos componentes do sistema;
- erros lógicos;
- erros do sistema como na detecção de deadlocks.

Para impedir que estes erros acarretem em um estado inconsistente no banco de dados, são ativados mecanismos de recuperação do sistema. Nos atuais SGBDs este é um problema já bem estudado e resolvido.

Entretanto, quando voltamos o contexto para o ambiente de integração Web banco de dados, além das falhas do SGBD, temos vários outros pontos de falha que merecem destaque. [Lim97] ressalta sete pontos de falhas possíveis durante a execução de uma transação.

1. Falha no cliente Web;
2. Falha na rede entre o cliente Web e o servidor Web;
3. Falha no servidor Web;
4. Falha na conexão entre o servidor Web e gateways;
5. Falha no gateway;
6. Falha na conexão entre o gateway e o SGBD;
7. Falha no SGBD.

A figura 3.6 ilustra estes pontos de falha.

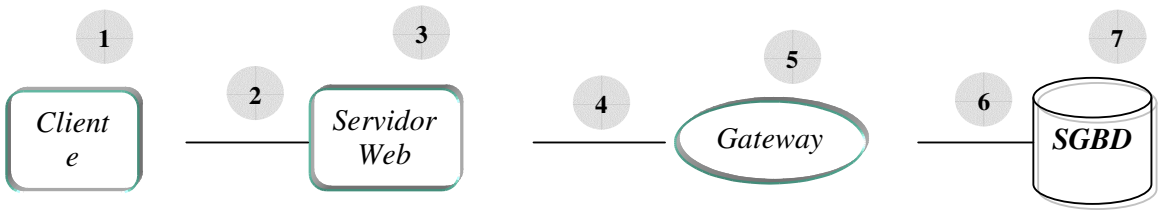

FIGURA 3.6 – PONTOS DE POSSÍVEIS FALHAS NO AMBIENTE WEB BANCO DE DADOS

Além dos sete pontos de falhas, devemos levar em consideração para implementação do processo de recuperação da transação, a semântica dos estados da transação.

[Ora96] destaca três estados possíveis durante a execução de uma transação:

1. *Transação iniciada, mas não confirmada:* o usuário solicitou o início de uma transação e nenhuma resposta de confirmação foi recebida. Neste caso um erro pode ter ocorrido em qualquer dos sete pontos.
2. *Transação em processo:* o usuário recebeu a confirmação de que uma transação se iniciou e está em execução, portanto, a sessão para ele será mantida pelo gateway junto ao SGBD. Nenhuma confirmação de fim de transação foi solicitada.
3. *Transação finalizada, mas não confirmada:* o usuário decidiu confirmar o fim da transação, mas nenhuma resposta de confirmação foi recebida. Neste caso o usuário fica, em princípio, sem saber se sua transação foi ou não efetivada no banco de dados

Como atualmente os clientes e servidores Web não estão habilitados a tratar de falhas que possam ocorrem no ambiente Web, então para que a capacidade de recuperação seja válida no ambiente de integração, é necessário que essa capacidade seja tratada no gateway responsável pela integração das duas tecnologias sem, entretanto, desconsiderar os mecanismos de recuperação presentes no atuais SGBDs.

3.7 Conclusão

A partir de uma avaliação cuidadosa das transações Web banco de dados, podemos concluir que os mecanismos tradicionais empregados para a verificação da execução serial das transações em SGBDs tradicionais podem não ser apropriados ao ambiente Web banco de dados. Assim, com relação ao critério de corretude e seriabilidade, devemos levar em consideração os seguintes aspectos:

- Ser orientado a páginas Web. Pois uma página pode conter várias operações no Banco de dados ou ser parte de uma transação única.
- Ser baseada num controle de concorrência para interação com transações de

longa duração.

- Ser flexível o suficiente de forma a permitir a combinação de várias páginas Web para a execução de uma única transação.
- Ter mecanismos para recuperação por falhas na comunicação cliente Web, servidor Web e servidor de Banco de dados para garantir a integridade das transações Web banco de dados.

Capítulo 4

Modelos de Transações

4.1 Introdução

Um modelo de transações define quando uma transação começa, quando ela finaliza e que ações são tomadas no caso de falhas [OV91].

O modelo tradicional de transações adotado pela grande maioria dos SGBDs e Gerenciadores de Transação é o modelo de transações planas, assim chamado por apresentar uma única camada de controle. Uma transação neste modelo é caracterizada por apresentar uma estrutura simples e pela impossibilidade de fazer referência a resultados intermediários. Além disso, o tempo de execução dessas transações é geralmente curto.

Durante anos, este modelo foi a base para o desenvolvimento das aplicações. Entretanto, na medida em que foram surgindo novas tecnologias e novos requisitos, as aplicações passaram a solicitar das transações novas características. Como por exemplo, a capacidade de fazer referência a resultados intermediários e cooperação entre as transações.

Motivados pelo intuito de adicionar novas características às transações, começaram a surgir novos modelos de transações [Elm90], tendo como finalidade básica dar suporte as necessidades de determinados tipos de aplicações.

Neste capítulo vamos apresentar vários destes modelos, enfatizando suas características e o tipo de aplicação a que se aplicam.

4.2 Modelo de Transações Aninhadas

O Modelo de Transações Aninhadas (Nested Transactions Model) ou Modelo de Transações Aninhadas Fechado foi introduzido por Moss em [Mos81]. Uma transação neste modelo pode conter qualquer número de subtransações e cada subtransação, por sua vez, pode ramificar-se em outras subtransações. Assim uma transação completa forma uma árvore de (sub)transações, chamada de Árvore Transacional. A figura 4.2. ilustra a estrutura de uma transação neste modelo e seus componentes.

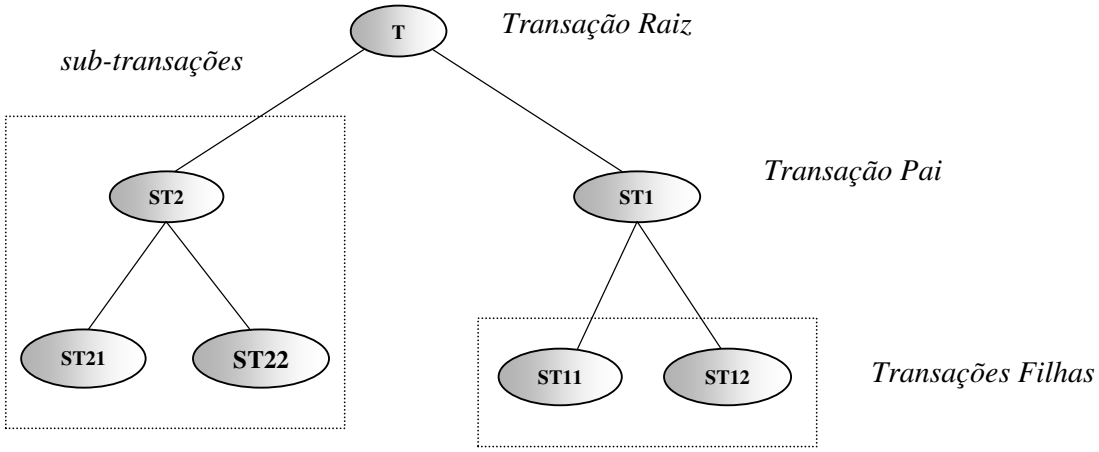

FIGURA. 4.2 - TRANSAÇÃO ANINHADA E SEUS ELEMENTOS

A raiz da Árvore Transacional é chamada "root" ou nível de topo transacional. Transações que tem subtransações são chamadas de "pais", e suas subtransações de "filhas". Transações que possuem o mesmo ascendente são chamadas de transações "irmãs". As transações que estão entre uma transação e o topo são chamadas de superiores. Uma transação filha deve iniciar depois de sua transação pai e terminar antes dela.

A raiz de uma transação satisfaz todas as propriedades do modelo tradicional, tais como: Atomicidade, Consistência, Isolamento e Durabilidade. Assim, as raízes das transações devem ser isoladas uma das outras e no caso de falha, elas devem ser desfeitas sem afetar o resultado das outras. A propriedade de Atomicidade é válida para as subtransações (elas podem validar ou abortar independentemente). Uma transação não pode ser validada até que todas as suas filhas tenham sido terminadas. O aborto de uma

transação filha não implica no aborto da transação pai. Neste caso a transação pai pode agir de várias formas, tais como:

- Ignorar o erro (Caso da transação filha ser não-vital[3]).
- Desfazer a subtransação.
- Iniciar outra subtransação (Uma transação contigente[4] que implemente uma ação alternativa).
- Abortar a transação.

Se a transação pai abortar, todas as subtransações serão desfeitas. As atualizações de uma subtransação serão permanentes somente quando o topo da transação finalizar.

O algoritmo de controle de concorrência utilizado neste modelo é baseado no protocolo de bloqueio em duas fases [OV91], no qual uma transação não pode liberar qualquer bloqueio até o ponto de validação ou Rollback da transação [Mos81]. As filhas de uma transação pai são sincronizadas, como se fossem transações separadas. Quando uma filha valida, seus bloqueios são herdados pelo seu pai. Quando o pai obtém um bloqueio, este não interfere na execução de seus filhos, somente interfere nas transações que não são suas descendentes. Uma transação pode obter um bloqueio se qualquer outra transação que tenha bloqueio conflitante seja pai da transação. Na herança de um bloqueio, se ambos pai e filho obtém o bloqueio para o mesmo item de dado, o pai escolhe obter o bloqueio em modo mais exclusivo. Este protocolo permite o acesso compartilhado de dados de forma a garantir a propriedade da seriabilidade. Por causa da herança do bloqueio, a árvore da transação completa é isolada das outras árvores.

As principais vantagens do modelo de transações aninhadas são: modularidade, tolerância a falhas e paralelismo intra-transações.

- A modularidade é obtida pela decomposição da transação hierarquicamente.
- Mecanismo de tolerância a falhas deste modelo permite o usuário definir mais gradualmente a unidade de reabilitação.

[3] Transação Não-Vital – é aquela transação onde o seu aborto não impede que sua transação pai continue executando.
[4] Transação Contigente – é aquela transação que tem a função de substituir uma outra transação que foi abortada.

- Um maior grau de paralelismo intra-transações é obtido, porque as subtransações podem ser executadas concorrentemente.

As transações aninhadas são utilizadas em aplicações com estruturas complexas por possuir um melhor refinamento no nível de escopo do *"rollback"*, devido a sua estrutura hierárquica. Estas também são utilizadas em projetos de engenharia de software, devido a semelhança entre a estrutura hierárquica de uma transação aninhada e a de um programa.

4.3 Modelo de transações SAGAS

O modelo Sagas foi proposto por Garcia-Molina e Salem em [GMS87], tendo como ponto chave o conceito de transações compensatórias [Gra81]. Uma transação C é dita compensatória de T, se corresponder semanticamente ao efeito do "undo" de T, depois de T ter sido validada. Este modelo tem como finalidade resolver o problema de transações de longa duração.

Uma transação Saga é uma transação de longa duração que consiste de um conjunto de passos relativamente independentes, T1, T2, ..., Tn. Associado a cada subtransação T1, T2, ..., Tn estão as subtransações compensatórias C1, C2, ..., Cn. Para executar uma Saga, o sistema deve garantir que a seqüência:

T1, T2, ..., Tn

Ou a seqüência:

T1, T2, ..., Tj, Cj, ..., C2, C1, para 1<= j<=n irá ser executada.

Isto é, ou todas as subtransações na Saga são completadas ou qualquer execução parcial será desfeita pelas subtransações compensatórias.

Cada subtransação na Saga não necessita observar o mesmo estado consistente da base. Assim quando uma subtransação é validada, ela pode validar sem esperar as outras subtransações da mesma Saga, e liberar seus resultados para outras transações concorrentes.

Se algumas das subtransações da Saga falhar, ela pode continuar a execução chamando outra subtransação, (Forward Recovery). No caso do Forward Recovery não puder ser executado, a transação deve ser abortada pela execução das transações

compensatórias para cada subtransação já validada (Backward Recovery).

Com relação às propriedades ACID, o modelo Sagas relaxa a propriedade de isolamento permitindo a Saga revelar seus resultados parciais para outras transações antes que ela complete. A consistência (seriabilidade) pode ser comprometida, pois a Saga pode intercalar suas subtransações em qualquer ordem com outras subtransações de outras Sagas. O modelo Sagas, entretanto, preserva as propriedades de atomicidade e durabilidade, pois requer que todas as subtransações de uma Saga sejam completadas com sucesso ou nenhuma delas é validada. O resultado da Saga uma vez validada deve persistir mesmo que ocorram falhas.

O modelo Sagas é útil somente quando as subtransações na Saga são relativamente independentes (devido aos problemas de consistência), e possam ser sucessivamente compensadas. Porém na vida real, existem transações que não são compensáveis. Por exemplo, a transação que libera o dinheiro no caixa eletrônico não é compensável, pois após liberar o dinheiro, não é mais possível mandar a pessoa devolvê-lo.

A principal contribuição deste modelo é a incorporação de transações compensatórias dentro do modelo de transações.

4.4 Cooperative Transaction Hierarchy (Hierarquia de Transações Cooperativas)

O modelo Cooperative Transaction Hierarchy [NZ84] foi introduzido por Nodine e Zdonik para suportar aplicações cooperativas como CAD.

Para eliminar o problema do não compartilhamento de dados comuns (chave para operações cooperativas) entre transações no modelo tradicional, Nodine e Zdonik propõem estruturar uma aplicação cooperativa como um nó de uma árvore. Os nós externos da hierarquia representam as transações associadas como um determinado projetista. Os nós internos são chamados grupos de transações, e contém o conjunto de membros que cooperam para a execução de uma simples tarefa. O termo transações cooperativas neste modelo refere-se às transações com o mesmo pai na árvore transacional. A figura 4.4 ilustra os membros deste modelo através de sua utilização no projeto de um carro.

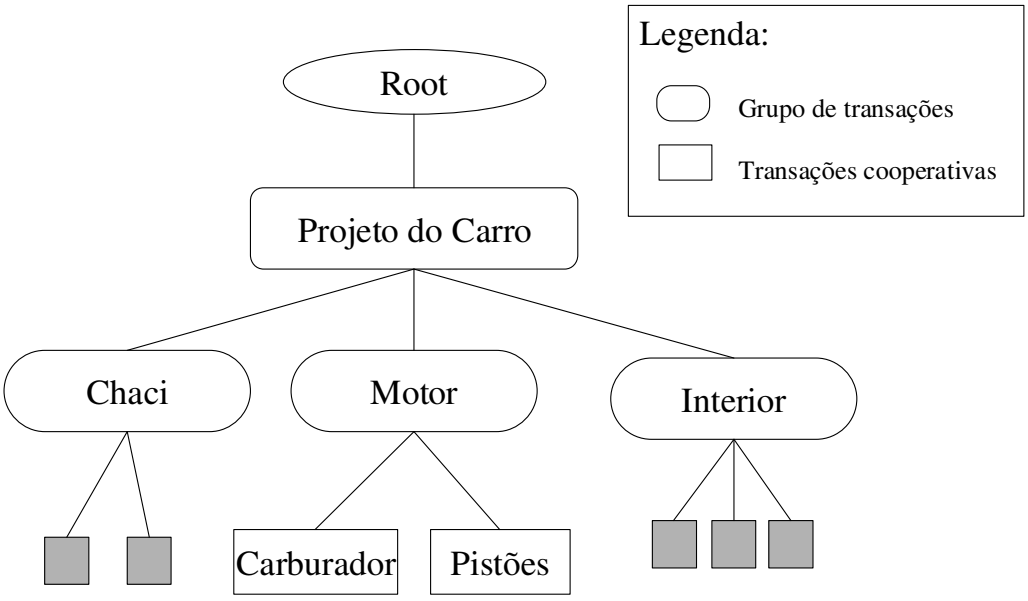

FIGURA 4.4 – DECOMPOSIÇÃO HIERÁRQUICA NO PROJETO DE DESENVOLVIMENTO DE CARROS

As transações cooperativas não necessitam ser serializáveis. Em vez disso, o grupo de transações destas define um conjunto de regras que regulam o modo de interação entre elas. As regras são definidas em termos de um conjunto de autômatos de estado-finito. Um autômato de estado-finito especifica, para cada conjunto de objetos, as operações permitidas a cada transação cooperativa, e os modos permitidos de intercalar as operações entre as transações do grupo.

A principal contribuição deste modelo é a substituição da noção de corretude definida a partir da seriabilidade pela noção de corretude definida pelo usuário.

4.5 Cooperative SEE Transaction (Transação SEE Cooperativa)

O modelo Cooperative SEE Transaction [NZ90] foi desenvolvido para suportar trabalhos cooperativos em ambientes de desenvolvimento de software (SEE). Neste ambiente é necessário o suporte de transações de longa duração e transações cooperativas.

Este modelo usa transações aninhadas ativas e critério de corretude definido pelo usuário. Em virtude do critério de corretude ser definido pelo usuário, é possível: incrementar o nível concorrência; aumentar o grau de cooperação entre diferentes

transações; permitir vários níveis de cooperação; e suportar transações de longa duração. Tal modelo, porém não suporta as propriedades ACID.

O usuário define as condições de corretude baseado nos seguintes itens:

- *Conflicts*: operações que não são permitidas executarem concorrentemente.
- *Patterns*: Seqüência de funções que devem ser executadas.
- *Triggers*: ações que são tomadas quando uma requisição começa e finaliza.
- *Commit or Abort Semantics*: ações que devem ser tomadas quando uma transação finaliza.

4.6 DOM Transactions

O modelo de transações DOM foi projetado para dar suporte ao Projeto de Gerenciamento de Objetos Distribuídos (DOM) desenvolvido pelo Laboratório GTE.

O objetivo deste projeto é suportar o desenvolvimento de aplicações em um ambiente orientado a objetos distribuídos, no qual os componentes que integram o sistema podem ser autônomos e heterogêneos.

O modelo de transações DOM permite tanto transações aninhadas abertas[5] quanto fechadas, bem como a combinação das duas.

Transações compensatórias podem ser especificadas para desfazer o efeito de uma transação validada, e transações contigentes no caso de falha uma das subtransações. As subtransações podem ser especificadas como vital ou não-vital. Se uma subtransação vital abortar, sua transação pai deve ser abortada. Porém, se uma subtransação não-vital abortar, sua transação pai pode continuar a execução.

As subtransações podem ser executadas concorrentemente, porém neste modelo pode-se especificar dependências entre elas, ou seja, pode-se especificar uma ordem de execução.

A teoria da corretude do modelo DOM não foi ainda desenvolvida. A dificuldade é

que os componentes do sistema podem não garantir as propriedades ACID. Além disso, a autonomia entre os componentes dificulta a execução de transações globais.

4.7 Transaction Model for an Open Publication Environment (Modelo de Transação para um Ambiente de Publicação Aberto)

O Modelo de Transação para um Ambiente de Publicação Aberto [MRKN91] foi desenvolvido para o ambiente de publicação distribuída aberta. Um ambiente deste tipo é um sistema distribuído que combina vários componentes do sistema, incluindo SGBD, base de conhecimento e componentes ativos do sistema. Este ambiente inclui uma variação entre transações de longa e/ou baixa duração, e níveis de alta e/ou baixa cooperação entre elas.

Este modelo propõe transações aninhadas abertas com as seguintes características:

- Redução dos conflitos entre as operações do usuário, pelo uso da semântica da informação para a definição do conflito.
- Facilidade de integração de ferramentas existentes.
- Garantia das propriedades ACID, caso sejam suportadas pelas ferramentas.

Para manter a consistência, já que no modelo de transações aninhadas aberta o resultado da validação das subtransações pode ser visto por outras transações, somente aquelas (sub) transações que comutam com a subtransação validada podem ver o seu resultado. Duas transações comutam se o resultado da execução de ambas é independente de sua ordem de execução.

4.8 Contract Model (Modelo de Contrato)

O modelo Contract foi proposto por Reuter [Reu89] para definir e controlar as

[5] Transações Aninhadas Abertas: Em contraste ao modelo de transações aninhadas tradicional (fechado), as transações aninhadas abertas tornam as mudanças das subtransações visíveis no final da subtransação, e não no final da transação completa. Mas as mudanças não são visíveis para todas as outras transações.

operações complexas e de longa duração existentes em aplicações não-convencionais como automatização de escritórios e CAD.

Uma transação neste modelo é definida como um conjunto de ações pré-definidas, com a especificação explicita do controle de fluxo entre elas [RW91].

A principal ênfase do modelo Contract é que a execução de uma transação Contract deve ser "forward–recoverable". Isto significa que quando a execução de uma transação é interrompida por falha, deve ser instanciada e continuada de onde foi interrompida. Para que isto seja possível, todos os estados da informação, incluindo o estado do banco de dados, das variáveis de cada passo da aplicação, e o estado global do Contract, devem ser armazenados.

Como no Sagas, uma transação neste modelo permite tornar os resultados visíveis antes de ser finalizada.

Transações compensatórias são usadas para desfazer os resultados validados que não são necessários.

Este modelo usa restrições de consistência no banco de dados para validar o controle de concorrência.

Por fim, o Contract permite resolver conflitos de forma mais flexivel, especificando o que fazer quando o conflito ocorrer.

4.9 Split Transaction

Split Transaction foi proposto em [PKH88] para suportar aplicações *"open-ended"*. Exemplos deste tipo de aplicações são CAD/CAM, desenvolvimento de software e projeto de circuitos. As aplicações *"open-ended"* são caracterizadas por:

- Tempo de duração incerto;
- Ações não previstas;
- Interação com outras atividades concorrentes.

O princípio das transações splits é a divisão de uma transação dentro de duas outras transações seriáveis e de seus recursos entre os resultados das transações.

Para dividir uma transação T em duas transações A e B tal que A é seriável antes de B, as seguintes propriedades devem ser garantidas:

1. AwriteSet $\cap$ BwriteSet $\subseteq$ BwriteLast
2. AreadSet $\cap$ BwriteSet = $\varnothing$
3. BreadSet $\cap$ AwriteSet = ShareSet , onde

   AwriteSet = Conjunto de objetos que sofreram a operação de escrita (Write) durante a execução da transação A.

   BwriteSet = Conjunto de objetos que sofreram a operação de escrita (Write) durante a execução da transação B.

   BwriteLast = Subconjunto dos objetos de BwriteSet que sofreram a última operação de escrita (Write) durante a execução da transação B.

   AreadSet = Conjunto de objetos que sofreram a operação de leitura (Read) durante a execução da transação A.

   BreadSet = Conjunto de objetos que sofreram a operação de leitura (Read) durante a execução da transação B.

   ShareSet = Conjunto de objetos que são compartilhados por A e B, ou seja, são acessados tanto por A quanto por B.

A propriedade 1 estabelece que se A e B escrevem dentro do mesmo objeto, as operações de escrita de B devem seguir as operações de escrita de A. Em outras palavras, A não pode sobrescrever as saídas de B, mas B pode sobrescrever as saídas de A.

A propriedade 2 diz que A não pode ver nenhum resultado de B.

A propriedade 3 diz que B pode ver os resultados de A.

Estas propriedades, juntas, garantem que A é seriável antes de B. Se ambos os conjuntos ShareSet e BwriteLast são vazios (chamado de caso independente), então A e B podem ser executados em qualquer ordem e ambos podem validar independentemente.

Entretanto, se os conjuntos ShareSet e BwriteSet não são vazios (chamado caso serial), então A é seriável antes de B por causa da dependência dos dados acessados. Neste caso, os objetos no conjunto ShareSet não devem ser mudados em A, caso contrário, B irá

finalizar usando dados de A não validados. Além disso, se A abortar, B deve abortar, porque B lê os dados escritos por A.

O principal propósito das transações split é finalizar uma das transações divididas e liberar resultados úteis da transação original, enquanto a outra transação dividida continua.

Duas vantagens deste modelo são:

1. Reabilitação adaptativa: as partes validadas de uma transação não serão afetadas por falhas posteriores.
2. Redução do isolamento: liberando recursos das partes validadas da transação.

4.10 Flex Transaction Model (Modelo de transações Flex)

O modelo Flex Transaction [ELLR90] foi desenvolvido para permitir maior flexibilidade no processamento das transações. Neste modelo uma transação é considerada como um conjunto de tarefas. Para cada tarefa, o modelo permite o usuário especificar um conjunto de subtransações funcionalmente equivalentes. A execução da transação flex tem sucesso se todas as tarefas forem validadas.

Este modelo é resistente a falhas no sentido em que a transação flex pode prosseguir e validar se algumas das subtransações falhar.

O modelo também permite especificar dependências entre as subtransações da transação flex. Algumas destas dependências são: a dependência de falha, dependência de sucesso e dependência externa. As duas primeiras dependências estão relacionadas com a ordem de execução das subtransações e a última relaciona-se com a ocorrência de eventos externos a transação.

Esse modelo permite a utilização de transações compensatórias para o usuário melhor controlar o nível de isolamento.

4.11 Transaction Tool Kits (Kit de ferramenta para transações)

O modelo Transaction Tool Kits foi proposto por Unland e Schlageter [US91], para tentar modelar e controlar as transações que têm requisitos de conflitos nas suas subtransações, ou seja, as subtransações podem apresentar comportamentos (características) diferentes da transação pai. Para isso, foi criada a noção de Gerenciador de Transação Especifico para Aplicação ("*Application-Specific Transaction Manager*"). Um Gerenciador de Transação Especifico para Aplicação dá suporte a uma variedade de tipos de transações onde cada uma delas é especificamente desenvolvida para um tipo de aplicação. É permitido ao usuário arbitrariamente estruturar diferentes tipos de subtransações em uma transação aninhada para formar uma árvore de transações heterogêneas.

Para permitir diferentes tipos de transações coexistirem na mesma árvore de transação, é necessário que sejam definidas algumas regras.

A primeira regra impõe que uma folha numa árvore de transação pode somente requisitar objetos que foram obtidos pelo pai. Se o filho necessitar de um objeto que não foi acessado pelo pai, este pode executar uma ação (*"stepwise transfer"*), solicitando ao seu pai que obtenha o objeto. Caso o pai não possa acessar o objeto, esta ação é propagada aos seus sucessores até que o objeto seja obtido.

A segunda regra, afirma que a transação pai é somente responsável pela coordenação da corretude e da execução do trabalho no seu nível. Em outras palavras, as características transacionais somente são aplicadas para os filhos da mesma transação. Os filhos, por sua vez, podem estabelecer um ambiente diferente para seus próprios filhos. Esta regra, chamada de *"two stage control-sphere"*, é essencial para o suporte de árvores de transações heterogêneas.

Para implementar este modelo, o sistema tool kit suporta um conjunto básico de componentes, que podem ser agrupados em diferentes modos para diferentes tipos de transações. A uma transação pode-se atribuir qualquer tipo de transação, caso seja apropriado à aplicação.

4.12 S transaction (Transações S)

O modelo de transações S [EVT88] foi introduzido para suportar cooperação no sistema bancário internacional. Neste ambiente, a autonomia local é o maior requisito. Este modelo segue o formato do modelo de transações aninhadas com algumas diferenças.

A primeira diferença, referente aos requisitos de autonomia e confiabilidade, é que o componente do sistema criador da S-transaction não pode a priori ditar que outros sistemas irão participar na execução da S-transaction. Um componente do sistema é autorizado a executar uma subtransação de uma S-transaction em qualquer local que deseje. No caso de falha, um componente do sistema pode emitir uma requisição equivalente para um outro sistema alternativo. Consequentemente, a exata execução da árvore de uma S-transaction é indeterminada e pode variar de uma execução para outra.

A segunda diferença, é que uma S-transaction pode usar transações compensatórias para a reabilitação. Cada subtransação irá validar quando estiver completa. Portanto, o isolamento do S-transaction é somente suportado ao nível da subtransação. Nenhum controle de concorrência ou de validação no nível da S-transaction é requerido. O sucesso e a falha das subtransações dependem do sucesso ou falha de suas filhas, e é descrito como uma expressão algébrica chamada restrição semântica (*"semantic constraint"*).

4.13 Multilevel Transaction Model (Modelo de transação multinível)

O modelo de transação multinível [WS91, Wei91] é uma generalização do modelo de transações aninhadas abertas, onde a árvore das subtransações é balanceada. Os nós de mesma profundidade da árvore correspondem às operações no mesmo nível de abstração no SGBD. Os ramos em uma árvore de transação representam a implementação de uma operação, que é invocada no nível L(i+1) por uma seqüência de operações executadas no nível abaixo L(i). Sendo que os níveis de uma transação de n-níveis são denominados por L(0), ..., L(n) na ordem de baixo para cima. Uma ação L(i) ($0 < i \leq n$) pode ser vista como uma subtransação L(i) que fica abaixo do nível de implementação L(i-1).

A idéia chave das transações multiníveis é explodir o nível de especificação semântica para controle de concorrência. Devido a este nível, duas operações L(i) podem não conflitar, ainda que seus níveis de implementação L(i-1) tenham conflitos. Neste caso, o conflito no nível L(i-1) torna-se um pseudoconflito no nível L(i). Assim uma execução que não é seriável no nível L(i-1) pode ser seriável no nível L(i). Os benefícios obtidos com a informação no nível de especificação de conflito permitem às transações multiníveis um grau de concorrência maior que nas transações tradicionais.

4.14 Polytransactions

Polytransactions [RS91] foi desenvolvido com a finalidade de criar um modelo de transações que pudesse facilitar o suporte de dados interdependentes em ambiente multidatabase[6]. Dados interdependentes são dois ou mais itens de dados armazenados em diferentes bases de dados que são relacionados entre si, através de alguma restrição de integridade. A restrição de integridade especifica a dependência dos dados e requisitos de consistência entre eles.

Neste modelo não é possível especificar que subtransações e dependências de execução vão ocorrer durante a execução da transação. Entretanto, existe um IDS (Interdatabase Dependency Schema) que contém um conjunto de descritores de dependência dos dados (D3) que especificam a interdependência dos dados. Os D3´s contém informações indicando as condições quando a dependência dos dados é considerada violada, e a ação que deve ser tomada para manter a dependência dos dados.

Uma Polytransaction (T+) é um fechamento transitivo de uma transação T com relação ao IDS. Isto é, quando uma transação T é executada os descritores de dependência dos dados são checados para vê se alguma ação deve ser tomada para manter a consistência dos dados. A ação tomada por sua vez pode ativar outras ações e assim por diante. Com efeito, a transação T é considerada a raiz transacional da polytransaction T+ e as ações geradas por T são as subtransações de T.

A idéia chave deste modelo é livrar dos programadores a preocupação com a manutenção da dependência global dos dados, pois o próprio sistema, com a utilização do

[6] Multidatabase: Coleção interconectada de banco de dados autônomos [OV91].

IDS, mantém a consistência destes dados. Outra vantagem, é que se houver qualquer mudança na dependência dos dados, não é necessário reescrever os programas da aplicação, bastando atualizar o IDS.

4.15 Chained Transaction (Transações encadeadas)

O objetivo do modelo de transações encadeadas [GR93] é ter um *"rollback"* mais flexível com menor perda de trabalho quando o sistema falhar.

Uma transação encadeada é um conjunto de transações que executam uma após a outra, sendo que a medida que as transações vão sendo executadas, estas são validadas, não podendo ser mais desfeitas. Entretanto, os objetos bloqueados pelas transações não são liberados quando a transação finaliza. Ao contrário, eles vão sendo passados de uma transação para outra, até que a transação encadeada finalize. Isto faz com que os resultados das transações sejam somente visíveis ao final da execução de todas a transações.

Em alguns casos é possível liberar objetos bloqueados que não vão ser compartilhados pelas transações subseqüentes, permitindo assim que os resultados da transação sejam visíveis antes da transação encadeada finalizar.

No caso de falha, a transação corrente será desfeita e uma nova transação será iniciada para tentar compensar a quebra da seqüência da execução.

4.16 Distributed Transaction (Transações Distribuídas)

Uma transação distribuída é tipicamente uma transação plana que executa em um ambiente distribuído e, por essa razão, tem que ser dividida nos vários nós da rede, dependendo da localização dos dados.

Formalmente uma transação distribuída T é um conjunto de subtransações Ti, onde cada Ti representa uma porção da transação T que é executada no site (computador) i da rede. T só é validada se todas Ti validarem. Se alguma Ti abortar todas as outras (sub)transações devem ser abortadas.

Este modelo suporta todas a propriedades ACID. Porém, estas propriedades somente são válidas, se cada site garantir que as suas subtransações são executadas atomicamente e

que as decisões de abortar e validar sejam coordenadas entre os sites.

Existem vários protocolos utilizados para coordenar as subtransações [OV91]. O mais comum atualmente é protocolo da validação em duas fases centralizado [OV91]. Neste caso, o site que inicia a transação, denominado site coordenador, pede a todos os participantes para executarem as suas subtransações correspondentes. Quando cada participante termina, ele entra no estado de prepared-to-commit, no qual espera a informação de desfazer ou de validar a subtransação. Quando o coordenador receber a mensagem prepared-to-commit de todos os participantes, ele envia a mensagem de commit para todos. Porém, se algum participante não quiser, ou não puder completar as suas subtransações, o coordenador escreve um registro de abort e envia a mensagem de abort para todos os sites.

Exemplos de sistemas que implementam este modelo são o ENCINA, CAMELOT e ARGUS.

4.17 Sumário dos modelos de transações

A tabela 4.17.c resume os modelo de transações apresentados neste capítulo. Os modelos são apresentados segundo três propriedades, conforme definido em [OV91]:

1. Estrutura da transação:

A estrutura de uma transação define seus componentes e os relacionamentos entre eles. Estes relacionamentos e a forma como os componentes se interligam são definidos através das dependências estruturais [Chr91].

Os modelos de transações podem ser agrupados basicamente em duas estruturas:

- *Estrutura plana* – nesta estrutura, todas as operações estão no mesmo nível, ou seja, pertencem ao mesmo escopo. Não existem subníveis (subtransações). Esta estrutura é adotada no modelo de transações tradicional. A figura 4.17.a ilustra este tipo de estrutura.

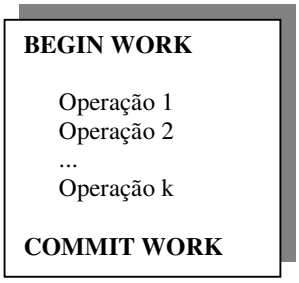

FIGURA 4.17.A – REPRESENTAÇÃO DA ESTRUTURA PLANA

- *Estrutura de subtransações* - neste caso, uma transação é constituída de um conjunto de outras transações, chamadas de subtransações. Esta categoria pode ainda ser subdividida segundo o relacionamento entre as subtransações e a forma como se interligam. Por exemplo, transações que são estruturadas na forma de árvore, são denominadas subtransações hierárquicas. A figura 4.17.b ilustra um exemplo da estrutura de subtransações. No caso do modelo Polytransaction, se a transação executar uma operação que quebre as restrições de integridade especificadas no esquema de dependências, será ativada uma subtransação (*"trigger"*), para restaurar a consistência da base de dados.

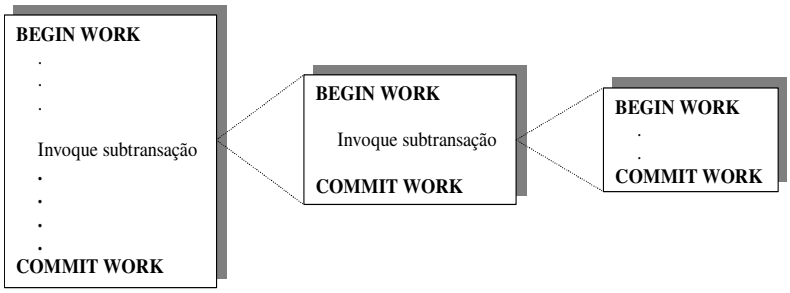

FIGURA 4.17.B – REPRESENTAÇÃO DE UMA TRANSAÇÃO COM A ESTRUTURA DE SUBTRANSAÇÕES.

2. **Tipos de subtransações:** Esta coluna especifica quais os tipos de subtransações são permitidas no modelo de transações. Subtransações Contigentes são subtransações que executam no lugar de outra subtransação, no caso de falha desta, que irá atuar como substituta da que falhou. Subtransações Não-Vitais são subtransações que, se falharem, não irão causar o *"rollback"* da transação pai.
3. **Corretude da transação:** esta coluna descreve o nível de corretude da transação especificado no modelo, ou seja, se propriedades ACID são válidas. No modelo DOM,

as propriedades ACID serão válidas se todos os componentes do sistema garantirem estas propriedades.

MODELO	ESTRUTURA DA TRANSAÇÃO	TIPO DAS SUBTRANSAÇÕES	CORRETUDE DA TRANSAÇÃO
TRADICIONAL	-	-	ACID
ANINHADA	Subtransação hierárquica	Contigente e Não-Vital	ACID
SAGAS	Subtransações	Compensatória	ACID[c]
COOPERATIVA HIERÁRQUICA	Subtransação hierárquica	Compensatória	Definida pelo usuário
COOPERATIVA SEE	Subtransação hierárquica	Trigger	Definida pelo usuário
DOM	Subtransação hierárquica	Contigente, Não-Vital e Compensatória	Dependente dos componentes do sistema
ANINHADAS ABERTA	Subtransação hierárquica	Contigente, Não-Vital e Compensatória	ACID[c]
CONTRATO	Subtransações	Contigente, Não-Vital e Compensatória	ACID[c]
SPLIT	Restruturação Dinâmica (Split e Join)	-	Preserva a semântica original
FLEX	Subtransação hierárquica	Contigente, Não-Vital e Compensatória	ACID[c]
KIT DE FERRAMENTA (TOOL KIT)	Subtransação hierárquica	Contigente e Não-Vital.	Definida pelo usuário
S	Subtransação hierárquica	Contigentel, Não-Vital e Compensatória	CD Para ACID[ac]
MULTI-NÍVEL	Subtransação hierárquica balanceada	Contigente, Não-Vital e Compensatória	ACID[c]
POLYTRANSACTION	Subtransação hierárquica dinâmica	-	Definida pelo usuário
SEQUENCIADA	Restruturação Dinâmica (Join)	-	ACID[c]
DISTRIBUÍDA	Subtransação hierárquica	Contigente e Não-Vital	ACID

TABELA 4.17.C – CARACTERÍSTICAS DOS MODELOS DE TRANSAÇÕES

Capítulo 5

Um Modelo de Transações Web Banco de Dados

5.1 Introdução

Como apresentado nos capítulos anteriores, a integração Web com banco de dados não se dá de forma natural, é necessário que sejam solucionados vários problemas.

Por vários anos, as pesquisas nesta área restringiram-se basicamente ao problema de como fazer o mapeamento entre as operações *stateless* da Web com as operações *statefull* dos SBGDs. Entretanto, a medida que foram surgindo soluções mais eficientes para este mapeamento, observou-se que a simples resolução deste problema não garantiria a ampla integração deste ambiente, já que novas características, que em princípio não estão presentes, passaram a ter que ser suportadas tanto pela Web, quanto pelos SGBDs.

Uma dessas propriedades está relacionada com o suporte das características transacionais neste ambiente. Como abordamos no capítulo 3, muito destas novas características não são suportadas pelo modelo de transações implementado nos SGBD's tradicionais (modelo de transações planas), como por exemplo, a necessidade de fazer referência a resultados intermediários. O que nos leva a identificar, a necessidade da especificação de um novo modelo de transações para este ambiente.

Portanto, este capítulo pretende especificar um modelo de transações que satisfaça essas novas propriedades. Na primeira parte deste capítulo será apresentada uma visão informal simplificada do modelo, ressaltando a sua estrutura, tipos de subtransações

suportadas e o controle de concorrência. Na segunda parte o modelo será especificado formalmente utilizando o meta modelo ACTA. Serão discutidas detalhadamente nesta fase questões relacionadas com as regras de validação ou aborto, dependências entre as subtransações, eventos significativos e as relações de conflito.

5.2 Estrutura da Transação

Conforme descrito na seção 4.17, a estrutura de uma transação define seus componentes e os relacionamentos entre eles. Estes relacionamentos e a forma como os componentes se interligam são definidos através das dependências estruturais.

Nesta seção será definida a estrutura do modelo proposto, enfatizando as limitações da estrutura do modelo de transações planas quanto às características das transações Web.

5.2.1 Limitações da estrutura plana para transações Web banco de dados

O problema da utilização da estrutura plana para transações Web banco de dados é decorrente da diferença da natureza do escopo das operações, como mencionado no capítulo 3.

No modelo tradicional, que utiliza a estrutura plana, todas as operações de uma transação estão no mesmo escopo, delimitadas pelos operadores *begin_work* e *commit _work*. Já numa transação Web banco de dados, onde a unidade de consistência é a página, o escopo das operações deve variar segundo a página onde se encontram [Lim97], ou seja, todas operações envolvidas na criação de uma página devem ser tratadas como uma transação única, ou todas as operações são executadas ou nenhuma delas.

Imagine a situação onde uma transação compreende várias páginas Web (várias páginas como unidade atômica), como na figura 5.2.1.a.

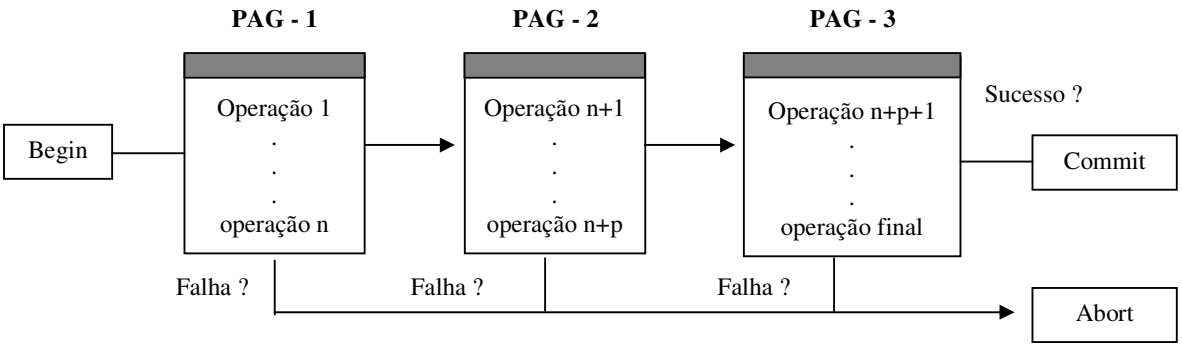

FIGURA 5.2.1.A – EXEMPLOS DE UMA TRANSAÇÃO COM VÁRIAS PÁGINAS DE INTERAÇÃO

Para implementar esta situação utilizando o modelo tradicional, indica-se antes da primeira operação da página 1, o início da transação (*begin_work*) e depois da última operação da página 3, o fim da transação (*commit*). Assim, todas as operações que ocorrem entre estes demarcadores serão consideradas como parte de uma única transação atômica global. A figura 5.2.1.b representa esta situação.

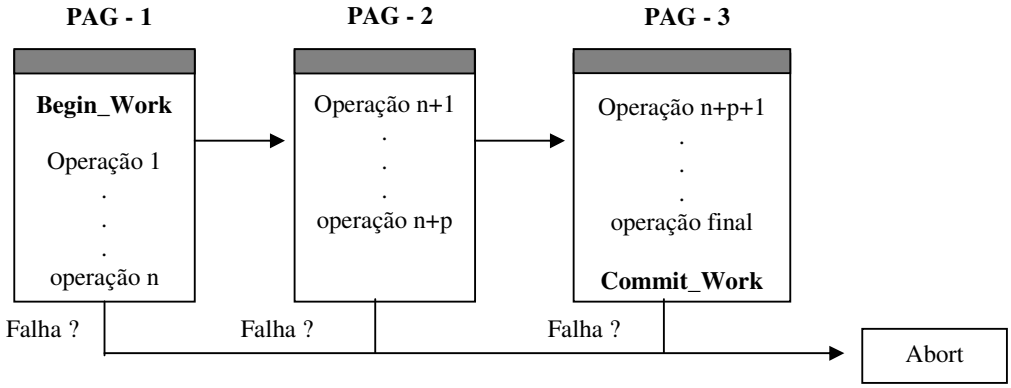

FIGURA 5.2.1.B – ADEQUAÇÃO DO MODELO DE TRANSAÇÕES PLANA, AO CASO DE UMA TRANSAÇÃO QUE ENVOLVA VÁRIAS PÁGINAS DE INTERAÇÃO

Esta solução parece satisfatória, mas pode resultar em alguns inconvenientes. Por exemplo, no caso da figura 5.2.1.b, se o usuário executar as operações da primeira e segunda páginas com sucesso e na terceira alguma operação falhar, então todas as operações da transação serão desfeitas. Esta situação pode parecer normal para usuários de SGBDs, mas para os usuários da Web esta situação pode parecer não natural. Para eles, seria esperado o retorno para o estado da segunda página, desfazendo somente as operações da terceira e não todas, contrariamente ao que aconteceria no modelo tradicional.

Além dos problemas de falha, temos os problemas navegacionais resultantes da utilização dos botões *back* e *forward* presentes nos browsers Web, que permitem o usuário avançar ou voltar uma ou mais página. Imagine por exemplo, a situação onde o usuário está na segunda página e utiliza a opção de *back* para voltar à página anterior. Neste caso, seria esperável que somente as operações da segunda página fossem desfeitas, entretanto isto não é possível, pois no modelo tradicional, ou todas as operações são desfeitas ou nenhuma delas. Assim, para que as operações da página 2 fossem desfeitas, todas as outras operações teriam que ser desfeitas.

5.2.2 Estrutura da transação do modelo proposto

Para resolvermos os problemas anteriores, substituímos a estrutura plana da transação por uma estrutura de subtransações encadeadas, onde cada subtransação corresponde a uma página de interação da transação, semelhante ao modelo de transações encadeadas.

Assim, a estrutura do modelo proposto nesta dissertação para o ambiente de integração é formada por um conjunto de subtransações executadas seqüencialmente, onde cada subtransação representa uma página de interação.

O exemplo da figura 5.2.1.a pode ser refeito, utilizando o modelo proposto, como representado na figura 5.2.2.

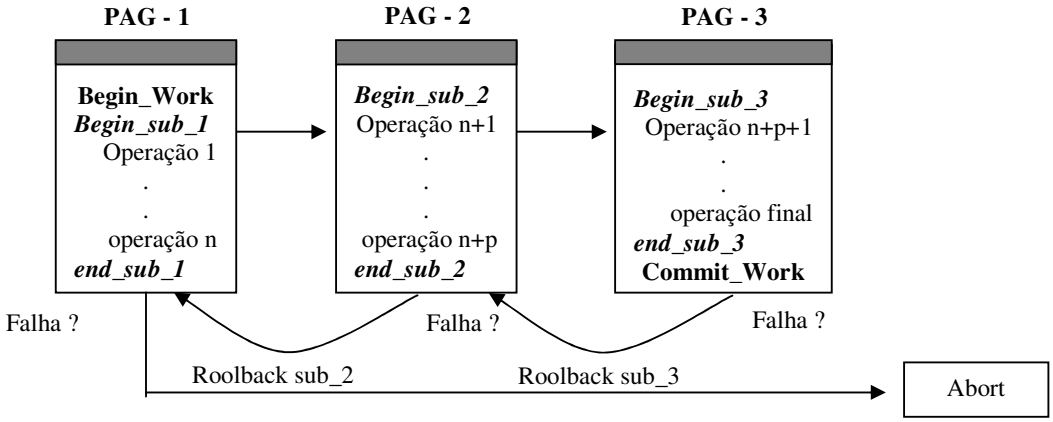

FIGURA 5.2.2 - ADEQUAÇÃO DO MODELO DE TRANSAÇÕES PROPOSTO, AO CASO DE UMA TRANSAÇÃO QUE ENVOLVA VÁRIAS PÁGINAS DE INTERAÇÃO

A transação plana original é quebrada em três subtransações, cada uma contendo as

operações contidas na página Web correspondente. As operações em cada página Web são agrupadas dentro de uma subtransação, que será executada seguindo a ordem de execução das páginas Web.

Neste caso, se ocorrer alguma falha em qualquer uma das páginas, não será preciso desfazer toda a transação, bastando apenas desfazer a subtransação correspondente aquela página, voltando ao estado da página anterior. No caso do *back*, basta desfazer a subtransação correspondente. Ou seja, se por acaso o usuário estiver na página 2 e resolver voltar para a página 1 (*back*), então basta executar o *roolback* da subtransação 2 e voltar à página 1.

5.3 Tipos de subtransações

Assim como os modelos de transações podem variar segundo a estrutura das suas subtransações, estes podem variar segundo a função desempenhada e os eventos que acionam a subtransação.

Dependendo do comportamento, as subtransações podem apresentar diferentes denominações. Subtransações compensatórias, não-vitais e contigentes são exemplos de tipos de subtransações [Elm90].

O modelo de transações adotado nesta dissertação apresenta dois tipos de subtransações: subtransação página e subtransação compensatória.

5.3.1 Subtransação página

Como apresentado anteriormente, as subtransações páginas compreendem ao conjunto das operações da transação que devem ser executadas em cada página Web, que formam a transação Web global.

Assim, se a transação Web for constituída de cinco páginas, esta possuíra cinco subtransações páginas, uma para cada página, conforme a figura 5.3.1. Deve-se ressaltar que a ordem de execução das subtransações obedece à ordem da execução seqüencial das páginas Web.

A finalidade da subtransação página é adequar a transação Web global ao escopo da

página Web, criando para cada página, que faz parte da transação global, um ponto de reabilitação.

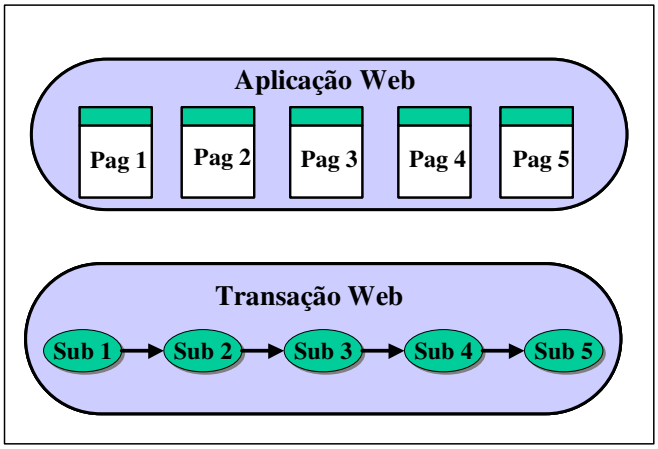

FIGURA 5.3.1 – RELAÇÃO ENTRE PÁGINAS WEB E AS SUBTRANSAÇÕES PÁGINAS

Semelhante ao modelo de transações encadeadas, as subtransações páginas são executadas uma após a outra, diferenciando-se apenas com relação o ponto de validação, que pode ocorrer ao final da subtransação ou somente ao final da transação pai, e que uma subtransação finalizada pode ser desfeita. Nas seções seguintes, estas características são mais detalhadas.

5.3.2 Transações compensatórias

Uma transação C é dita compensatória [Gra81] de uma transação T', se corresponde semanticamente ao efeito do undo de T', depois de T' ter sido validada, não precisando necessariamente restaurar o banco de dados para o estado a que se encontrava quando T' começou a executar.

Transações compensatórias são úteis para atividades de longa duração, pois permite liberar os resultados das subtransações antes que a transação global finalize.

Como apresentado na seção 3.2, as transações Web banco de dados são caracterizadas por apresentar um longo tempo de execução. Tal característica, entretanto, pode ser comprometida quando utilizamos o modelo de transações planas, pois este modelo requer que todos os bloqueios aos dados sejam mantidos até a transação finalizar, o que pode ocasionar sérios problemas de desempenho, já que limita a concorrência dos

dados.

Imagine por exemplo, o processo de execução de uma transação de aquisição de produtos em um Shopping Virtual na Web, conforme a figura 5.3.2.

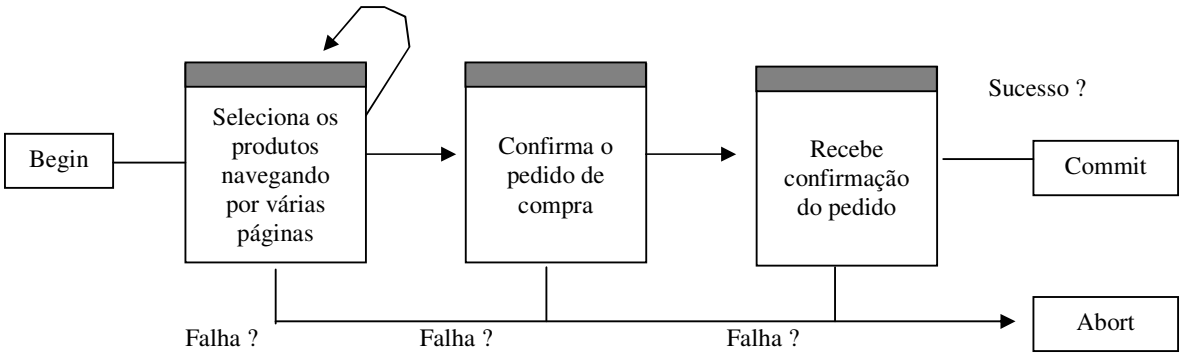

FIGURA 5.3.2 - TRANSAÇÃO DE COMPRAR DE PRODUTOS EM UM SHOPPING VIRTUAL

Neste tipo de transação, o usuário vai navegando pelas páginas do shopping selecionando os produtos desejados, que por sua vez são bloqueados junto ao SGBD para evitar inconsistência. Ao final da compra, a transação é finalizada e os bloqueios são liberados. O problema na execução desta transação é que o tempo de execução é bastante considerável, e como os bloqueios somente são liberados ao final da transação, os itens bloqueados ficam indisponíveis aos outros usuários por um longo período, mesmo havendo em estoque, até a transação finalizar, comprometendo assim, a concorrência.

Para solucionar este problema (transações de longa duração) foi adicionada ao modelo proposto a figura da transação compensatória. Neste caso, a transação global será quebrada em um conjunto de subtransações páginas (uma subtransação para cada página), como no caso da seção 5.2.2, sendo que ao final da execução de cada subtransação, o projetista da transação pode optar em liberar ou não os bloqueios obtidos pela subtransação, permitindo assim, que esta possa liberar certos resultados para outras transações, antes que a sua transação pai finalize.

Se o projetista decidir em liberar os bloqueios, este deverá associar a subtransação uma transação compensatória, que será ativada caso a subtransação tenha que ser desfeita. É responsabilidade do projetista definir (criar) a transação compensatória, que deverá desfazer o efeito da subtransação página associada a ela. A subtransação será desfeita no caso da transação abortar ou quando o usuário solicitar, através da utilização do botão *back*. É importante mencionar que nem toda transação é compensável, e qualquer erro na

implementação da compensatória poderá levar o SGBD para um estado inconsistente.

Já se o projetista decidir em não liberar os bloqueios, não há a necessidade da definição da compensatória, sendo os bloqueios somente liberados ao final da transação Web.

Além do fato de solucionar o problema de transações de longa duração, a utilização das transações compensatórias permite também solucionar o problema com a utilização do botão *back* dos navegadores. Por exemplo, no caso do problema de atualização descrito na seção 3.4, pode-se associar a transação de atualização uma transação compensatória que será ativada quando ocorrer um *back* para esta página, desfazendo os efeitos da atualização.

Portanto, as transações compensatórias terão duas funções básicas:

1. Permitir a execução de transações de longa duração.
2. Permitir a representação semântica do *back* do browser no estado do banco de dados.

5.4 Controle de concorrência

No modelo aqui proposto, o controle de concorrência esta intimamente ligado a presença ou não das subtransações compensatórias.

Quando não existir subtransações compensatórias associadas as subtransações páginas, os bloqueios obtidos durante a execução da subtransação somente serão liberados ao final da execução da transação pai, não permitindo que outras transações concorrentes tenham acesso a estes itens de dados bloqueados. Entretanto, os bloqueios obtidos pelas subtransações podem ser compartilhados pelas outras subtransações subsequentes a ela, pertencente a mesma transação pai.

Quando houver a figura da subtransação compensatória, os bloqueios obtidos pela própria subtransação página serão liberados ao final de sua execução, permitindo que as outras transações tenham acesso a resultados intermediários antes da finalização da transação pai.

Já o controle de concorrência da subtransação compensatória é semelhante a de uma transação tradicional onde os bloqueios obtidos por ela serão liberados ao final de sua

execução.

5.5 Especificação formal do modelo

Para eliminar ambigüidades, permitir análises do modelo de transações e facilitar comparações com outros modelos, é necessário expressar o modelo de modo formal. Nesta seção, serão definidas as semânticas das operações do modelo de transações Web banco de dados, utilizando para isto o meta modelo de transação ACTA. Antes de começar a especificação do modelo, iremos primeiro apresentar um breve resumo sobre o modelo ACTA. Em [Chr91] é apresentado detalhadamente este formalismo.

5.5.1 Resumo do formalismo ACTA

O ACTA é uma notação formal para a especificação e análise dos modelos de transações. Os conceitos básicos deste formalismo são:

- Um modelo de transações é caracterizado pela interação de objetos e transações. Cada objeto tem um estado e um "status". O estado representa um valor do objeto e o "status" contém informações do controle de concorrência.
- Uma transação T possui um conjunto de eventos significativos (denominado SE_T), que correspondem aos eventos que são relevantes para a transação T. Estes por sua vez, podem ainda ser classificados em evento de inicialização (IE_T) ou eventos terminais (TE_T).
- Cada transação T tem um conjunto visão (denominado $ViewSet_T$) que é o conjunto de objetos potencialmente visível para esta, e um conjunto de acessados (denominado $AcessSet_T$), que é o conjunto de todos os objetos acessados pela transação. Nenhuma distinção é feita no conjunto "ViewSet" entre os objetos acessados para leitura e escrita.
- Cada transação T tem um conjunto de conflitos (denominado $ConflictSet_T$), que contém aquelas operações em execução (operações ainda não validadas) que são conflitantes com T.

- Transações podem delegar objetos que pertencem a seu conjunto de acessados através do operador “delegate”. Uma transação pode delegar o estado ou "status" dos objetos do seu conjunto de acessados. Na delegação do estado, resultados parciais tornam-se visíveis para outras transações. Na delegação do “status”, as modificações de quem delegou são desfeitas antes dos objetos serem adicionados ao conjunto de acessados para quem foi delegado.
- As transações interagem com as outras através das dependências:
  * “Begin-on-Commit” (t_j BCD t_i) - a transação t_j não pode começar a execução até que t_i valide (commit);
  * “Weak-begin-on-Commit” (t_j WCD t_i) – se t_i validar, t_j pode começar a executar depois de t_i validar.
  * “Weak-Abort” (t_j WD t_i) – se t_i abortar e t_j ainda não tiver validado, então t_j aborta.
  * “Force-Commit-on-Abort” (t_j CMD t_i) – se t_i abortar, t_j valida.
  * “Strong-Commit” (t_j SCD t_i) – se a transação t_i validar então t_j valida.
  * Begin-on-Abort (t_j BAD t_i)- a transação t_j não pode começar a executar até que t_i aborte.
  * Begin-on-Back (t_j BBD t_i) a transação t_j não pode começar a executar até que t_i realize um “back”. $(Begin_{tj} \in H) \Rightarrow (back_{ti} \rightarrow Begin_{tj})$.

  Deve-se ressaltar que esta última dependência não é parte do modelo original ACTA, e que foi definida para capturar a semântica do evento de navegação back, existente no modelo hipertexto da Web. Na seção seguinte o evento back será definido.

5.5.2 Definição dos axiomas do modelo de transações Web Banco de Dados

Nesta seção vamos expressar as propriedades do modelo de transações Web Banco de Dados definidas nas seções anteriores, através de um conjunto de axiomas utilizando o

ACTA.

W – Representa um transação Web com n páginas, havendo g transações compensatórias

NP_i – Representa uma página não compensável

P_i – Representa uma página compensável

$CP_{i,j}$ – Representa a transação compensável de Pi, correspondente a j-ésima transação compensatória criada

t – Representa NP_i ou P_i ou $CP_{i,j}$

t' – Representa tanto NP_i ou P_i

V – Representa tanto P_i ou $CP_{i,j}$

T_i – Representa tanto NP_i ou P_i

1. $SE_w = \{Begin, Commit, Abort, Back\}$
2. $IE_w = \{Begin\}$
3. $TE_w = \{Commit, Abort\}$
4. $SE_t = \{Begin, Commit, Abort\}$
5. $SE_t = \{Begin\}$
6. $SE_t = \{Commit, Abort\}$
7. t satisfaz aos axiomas fundamentais de I a IV do ACTA
8. $View_w = \varnothing$
9. $View_t = H_{ct}$
10. $ConflicSet_w = \varnothing$
11. $ConflicSet_{CPi,j} = \{p_{t''}[ob] \mid t'' \neq CP_{i,j}, Inprogress\ (p_{t''}[ob])$
12. $ConflicSet_{t'} = \{p_{t''}[ob] \mid t' \neq t'', t'' \notin$ Anteriores de t', $Inprogress\ (p_{t''}[ob])$
13. $\forall ob\ \exists p\ (\ p_t[ob] \in H\ ) \Rightarrow$ (ob é atômico)
14. $(Commit_t \in H\ ) \Rightarrow \neg\ (t\ C^*_n\ t)$
15. $\exists ob \exists p \exists t(Commit_t[p_{t'}[ob]] \in H) \Rightarrow Commit_t$
16. $(Commit_w \in H) \Rightarrow \forall ob \forall p \forall h((ResponsibleTr(p_h[ob]=W) \Rightarrow (Commit_w[p_h[ob]] \in H))$
17. $(Commit_{Vi} \in H\ ) \ \forall ob \forall p((p_v[ob] \in H) \Rightarrow (Commit_V[p_v[ob]] \in H))$
18. $\exists ob \exists p \exists t(Abort_t[p_{t'}[ob]] \in H) \Rightarrow Abort_t$
19. $(Abort_t \in H) \Rightarrow \forall ob \forall p \forall h((ResponsibleTr(p_h[ob]=t) \Rightarrow (\ Abort_t\ [p_h[ob]] \in H))$
20. $post(Begin_w) \Rightarrow (((T_i\ BCD\ T_{i-1}) \in DepSet_{ct}) \wedge (CP_{*,j}\ WCD\ CP_{*,j+1}) \in DepSet_{ct}) \wedge ((CP_{i,g}\ BAD\ W) \vee (CP_{i,g}\ BBD\ Pi)) \in DepSet_{ct}))$, onde $1 \leq i < n$, e $0 \leq j \leq g$, * corresponde qualquer valor de i
21. $post(Begin_{Ti}) \Rightarrow (((T_i\ WD\ W\ ) \in DepSet_{ct}) \wedge\ (CT_i\ BCD\ T_i) \in DepSet_{ct}))$, onde $1 \leq i < n$
22. $post(Commit_{Pi}) \Rightarrow (((CP_{i,j}\ CMD\ W) \in DepSet_{ct}) \wedge (\ (CP_{i,\ g}\ BAD\ W) \vee (CP_{i,j}\ BBD\ Pi)\ ) \in DepSet_{ct}))$, onde $1 \leq i < n$
23. $post(Begin_{Tn}) \Rightarrow ((W\ SCD\ T_n\ ) \in DepSet_{ct}))$
24. $(Commit_{NPi} \in H) \Rightarrow (delegate_{NPi}[W, AcessSetTr[NP_i] \in H])$, onde $1 \leq i < n$
25. $(back_W[T_j] \in H) \Rightarrow (((T_j \in NP) \vee (Commit_{Tj} \notin H)) \Rightarrow (Abort_{Tj} \in H) \vee ((T_j \in P) \Rightarrow (Commit_{CTj} \in H)))$

FIGURA 5.5.2 - AXIOMAS DO MODELO DE TRANSAÇÕES WEB BANCO DE DADOS

Pela definição do modelo, figura 5.5.2, o modelo de transação é composto por três tipos de transações, denominadas, transação Web (W), transação página (Ti) e a transação compensatória ($CP_{i,j}$). A transação página, por sua vez, pode ser de dois tipos:

compensável (P_i) e não compensável (NP_i).

5.5.2.1 Eventos significativos

Os axiomas de 1 a 6 definem os eventos significativos de cada tipo de transação, bem como os eventos de inicialização e finalização. Deve-se ressaltar a inserção do evento back nos eventos significativos da transação Web (W), para capturar o evento de navegação "backward" existente no ambiente Web. O evento back ($Back_W[T_i]$) indica para a transação Web (W) que esta deve desfazer (abortar) as operações da transação página T_i.

Pode-se observar que não foi definido nenhum evento significativo associado ao evento "forward" existente no navegador Web. Isto porque a atualização do estado do navegador e do estado do banco de dados somente é realizada quando é executada uma nova operação no banco. Portanto, nunca chegará ao banco a requisição para avançar um estado, mas somente retornar ao estado anterior, já que sempre a última página da aplicação requisitada pelo navegador corresponderá ao estado corrente do banco de dados.

Se a atualização dos estados fosse executada a cada mudança de evento do navegador (*"backward"* e "*forward*"), o evento "*forward*" deveria estar presente no modelo. Entretanto a inserção deste evento é complexa, pois para que a transação seja refeita, o banco de dados precisaria retornar ao estado a que se encontrava quando foi executada. Entretanto em alguns casos isto não é possível, uma vez que transações podem ter sido validadas durante o intervalo de tempo entre o "*backward"* e o "*forward"*, não podendo ser desfeitas, devido à propriedade da durabilidade.

5.5.2.2 Axiomas fundamentais

O axioma 7 define que as transações do modelo obedecem as propriedades fundamentais das transações definidas em [Chr91].

I. $\forall \alpha,\beta\ IE_t\ (\alpha \in H^t) \Rightarrow (\beta \notin H^t)$

II. $\forall \delta \in\ TE_t\ \exists \alpha \in\ IE_t\ (\delta \in H^t) \Rightarrow (\alpha \rightarrow \delta)$

III. $\forall \gamma,\delta \in\ TE_t\ (\gamma \in\ H^t) \Rightarrow (\delta \notin\ H^t)$

IV. $\forall ob \forall p,(p_t[ob] \in H) \Rightarrow (\exists \alpha \in IE_t(\delta \in H^t) \Rightarrow (\alpha \rightarrow p_t[ob]))\ \wedge (\exists \gamma \in\ TE_t\ (p_t[ob] \rightarrow \gamma)))$

O primeiro axioma define que uma transação não pode ser inicializada por dois eventos diferentes. O segundo especifica que uma transação só pode finalizar se tiver sido inicializada. O terceiro axioma define que uma transação não pode ser finalizada por dois eventos diferentes. E o último estabelece que somente transações em execução podem invocar operações em objetos.

5.5.2.3 Conjunto visão

Os axiomas 8 e 9 definem as regras de visões das transações, referentes ao conjunto de visão de cada transação. Pelo axioma 8, a transação Web (W) não pode diretamente operar sobre objetos do banco de dados. Já o axioma 9 define que as transações página (T_i) e compensatórias (CP_{ij}) possuem como conjunto de visão todos os objetos do banco de dados, podendo executar as operações sobre qualquer objeto do banco de dados..

5.5.2.4 Conjunto de conflito

Os axiomas de 10 a 12 definem o conjunto de conflitos das transações. O axioma 10 define que a transação Web não possui conflitos com nenhuma operação de uma outra transação do banco de dados. O axioma 11 define que o conjunto de conflitos da transação compensatoria é formado por todos as operações em progresso, que não pertencem a própria transação. O axioma 12 define que o conjunto de conflito da transação página é formado por todas operações em progresso que não pertencem a transação página e as suas antecessoras. Ou seja, a transação página pode acessar objetos acessados por suas antecessoras mesmo que as operações sejam conflitantes.

O axioma 13 define que todos os objetos acessados pelas transações são atômicos, ou seja, comportam-se corretamente e seriavelmente, conforme o axioma 3.26 do ACTA. Para um objeto (ob) comportar-se corretamente, deve-se garantir que quando uma operação abortar, qualquer outra operação dependente desta, deve também ser abortada. Já um objeto comporta-se seriavelmente se para toda operação validada sobre o objeto ob, não deve haver operações conflitantes com esta.

O axioma 14 garante que as transações página e compensatórias são seriáveis mesmo na presença de delegação, ou seja, só podem validar, se todas as operações da transação e

as operações delegadas a ela, não são conflitantes com outras operações de outras transações.

5.5.2.5 Regras de validação e aborto

Os axiomas de 15 a 17 estão relacionados com as regras de validação. O axioma 15 define que se qualquer operação de uma transação for validada, então aquela transação deve ser validada. O axioma 16 define que se a transação Web validar, então todos as operações da transação Web e todas as operações delegadas a ela, devem ser validadas. O axioma 17 define que se uma transação página compensável ou uma transação compensatória validar, então todas as suas operações devem ser validadas. Estes axiomas definem duas propriedades básicas do modelo. A primeira delas é que as operações das transações páginas não compensáveis, somente serão validadas se a transação Web validar, e a segunda é que os resultados das transações páginas compensáveis podem liberar seus resultados antes da transação Web finalizar.

Os axiomas 18 e 19 estão relacionados com o caso de falha das transações. O primeiro axioma especifica que se uma operação da transação falhar então a transação deve ser abortada. O outro axioma especifica que se a transação falhar então todas as suas operações devem ser abortadas, juntamente com aquelas que lhe foram delegadas.

5.5.2.6 Conjunto de dependências

Os axiomas de 20 a 23 definem o conjunto de dependências que devem ser válidas durante a execução das transações.

O axioma 20 define três dependências. A primeira dependência especifica a propriedade da ordem de execução seqüencial das transações páginas, garantindo que uma transação só pode iniciar depois que a sua sucessora validar. A segunda dependência está relacionada com a ordem de execução das transações compensatórias, esta define que se a transação compensatória sucessora validar então a posterior deve iniciar depois dela, seguindo a ordem inversa da ordem de execução das transações páginas correspondentes. A terceira dependência está relacionada com o evento de ativação das transações compensatórias. Estas podem ser ativadas em dois casos, o primeiro se a transação Web

falhar e a segunda se a transação Web realizar um back.

O axioma 21 define duas dependências. A primeira dependência especifica que se a transação Web falhar e se a transação página i ainda não tiver validado, então esta transação deve ser abortada. A segunda dependência especifica que a transação compensatória $CP_{i,j}$ só pode começar a executar depois que P_i validar.

O axioma 22 define duas dependências. A primeira dependência específica que se a transação Web falhar a transação compensatoria $CP_{i,j}$ deve validar. A segunda dependência especifica que se a transação $CP_{i,j}$ só pode começar a executar depois que W abortar ou executar o back de P_i.

O axioma 23 especifica que a transação Web só pode validar se a transação Tn (a última transação página) validar.

O axioma 24 define que se uma transação página não compensável validar, esta delega para a transação Web a responsabilidade de validar ou abortar as operações da transação página não compensável.

O último axioma do modelo especifica que se uma transação página sofrer um back esta pode ativar uma de duas ações, dependendo se a transação é compensável ou não. Caso a transação seja não compensável ou ainda não tenha sido validada esta transação deve ser abortada. Já se a transação for compensável então a sua transação compensatória deve ser validada.

5.6 Critério de correção

O critério de correção transacional descreve o nível de corretude da transação definida pelo modelo.

Em geral, os modelos de transações adotam o critério de correção baseado nas propriedades ACID. Entretanto, como as aplicações estão se tornando cada vez mais complexas, estas requerem critérios diferentes dos adotados tradicionalmente.

Para satisfazer estes novos critérios, os modelos de transações passaram a relaxar as propriedades ACID. Como por exemplo, no modelo Sagas, a propriedade de isolamento é relaxada, permitindo a transação liberar resultados parciais as outras transações, antes que

esta finalize.

Em outras situações, o critério de correção baseado nas propriedades ACID, é substituído por critérios definidos pelo próprio usuário. Exemplos de modelos que adotaram este novo critério de correção são: Polytransactions, Transaction Tool Kits, entre outros.

Apesar deste novo critério permitir uma melhor adaptação às necessidades das aplicações, este torna o gerenciamento das transações mais complexo e requer por parte do projetista do banco de dados, um trabalho substancial para definição destes critérios.

5.6.1 Critério de correção do modelo proposto

O critério de correção do modelo proposto depende do tipo de subtransação que constitui a transação Web.

Se a transação não apresentar subtransações do tipo compensatórias, esta transação satisfaz todas as propriedades ACID. Isto é verdade pois uma transação Web banco de dados, neste caso, corresponde a uma transação plana com vários pontos de reabilitação. As subtransações são executadas sequencialmente e os resultados somente são liberados ao final da transação.

No caso, onde a transação Web banco de dados apresenta subtransações compensatórias, o critério de corretude satisfaz as propriedades ACID, se as subtransações são compensáveis. Neste caso o modelo proposto relaxa a propriedade de isolamento, uma vez que os resultados das subtransações tornam-se visíveis para as outras subtransações antes que a transação finalize.

5.7 Conclusão

Este capítulo apresentou os principais resultados desta dissertação. Como primeiro resultado, temos a comprovação da limitação do modelo de transações planas para o ambiente de integração Web banco de dados. Pelo apresentado nas seções 5.2.1 e 5.3.2 foi constatado que as limitações do modelo de transações planas estão intimamente

relacionados com a incapacidade de desfazer parte da transação e falta de suporte a transações de longa duração.

Para solucionar estes problemas, foi especificado um novo modelo de transações que quebrou a estrutura plana do modelo tradicional em um conjunto de subtransações páginas, que agrupam as operações executadas em cada página Web, possibilitando assim a capacidade de desfazer parte da transação.

Para suportar transações de longa duração foi inserido ao modelo o conceito de transações compensatórias, que permite ao projetista definir que alguns resultados da transação Web sejam liberados antes que a transação valide. Apesar das vantagens da utilização das transações compensatória (aumento da concorrência), deve-se ressaltar que o projetista deve ter muito cuidado na sua utilização, pois não existe nenhuma técnica para definição da transação compensatória, assim se não tiver cuidado pode gerar sérios problemas durante a execução da aplicação.

Para eliminar qualquer ambigüidade ou dúvida sobre o modelo, este foi especificado formalmente utilizado o meta modelo ACTA. A especificação formal é extremamente importante por possibilitar que trabalhos futuros possam analisar comparativamente este modelo com outros de forma mais criteriosa.

Deve-se ressaltar que poderíamos ter utilizado estruturas mais complexas na definição do modelo. Entretanto buscou-se utilizar estruturas mais simples, para que este possa ser simulado utilizando somente os recursos transacionais existentes nos SGBDs tradicionais. No capítulo seguinte iremos apresentar a implementação de um gateway de integração que simule o modelo proposto utilizando somente os recursos existentes no SGBD.

Capítulo 6

Um gateway transacional

6.1 Introdução

Nos capítulos anteriores, comprovamos que o modelo de transações planas é ineficiente para o ambiente de integração Web Banco de dados e propomos um novo modelo de transações que melhor se adapta às restrições impostas pelo ambiente de integração.

Definido o modelo, surge agora um impasse de onde implementá-lo, uma vez que, se implementarmos diretamente sobre um SGBD que suporte esse modelo ou em um dos elementos constituintes da Web (browser, Servidor Web), estaríamos burlando a restrição da não alteração das tecnologias, apresentado na seção 2.4.2 .

Portanto, para satisfazer essa restrição, decidimos implementar um novo Gateway de Banco de Dados, que além de implementar as funções de um gateway tradicional, também implemente o modelo proposto nesta dissertação, utilizando para isto, somente os recursos transacionais existentes nos SGBDs tradicionais.

Neste capítulo, apresentaremos o desenvolvimento do Gateway Transacional. Na seção 6.2 apresentaremos a arquitetura adotada, na seção 6.3 o diagrama de objetos, na seção 6.4 o funcionamento do gateway e na seção 6.5 as tecnologias envolvidas e implementação do modelo.

6.2 Arquitetura do Gateway

Conforme apresentado na seção 2.4.4, os gateways de banco de dados podem apresentar os mais variados tipos de arquiteturas, estas por sua vez podem também ser classificadas segundo vários critérios. Nesta dissertação, adotamos a taxánomia baseada na localização do gateway, se no lado cliente ou no lado servidor Web.

Depois de compararmos as vantagens e desvantagens de cada arquitetura, analisadas em [Lim97], resolvemos adotar, para o gateway proposto, uma variação da arquitetura de Gerenciador de Aplicação, uma vez que esta apresenta as melhores características.

A figura 6.2 ilustra a arquitetura do gateway proposta, dividida em cinco módulos: Módulo Despachante CGI, Módulo Gerente, Módulo Aplicação, Módulo de Dados e Módulo URL.

A especificação dos módulos Despachante CGI, Dados e URL foi definida tendo como base à arquitetura de desenvolvimento de aplicação CGI proposta no Delphi 4.0, na qual existe um módulo para tratar a interface CGI, um para o acesso ao SGBD e um para a criação das páginas Web.

O módulo Grente e o Aplicação foram definidos para possibilitarem a manutenção da conexão.

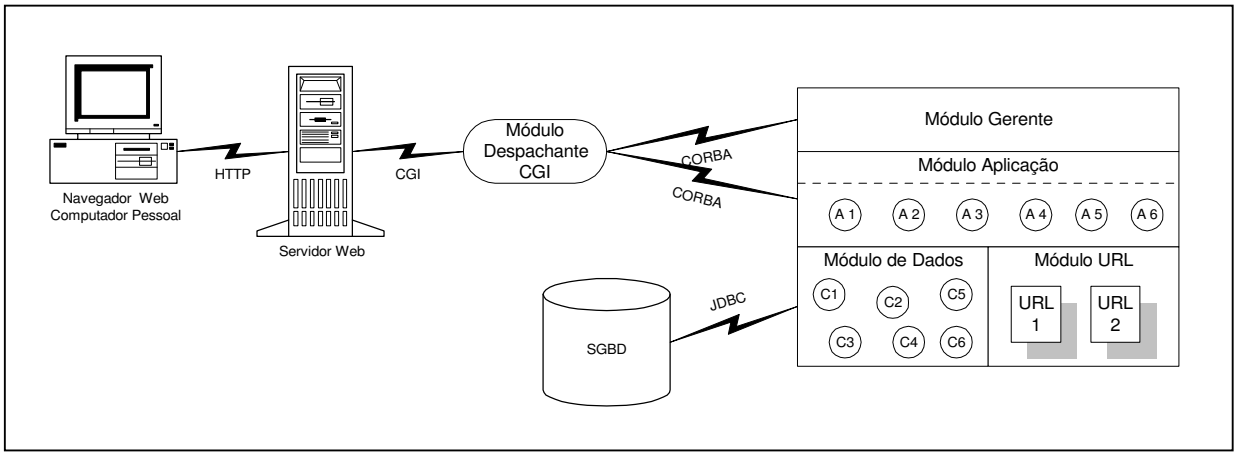

FIGURA 6.2 – ARQUITETURA DO GATEWAY PROPOSTO

6.2.1 Módulo Despachante CGI

O objetivo do Módulo Despachante CGI é fazer a intermediação entre o Servidor Web e Gerenciador de Aplicação. Para desempenhar esta atividade, este realiza três funções básicas:

1. A primeira função é conectar-se com o Módulo Gerente na primeira interação, para a criação de uma instância do Módulo Aplicação, que passará a receber as próximas interações;
2. A segunda função é formatar as informações passadas via formulário HTML, através da interface CGI do Servidor Web, e repassá-las para o Módulo Aplicação;
3. A última função deste módulo é obter as informações passadas pelo Módulo Aplicação e repassá-las para o Servidor Web, que serão redirecionadas para o navegador Web.

6.2.2 Módulo Gerente

O objetivo deste módulo é gerenciar o ciclo de vida dos objetos do Módulo Aplicação, ou seja, coordenar a criação, a execução e a destruição dos objetos instanciados no Módulo Aplicação. Três funções são realizadas por este módulo:

1. A primeira função compreende a criação e identificação das instâncias dos objetos do Módulo Aplicação, requerido pelo Módulo Despachante CGI. O processo de identificação corresponde à criação de um identificador único para cada objeto, que será utilizado pelo Módulo Despachante CGI para fazer uma conexão direta com a aplicação sem necessitar da intervenção do Módulo Gerente. Para criar este identificador, foi utilizado um contador global, que é incrementado de um a cada nova aplicação. Para impedir objetos diferentes com o mesmo identificador foi necessário implementar um mecanismo de controle de concorrência, este processo será melhor analisado na seção 6.5.2.1.
2. A segunda função refere-se ao controle do tempo de execução das instâncias dos objetos do Módulo Aplicação, para impedir que objetos fiquem instanciados indefinidamente, no caso do usuário abandonar a navegação sem finalizar a aplicação;

3. A última função está relacionada com a destruição dos objetos do Módulo Aplicação. Esta função é realizada ao final da execução da aplicação ou como resultado da execução da função de controle de tempo, mencionada anteriormente.

6.2.3 Módulo Aplicação

O Módulo Aplicação é responsável pela execução e coordenação da aplicação propriamente dita. Existe uma instância deste módulo para cada usuário conectado ao Gerenciador de Aplicação. É através desta instância que o usuário interage com o Módulo de Dados e Módulo URL.

As funções deste módulo podem ser agrupadas em três grupos:

1. O primeiro grupo compreende as funções relacionadas com a criação, coordenação e destruição das instâncias do Módulo de Dados;
2. O segundo grupo contém as funções relativas à simulação do modelo de transações proposto no capítulo 6. Estas funções serão detalhadas na seção 6.5.
3. O último grupo de funções desempenha as atividades relacionadas com a manutenção da ordem de interação das páginas da aplicação e ao acesso dos objetos do módulo URL, ou seja, estas funções garantem que o usuário siga a ordem de execução das páginas Web, impedindo que certas páginas sejam acessadas antes de outras, e que o estado do navegador fique inconsistente com o estado da base, causado pela utilização dos botões back e forward do navegador. Para controlar a ordem de interação existe uma função responsável pela inscrição de um identificador (variável escondida) na página Web, que identifica a interação que aquela página foi criada. Uma outra função checa se a transição de uma página para outra é valida.

6.2.4 Módulo de dados

O objetivo do módulo de dados é fazer acesso aos dados armazenados no SGBD e disponibilizar primitivas do SGBD ao módulo aplicação para a simulação do modelo de transações proposto. As funções deste módulo podem também ser agrupadas em três grupos:

1. O primeiro grupo de funções está relacionado à abertura da conexão entre a aplicação e o SGBD.
2. O segundo grupo de funções é responsável pela execução das consultas, ou seja, o código SQL.
3. O último grupo corresponde às funções relacionadas com a navegação nos dados retornados pelas consultas.

6.2.5 Módulo URL

O objetivo do módulo URL é definir as páginas Web que irão constituir a aplicação. Este módulo executa o código da lógica da aplicação. Para desempenhar esta tarefa, o módulo disponibiliza uma classe abstrata que serve de molde para a definição das outras classes, utilizando para isto o mecanismo de herança. Cada classe definida representa uma página Web.

Para aumentar o desempenho, os objetos destas classes são instanciados uma única vez, sendo compartilhados por todas as aplicações. Isto é possível pois neste módulo só é necessário especificar o código executável, já que os dados são definidos no módulo de dados e repassados através do módulo aplicação. O ganho de desempenho está relacionado com a não necessidade de ficar criando e destruindo estes objetos.

6.3 Diagrama de objetos

Na seção anterior apresentamos a arquitetura adotada na implementação do gateway, ressaltando as funcionalidades e objetivos gerais de cada módulo.

Nesta seção abordaremos mais detalhadamente as funcionalidades destes módulos, apresentando os objetos que os formam e os relacionamentos entre eles. A figura 6.3 apresenta o diagrama de classes que constituem o gateway.

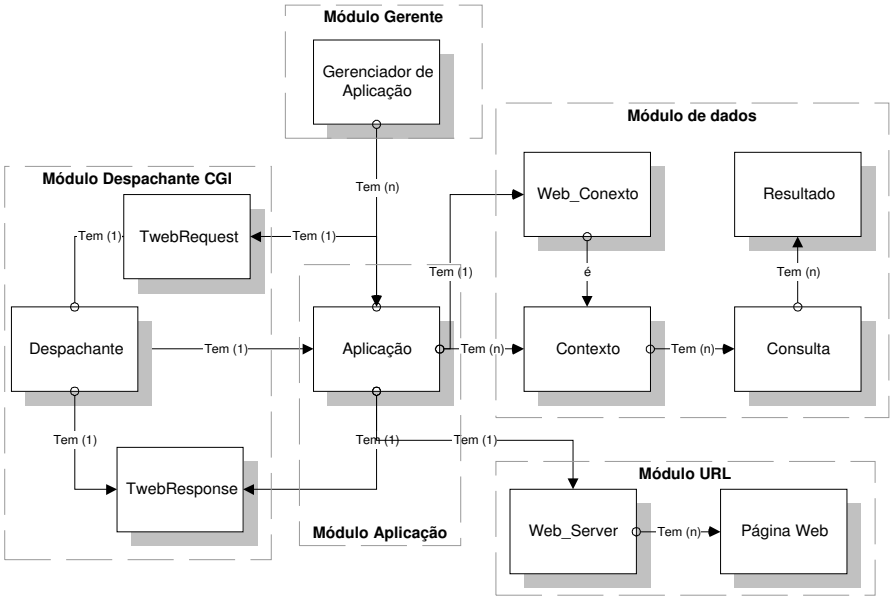

FIGURA 6.3 – DIAGRAMA DE CLASSES DO GATEWAY PROPOSTO

O modelo de diagramação adotado é bastante simples, mas cobre todas as necessidades desta seção. Os objetos são representados através de retângulos e os relacionamentos através de setas com os operadores *"é"* e *"tem"*. A seta indica o sentido do relacionamento e o operador o tipo. O operador *"é"* corresponde ao relacionamento de herança e o *"tem"* ao relacionamento de associação.

O operador *"tem"* requer que seja especificada a cardinalidade da relação. Esta é especificada colocando junto a seta o valor da cardinalidade entre parênteses.

Como podemos observar pelo diagrama da figura 6.3., o gateway proposto é formado por 11 classes, agrupadas segundo o módulo a que pertence. Este agrupamento foi realizado para facilitar a apresentação das classes.

6.3.1 Classes do Módulo Despachante CGI

O Módulo Despachante CGI é composto de três objetos. Estes têm por objetivo fazer o interfaceamento entre o servidor Web e o Gerenciador de Aplicação. A figura 6.3.1 apresenta as interfaces destas classes.

```
interface TwebRequest {  string variavel_ambiente(in string variavel);
    string get_dados_valor_campo(in string campo);
    long get_count();
    boolean get_vazio();
    boolean get_dado_exist(in string campo);
    boolean get_item_exist(in string item);
    string post_dados_valor_campo(in string campo);
    long post_count();
    boolean post_vazio();
    boolean post_dado_exist(in string campo);
    boolean post_item_exist(in string item);
    string cookies_dados_valor_campo(in string campo);
    long cookies_count();
    boolean cookies_vazio();
    boolean cookies_dado_exist(in string campo);
    boolean cookies_item_exist(in string item);
};
  interface TwebResponse {   attribute string conteudo;
    attribute string TipoConteudo;
    attribute string WWWautenticacao;
    void finaliza(in long count);
    boolean SetCookieCampo(in string nome, in string valor, in string
dominio, in string caminho, in string data, in boolean secure);
  };
interface Despachante {  attribute Aplicação Aplic;
    attribute TwebRequest Request;
    attribute Tweb_Response Response;
    Aplicacão Connect(in long count);
  };
```

FIGURA 6.3.1 – DEFINIÇÃO DAS INTERFACES DAS CLASSES DO MÓDULO DESPACHANTE CGI, ESPECIFICADO EM IDL CORBA

A primeira classe é a TwebRequest, que é responsável em formatar os dados passados via interface CGI, e disponibilizá-los à aplicação através de seus métodos. Esta classe possibilita ao usuário desenvolver a aplicação sem conhecer os detalhes da interface CGI. Além disso, permite que o gateway possa utilizar outros mecanismos de comunicação entre o Servidor Web (SSI, API proprietária, ServerLet) e o Gerenciador de Aplicação, sem que seja necessário mudar a implementação dos outros módulos.

Conforme a figura 6.3.1, esta classe possui dezesseis métodos, que podem ser divididos em quatro grupos.

O primeiro grupo corresponde aos métodos utilizados para obter as variáveis ambientes passadas pela interface CGI. Este grupo possui apenas um método.

- *método variariavel_ambiente* – neste método o usuário passa o nome da variável de ambiente desejada, e é retornado o valor da variável na forma de texto.

O segundo grupo corresponde aos métodos relacionados com as variáveis passadas via método GET da interface CGI. Este grupo possui cinco métodos:

- *método get-vazio* – este método indica se existe alguma variável passada via método GET, se existe, é retornado verdade (TRUE), senão falso (FALSE).
- *método get_dados_valor_campo* – neste método o usuário passa o nome da variável, e é retornado o valor na forma de texto.
- *método get_count* – retorna a quantidade de variáveis passadas via método GET.
- *método get_dado_exist* – este método retorna verdade (TRUE) se existe alguma variável com o valor igual ao passado pelo usuário, caso não exista retorna falso (FALSE).
- *método get_item_exist* - este método retorna verdade (TRUE) se existe alguma variável com o nome (identificador) igual ao passado pelo usuário, caso não exista retorna falso (FALSE).

Os métodos do terceiro e quarto grupos têm funções semelhantes aquelas do segundo grupo, todavia os métodos do terceiro grupo tratam os parâmetros passados através do método POST da interface CGI, e os métodos do quarto tratam das variáveis passadas através dos cookies.

Por fim, é importante mencionar, que se o formulário HTML passar variáveis multivaloradas[7], seus valores serão retornados com um texto único separados pelo identificador ";".

A segunda classe é a TwebResponse, utilizada para retornar ao Servidor Web os resultados da execução da aplicação, formatando-os segundo a interface CGI. Esta classe também foi desenvolvida para permitir o usuário desenvolver as aplicações sem conhecer detalhes da interface CGI.

Segundo a figura 6.3.1, a classe possui três atributos e dois métodos.

- *Atributo conteúdo* – este atributo é utilizado para retornar o resultado da aplicação.
- *Atributo tipo_conteudo* – este atributo é utilizado para indicar o tipo de resultado que será retornado pela aplicação.
- *Atributo WWWautenticacao* – este atributo é utilizado para especificar o tipo de autenticação usado para acessar o documento desejado. Estes valores podem ser

obtidos na especificação do HTML versão 3.1.

- *Método finaliza* – utilizado para indicar ao objeto TwebResponse que a interação da aplicação finalizou, e que este pode retornar ao Servidor Web o resultado da aplicação. O atributo count indica qual a interação que esta sendo executada pela aplicação. Este atributo vai ser muito importante para simulação do modelo de transação, como veremos posteriormente.
- *Método SetCookieCampo* – este método é utilizado para facilitar a criação dos cookies, que serão armazenados no navegador Web do usuário. Os parâmetros deste método são os mesmos existentes na especificação de um cookie.

A terceira e última classe deste módulo é a Despachante. O objeto desta classe representa o código executável que será realizado através da interface CGI do Servidor Web. É nesta classe onde são instanciados os objetos TwebResponse e TwebRequest, que serão repassados para a aplicação.

Segundo a figura 6.3.1., a interface desta classe possui três atributos e um método:

- *Atributo Aplic* – representa o objeto aplicação ao qual o despachante está conectado.
- *Atributo Request* – representa a instância do objeto TwebRequest que será repassada para a aplicação.
- *Atributo Response* – representa a instância do objeto TwebResponse que será repassada para a aplicação.
- *Método Connect* – este método é utilizado para fazer a conexão do objeto despachante com o objeto aplicação instanciado pelo Gerenciador de Aplicação. Caso o objeto aplicação referente àquele objeto despachante não tenha sido ainda instanciado, então se conecta com o objeto Gerenciador_Aplic, para criar uma nova instância da classe aplicação, que passará a fazer referência nas próximas interações, até finalizar a aplicação.

[7] Variáveis Multivaloradas – são variáveis com mais de uma valor para um mesmo campo.

6.3.2 Classes do módulo Gerente

Como foi apresentado anteriormente o módulo Gerente tem a função de gerenciar o ciclo de vida dos objetos aplicação. Para desempenhar esta atividade, este módulo apresenta uma única classe, a Gerenciador_Aplic.

```
interface Servidor_Aplic {
Aplicacao Create_Aplic();
Boolean Finaliza_Aplic(in String Id_Aplic)

};
```

FIGURA 6.3.2 – DEFINIÇÃO DAS INTERFACES DAS CLASSES DO MÓDULO GERENTE

A classe Gerenciador_Aplic tem como função principal receber as requisições dos despachantes para criação das instâncias da classe aplicação e retornar para este o identificador do objeto criado. Para desempenhar estas funções, a classe possui dois métodos, como apresentado na figura 6.3.2.

- *Método Create_Aplic* – este método é utilizado pelo objeto despachante, para requisitar ao objeto Gerenciador_Aplic que crie uma instância da classe Aplicação, e retorne para este a instância criada. Este método pode parecer simples, mas requer um tratamento especial, no que se refere a criação de um identificador único para cada instância da aplicação, uma vez que é através deste identificador que o objeto despachante fará as próximas requisições diretamente ao objeto aplicação.

- *Método Finaliza_Aplic* – este método é utilizado para destruir uma instância da classe aplicação, e finalizar as operações pendentes com o SGBD. O parâmetro id_aplic representa o identificador do objeto aplicação que será destruído. Este método pode ser requisitado pelo próprio Gerenciador_Aplic ou pelo despachante. No caso do Gerenciador_Aplic, isto ocorre quando constata-se que o tempo de execução do objeto aplicação é maior que o permitido, podendo-se concluir que talvez a conexão entre o usuário e a aplicação tenha se perdido, requerendo assim, a finalização da aplicação. No caso do despachante, este método é executado para indicar que o usuário finalizou todas as operações e o objeto aplicação já pode ser destruído.

Por fim, deve-se ressaltar que para efeito de um melhor balanceamento de carga, cada instância da classe Gerenciador_Aplic é responsável em gerenciar um tipo de aplicação, permitindo que as aplicações sejam distribuídas em várias máquinas. Para cada aplicação desenvolvida existe um objeto Gerenciador_Aplic.

6.3.3 Classes do Módulo Aplicação

Semelhante ao Módulo Gerenciador de Aplicação, existe uma única classe pertencente a este módulo, a classe Aplicação. Esta classe é responsável em gerenciar e coordenar a aplicação propriamente dita, pois é através dela que o usuário tem acesso às páginas Web e aos dados armazenados no SGBD.

```
interface Aplicacao {
attribute TwebRequest request;
attribute TwebResponse response;
Contexto Get_Contexto(in string name);
Contexto Get_Web_Contexto(in string name);
Contexto Create_Contexto(in string name, in string datasource,in
         string db,in string id,in string pw);
Contexto Create_Web_Contexto(in string name, in string datasource,in
         string db,in string id,in string pw);
string Get_name();
boolean execute();
boolean Commit();
};
```

FIGURA 6.3.3 – DEFINIÇÃO DAS INTERFACES DAS CLASSES DO MÓDULO APLICAÇÃO

Conforme a figura 6.3.3, esta classe possui dois atributos e sete métodos.

- *Atributo request* – este atributo representa a instância da classe TwebRequest, repassada pelo objeto despachante ao objeto aplicação, onde é possível obter os dados passados pelo navegador Web através da interface CGI do servidor Web.
- *Atributo response* – representa o objeto TwebResponse repassado pelo objeto despachante ao objeto aplicação. É através deste objeto que é retornado o resultado da execução da aplicação.
- *Método Create_Contexto* – este método é responsável em criar uma instância do objeto contexto. Como veremos adiante, este objeto representa uma conexão entre aplicação e o SGBD.
- *Método Create_Web_Contexto* – este método é responsável em criar uma

instância da classe Web_Contexto. Como veremos adiante, este objeto também pode ser utilizado para abrir uma conexão entre aplicação e o SGBD.

- *Método Get_Contexto* – este método é utilizado para retornar a instância da classe contexto, que tem o mesmo nome (identificador) que o indicado no parâmetro name.
- *Método Get_Web_Contexto* – este método é utilizado para retornar a instância da classe web_contexto. Neste método não é preciso indicar o identificador do objeto, pois só existe uma única instância do objeto web_contexto por aplicação. No caso do objeto contexto pode haver mais de uma.
- *Método Get_ident* – este método é utilizado para retornar o texto que representa o identificador o objeto aplicação.
- *Método execute* – é através deste método que o usuário indica que objeto Web_Page deseja acessar. O usuário indica este objeto através do valor da variável SUBMIT, especificada no formulário HTML.
- *Método Commit* – este método é utilizado para indicar o fim da aplicação, finalizando todas as operações com o SGBD.

6.3.4 Classes do módulo URL

Como apresentado, o módulo URL é responsável pela definição das páginas Web que formam a aplicação. Para desempenhar esta tarefa foram definidas duas classes: a Web_Server e Web_Page. A figura 6.3.4 apresenta a interface destas classes.

```
interface Web_Page{
boolean execute(in Aplicacao Aplic);
};
interface WebServer {
 Web_Page Get_Web_Page(in string estado_corrente, in string novo_estado);
};
```

FIGURA 6.3.4 – DEFINIÇÃO DAS INTERFACES DAS CLASSES DO MÓDULO APLICAÇÃO

A classe WebServer é responsável em retornar ao objeto aplicação o objeto WebPage requerido. É nesta classe que é checado se a transição de uma página para a outra é válida. Conforme a figura 6.3.4, esta classe possui um único método.

- *Método Get_Web_Page* – este método é utilizado pelo objeto aplicação para retornar a instância da classe Web_Page que possui o mesmo identificador que o passado através do atributo novo_estado.

A classe Web_Page é uma classe abstrata que serve de molde para se desenvolver as classes que vão representar os documentos Web. Para que uma classe seja um documento Web, basta que esta herde da classe Web_Page. A figura 6.3.4 ilustra a interface da classe Web_Page, que possui um único método.

- *Método execute* – este método permite ao usuário definir o código da aplicação que será executado quando este for solicitado. Cada classe que herda da Web_Page, deve redefinir este método, inserido o código que deve ser executado quando o objeto for ativado.

6.3.5 Classes do módulo de dados

As classes do módulo de dados são responsáveis em fazer o acesso aos dados armazenados no SGBD. A figura 6.3.5 apresenta a interface destas classes. As classes foram definidas tendo como modelo as classes existentes na API JDBC [Sha96], presentes na linguagem Java.

A primeira classe definida é classe Contexto, que é utilizada para abrir uma conexão com o SGBD. A conexão aberta é do tipo autocommit, ou seja, todas as operações SQL irão ser executadas e validadas como transações individuais. Esta classe é formada por dez métodos.

```
interface Resultado { boolean next();
     void close();
     boolean wasNull();
     string getString(in long columnIndex);
     boolean getBoolean(in long columnIndex);
     octet getOctet(in long columnIndex);
     short getShort(in long columnIndex);
     long getInt(in long columnIndex);
     float getFloat(in long columnIndex);
     double getDouble(in long columnIndex);
     string getStringStr(in string columnName);
     boolean getBooleanStr(in string columnName);
     octet getOctetStr(in string columnName);
     short getShortStr(in string columnName);
     long getIntStr(in string columnName);
     float getFloatStr(in string columnName);
     double getDoubleStr(in string columnName);
     string getCursorName();
     Object getObject(in long columnIndex);
     Object getObjectStr(in string columnName);
     long findColumn(in string columnName);
};
interface Consulta{ Resultado Get_Resultado(in string name);
    Resultado executeQuery(in string sql,in string ident) ;
    long executeUpdateSql(in string sql) ;
    boolean execute(in string sql);
    void cancel();
    void close() ;
    long getMaxFieldSize() ;
    void setMaxFieldSize(in long max) ;
    long getMaxRows() ;
    void setMaxRows(in long max) ;
    void setEscapeProcessing(in boolean enable) ;
    long getQueryTimeout() ;
    void setQueryTimeout(in long seconds) ;
    void setCursorName(in string name) ;
    boolean getMoreResults() ;
    void Compensa(in string sql);
};
interface Contexto { string nativeSQL(in string sql) ;
     void close();
     boolean isClosed() ;
     Consulta Get_Consulta(in string name);
     Consulta Create_Consulta(in string name);
     void setReadOnly(in boolean readOnly);
     boolean isReadOnly();
     void setCatalog(in string catalog) ;
     string getCatalog() ;
};
interface Web_Contexto : Contexto {   void commit() ;
   void rollback(in long ponto) ;
};
```

FIGURA 6.3.5 – DEFINIÇÃO DAS INTERFACES DAS CLASSES DO MÓDULO DE DADOS

- *Método nativeSQL* – neste método o usuário passa uma consulta SQL e retorna o código nativo que irá ser executado no SGBD.
- *Método close* – utilizado para fechar a conexão com o SGBD.
- *Método isclose* – checa se a conexão esta fechada.

- *Método Create_Consulta* – cria uma instância do objeto consulta, com o identificador passado pelo atributo name.
- *Método Get_Consulta* – retorna o objeto consulta pertencente aquele objeto Contexto, que possui o mesmo identificador, passado através do atributo name.
- *Método setReadOnly* – indica que a conexão ativa só é para leitura.
- *Método isReadOnly* – checa se a conexão ativa é do tipo ReadOnly (somente leitura).
- *Método getCatalog* – utilizado para se obter as informações dos metadados armazenados no SGBD.
- *Método setCatalog* – utilizado para alterar os metadados armazenados no SGBD, passado através do atributo catalog.

A segunda classe é a Web_Contexto, que herda da classe Contexto. A diferença desta classe para sua classe pai é que a conexão com o SGBD não é do tipo autocommit, o usuário tem que especificar explicitamente o commit, para validar os resultados das consultas SQL. Esta classe acrescenta dois métodos.

- *Método commit* – este método é utilizado para validar os resultados das consultas SQL.
- *Método rollback* – este método é utilizado para desfazer os resultados da transação, até o ponto especificado através do parâmetro ponto.

A terceira classe é a Consulta, que é utilizada para executar as consultas SQL junto ao SGBD. Esta classe possui 15 métodos.

- *Método executeQuery* – este método é utilizado para executar consultas que retornam resultados. Os resultados são retornados através do objeto Resultado. O parâmetro sql é utilizado para indicar a consulta SQL a ser executada e o Ident representa o identificador que será atribuído ao objeto Resultado.
- *Método Get_Resultado* – retorna o objeto Resultado referente aquele objeto Consulta, que possui o mesmo identificador do parâmetro name.
- *Método executeUpdateSql* – utilizado para executar consultas SQL que façam

atualizações (inserções, deleções ou atualizações).

- *Método execute* – utilizado para executar consultas SQL que retorna mais de um resultado.
- *Método cancel* – cancela a execução de uma consulta.
- *Método close* – libera os recursos utilizados na execução da consulta.
- *Método getMaxFieldSize* – retorna o número máximo de bytes possíveis de serem retornados por um campo da tabela.
- *Método setMaxFieldSize* – seta o número máximo de bytes possíveis de serem retornados por um campo da tabela.
- *Método getMaxRows* – retorna o número máximo de tuplas possíveis a serem retornados por uma consulta..
- *Método setMaxRows* – seta o número máximo de tuplas possíveis a serem retornados por uma consulta..
- *Método getQueryTimeout* – retorna o número máximo de tempo, em segundos, que uma consulta deve ser executada.
- *Método setQueryTimeout* – seta o número máximo de tempo, em segundos, que uma consulta deve ser executada.
- *Método setCursorName* – cria um nome para o cursor SQL.
- *Método getMoreResults* – retorna o próximo resultado da consulta.
- *Método compensa* – este método é utilizado para indicar a transação compensatória

A última classe é a Resultado, que é utilizada para navegar nos resultados retornados pelo objeto Consulta. Esta classe possui 21 métodos.

- *Método next* – este método é utilizado para mover-se de uma tupla para a outra. Faz com que o curso aponte para a próxima tupla. A primeira vez que este método é executado, move o cursor para a primeira tupla.
- *Método close* – libera todos os recursos utilizados pelo objeto Resultado.
- *Método wasNull* – este método é utilizado para saber se o valor lido pelo método

GETXXX é nulo (NULL) .

- *Método getString* – este método é utilizado para obter o valor da coluna, da tupla corrente, como uma string. O atributo columnIndex indica a posição da coluna.
- *Método getBoolean* – este método é utilizado para obter o valor da coluna, da tupla corrente, como um boolean. O atributo columnIndex indica a posição da coluna.
- *Método getOctet* – este método é utilizado para obter o valor da coluna, da tupla corrente, como um vetor de bytes. O atributo columnIndex indica a posição da coluna.
- *Método getShort* – este método é utilizado para obter o valor da coluna, da tupla corrente, como um inteiro curto. O atributo columnIndex indica a posição da coluna.
- *Método getInt* – este método é utilizado para obter o valor da coluna, da tupla corrente, como um inteiro. O atributo columnIndex indica a posição da coluna.
- *Método getFloat* – este método é utilizado para obter o valor da coluna, da tupla corrente, como um real. O atributo columnIndex indica a posição da coluna.
- *Método getDouble* – este método é utilizado para obter o valor da coluna, da tupla corrente, como um real duplo. O atributo columnIndex indica a posição da coluna.
- *Método getObject* – este método é utilizado para obter o valor da coluna, da tupla corrente, como um objeto da classe object da linguagem Java. O atributo columnIndex indica a posição da coluna.
- *Método getStringSt*r – este método é utilizado para obter o valor da coluna, da tupla corrente, como uma string. O atributo columnName indica o nome da coluna.
- *Método getBooleanStr* – este método é utilizado para obter o valor da coluna, da tupla corrente, como um boolean. O atributo columnName indica o nome da coluna.
- *Método getOctetStr* – este método é utilizado para obter o valor da coluna, da tupla corrente, como um vetor de bytes. O atributo columnName indica o nome da coluna.

- *Método getShortStr* – este método é utilizado para obter o valor da coluna, da tupla corrente, como um inteiro curto. O atributo columnName indica o nome da coluna.
- *Método getIntStr* – este método é utilizado para obter o valor da coluna, da tupla corrente, como um inteiro. O atributo columnName indica o nome da coluna.
- *Método getFloatStr* – este método é utilizado para obter o valor da coluna, da tupla corrente, como um real. O atributo columnName indica o nome da coluna.
- *Método getDoubleStr* – este método é utilizado para obter o valor da coluna, da tupla corrente, como um real duplo. O atributo columnName indica o nome da coluna.
- *Método getObjectStr* – este método é utilizado para obter o valor da coluna, da tupla corrente, como um objeto da classe object da linguagem Java. O atributo columnName indica o nome da coluna.
- *Método getCursorName* – retorna o nome do cursor.
- *Método findColumn* – neste método, o usuário indica o nome do campo da tabela, e é retornado o índice da posição deste campo na tabela.

6.4 Funcionamento do gateway

As seções anteriores apresentaram os objetivos e funcionalidades de cada módulo do gateway, bem como, as classes que o implementam. Esta seção, explicará como os objetos destes módulos interagem entre si, para que o gateway possa desempenhar suas funções.

O algoritmo de funcionamento do gateway é mostrado a seguir e a figura A7 dos anexos ilustra estes passos:

1° Passo: O cliente Web solicita ao servidor Web a execução de um aplicativo externo que neste caso é um Gerenciador de Aplicação.

2° Passo: O servidor Web inicia um processo para um dos despachantes (instancia o objeto despachante), que compõe o primeiro módulo da aplicação, através da interface padrão CGI, enviando os dados passados pelo cliente Web de acordo com a

implementação do servidor Web para interface CGI.

3° Passo: O objeto despachante instancia os objetos TwebRequest e o TwebResponse. O objeto TwebRequest é utilizado para acessar os dados passados via interface CGI e o TwebResponse para retornar o resultado da aplicação ao servidor Web.

4° Passo: O objeto Despachante checa, via TwebRequest, se existe o cookie id_aplic. Este cookie contém o identificador o objeto aplicação. Caso não exista o cookie é executado o passo 5, senão pula para o passo 6.

5° Passo: Não havendo o cookie id_aplic, o despachante conecta-se com objeto Servidor_Aplic para que este crie uma nova instância da classe Aplicação. Instanciado o objeto, o Servidor_Aplic cria um identificador único para este objeto, e retorna-o para o despachante. Pular para o passo 7.

6° Passo: Havendo o cookie id_Aplic, o despachante utiliza-o para acessar o objeto Aplicação, sem necessitar da intermediação do Servidor_Aplic.

7° Passo: Estando conectado com a Aplicação, o despachante repassa os objetos TwebResponse e o TwebRequest para o objeto Aplicação e ativa o método execute da Aplicação.

8° Passo: Ativado o método execute, o objeto Aplicação checa se o estado (a interação) do navegador do cliente encontra-se no mesmo que o seu. Caso não esteja, o objeto Aplicação faz um roolback do seu estado até o estado do navegador do cliente.

9° Passo: Estando agora no mesmo estado, o objeto Aplicação avalia se a transição da página atual para a página requerida é válida. Se não for, encerra-se a execução da aplicação e retorna uma menagem de erro. Senão, é feita uma requisição ao objeto WebServer pelo objeto Web_Page indicado pelo usuário, através da variável submit do formulário HTML.

10° Passo: Obtida a instância do objeto Web_page, o objeto Aplicação ativa o método execute, passando como parâmetro a si próprio.

11° Passo: O objeto Web_Page recebe a requisição da ativação do método, e executa o código da aplicação. É nesse ponto onde é feito o acesso ao banco de dados.

12° Passo: Finalizado o método execute do objeto Web_page, é retornado ao objeto Aplicação um código indicando se todas as operações foram executadas com sucesso ou houve alguma falha. Este código por sua vez é repassado para o despachante.

13° Passo: Antes de finalizar a execução, o objeto despachante checa se esta foi a última interação da aplicação. Caso seja verdade, este requer ao Servidor_Aplic a destruição do objeto Aplicação.

14° Passo: Recebido o sinal de finalização, o despachante checa o tipo do retorno e repassa o resultado da aplicação e os dados de controle (variáveis escondida e cookies) para o servidor Web, utilizando o objeto TwebResponse.

15° Passo: O Servidor Web repassa para o cliente Web o documento retornado pelo Gerenciador de Aplicação.

6.5 Implementação

Até o presente instante foi apresentada uma visão abstrata do Gerenciador de Aplicação, ressaltando os objetos que o constitui, juntamente com os seus relacionamentos.

Esta seção apresentará detalhes de como foram implementadas estas classes e como foi simulado o modelo de transações proposto no capítulo 6.

6.5.1 Tecnologias Utilizadas

As tecnologias utilizadas para o desenvolvimento do gateway foram Java e CORBA.

A linguagem Java foi utilizada devida a sua grande portabilidade, capacidade de acessar Banco de Dados, via JDBC e pela fácil integração com o CORBA. Para aumentar o desempenho do Gerenciador de Aplicação, utilizou-se uma ferramenta que converte os bytecodes do Java em código executável.

O padrão CORBA foi utilizado para exportar e gerenciar as funcionalidades dos objetos desenvolvidos em Java, permitindo assim, que o desenvolver das aplicações utilizem os objeto do gateway sem se preocupar com detalhes de comunicação entre eles.

Por exemplo, no caso dos objetos TwebRequest e TwebResponse, estes são compartilhados entre os objeto Despachante e Aplicação de forma transparente, mesmo pertencendo a processos diferentes, em diferentes máquinas.

Outra vantagem do CORBA é a possibilidade e facilidade de estender o gateway para possibilitar o acesso direto entre o navegador Web e o Gerenciador de Aplicação sem passar pelo servidor Web, utilizando para isso o protocolo IIOP (Internet Inter-ORB Protocol) [Orf98] para fazer a comunicação entre eles.

Deve-se ressaltar, que a linguagem Java poderia ser utilizada unicamente, utilizando a interface RMI (Remote Method Invocation) [OH97] para exportar e gerenciar os objetos. Entretanto com a utilização do CORBA é possível desenvolver a aplicação em qualquer linguagem que o suporte, não ficando restrito unicamente a Java. Por exemplo, poder-se-ia desenvolver os objetos do Módulo Despachante CGI em C++, aumentando o desempenho dos despachantes, e os outros módulos em Delphi, para utilizar os recursos de acesso a banco de dados existentes na linguagem, sem se preocupar como realizar a interoperabilidade entre objetos dessas linguagens e sem se preocupar com a localização destes. Com a utilização do CORBA foi possível desenvolver um gateway independente da linguagem de programação.

6.5.2 Implementação do modelo de transações

Nesta seção vamos apresentar o processo de implementação do modelo de transações proposto no capítulo 6.

A primeira solução imaginada foi utilizar algum sistema gerenciador de transação que suportasse o modelo proposto. Entretanto esta solução foi logo abandonada, uma vez, que ia de encontro ao pré-requisito da não alteração das tecnologias envolvidas. Assim, foi implementado no próprio gateway um mecanismo que simula o modelo proposto, utilizando somente os recursos transacionais existentes nos SGBDs tradicionais.

Para uma melhor compreensão, dividiu-se a implementação do modelo em três fases. A primeira fase compreende o processo de identificação da aplicação. A segunda fase o processo de simulação das subtransações seqüenciais e a terceira fase a simulação das subtransações compensatórias.

6.5.2.1 Processo de identificação

A necessidade de criar um mecanismo de identificação é referente à impossibilidade de existir uma conexão constante entre o navegador Web e a aplicação, utilizando a interface CGI. Assim, deve existir alguma forma que o navegador Web possa identificar o objeto Aplicação associado a ele, durante as várias interações da aplicação.

Para resolver o problema utilizamos um cookie (id_aplic) armazenado no navegador Web. Este é criado e retornado ao navegador durante a primeira interação da aplicação. Nas interações subsequentes o despachante utiliza o cookie para acessar diretamente o objeto Aplicação.

Uma dificuldade enfrentada na implementação da solução encontrada consiste da necessidade de garantir a unicidade do identificador, uma vez que pode ocorrer o caso onde dois despachantes solicitem ao mesmo tempo a criação de uma aplicação, configurando-se numa região crítica.

Para resolver este problema foram criados uma tabela e dois procedimentos armazenados (*Stored Procedure*), conforme as figura 6.5.2.1.a e 6.5.2.1.b, respectivamente.

```
CREATE TABLE Tab_Aplicacao (
      id_aplic                int          NOT NULL ,
      data_inicio_aplic       datetime     NULL ,
      data_fim_aplic          datetime     NULL,
      PRIMARY KEY (id_aplic )
);
```

FIGURA 6.5.2.1.A – DEFINIÇÃO DA TABELA TAB_APLICACAO, USANDO SQL

A tabela Tab_Aplicação contém três campos. O primeiro campo, Id_Aplic, é um inteiro, e representa o código do identificador da aplicação. O segundo e o terceiro campo são do tipo datetime e representam o tempo de criação e destruição do objeto aplicação, respectivamente. Estes são utilizados para o controle de tempo de execução da aplicação.

```
CREATE PROCEDURE NOVA_APLIC AS
DECLARE @ID INTEGER

BEGIN TRANSACTION
IF (SELECT COUNT(ID_APLIC)
FROM Tab_APLICACAO) = 0 BEGIN SET @ID=0 END
ELSE
BEGIN
SET @ID= ( SELECT MAX(ID_APLIC)
FROM Tab_APLICACAO)
END
SET @ID=@ID+1;
INSERT INTO Tab_APLICACAO
VALUES(@ID,GETDATE(),NULL)
SELECT @ID
COMMIT
GO

CREATE PROCEDURE DESTROY_APLIC (@ID INTEGER) AS

BEGIN TRANSACTION
UPDATE Tab_APLICACAO
SET data_fim_aplic = GETDATE()
WHERE id_aplic = @ID
COMMIT
GO
```

FIGURA 6.5.2.1.B – DEFINIÇÃO DOS PRODIMENTOS DESTROY_APLIC E NOVA_APLIC

O primeiro procedimento, NOVA_APLIC, insere na tabela uma nova tupla contendo o identificador da aplicação e o horário da criação. O valor do identificador é obtido selecionando o maior valor do campo *id_aplic*, existente nas tupla da tabela Tab_Aplicação, acrescentado de uma unidade. Ao final do procedimento é retornado o valor do identificador criado. Este procedimento é executado durante a execução do método Create_Aplic. Deve-se ressaltar que, caso o SGBD suporte o tipo auto-incremento, este procedimento pode ser substituído definindo o campo *id_aplic* como do tipo auto-incremento e criando um gatilho para que quando for inserida uma nova tupla na tabela NOVA_APLIC seja atribuída a data de criação.

O outro procedimento, DESTROY_APLIC, atualiza a tupla com o valor do horário de finalização da aplicação, indicando que o objeto Aplicação já foi destruído. A transação é executada durante a execução do método Finaliza_Aplic.

6.5.2.2 Processo de simulação das subtransações seqüenciais

Como apresentado no capítulo anterior, uma subtransação seqüencial representa um subconjunto das operações de uma transação global (transação pai), que são executadas numa mesma interação. Assim, se uma transação envolver duas interações (duas páginas), esta será quebrada em duas subtransações seqüenciais, que agrupam as operações executadas em cada interação.

Existem dois problemas para simular estas subtransações utilizando os recursos transacionais existentes nos SGBDs tradicionais. O primeiro diz respeito ao nível das operações transacionais no modelo tradicional, onde todas as operações estão no mesmo nível, não existem subníveis. Assim, para simular as subtransações seqüenciais é necessário utilizar algum mecanismo existente no SGBD para quebrar a transação plana em vários pontos de reabilitação.

O segundo problema trata da associação entre uma subtransação seqüencial a uma página Web, ou seja, como relacionar as operações que foram executadas durante a interação de uma determinada página Web. Isto é importante, pois se o usuário executar um *backward* de uma página, é necessário que as operações associadas àquela página sejam desfeitas.

6.5.2.2.1 Níveis de reabilitação

Para quebrar a transação plana em vários níveis de reabilitação, utilizamos um mecanismo bastante comum hoje nos SGBDs comerciais, o *savepoint*. Tal mecanismo está presente na especificação do SQL versão 3.

Os *savepoints* possibilitam ao usuário especificar vários pontos de reabilitação, permitindo desfazer apenas parte da transação. Este é estabelecido invocando a função *Save Work*, que solicita ao sistema armazenar o estado corrente do processamento. O sistema por sua vez, retorna para a aplicação um identificador, que pode ser utilizado para fazer referência àquele *savepoint*. É através deste identificador que o usuário pode retornar para o estado do *savepoint*. Para voltar, o usuário utiliza o comando *Roolback Work*, passando como parâmetro o identificador do *savepoint* ao qual deseja retornar.

No caso da simulação das transações seqüenciais, utiliza-se o *savepoint* para delimitar as operações pertencentes a cada interação, ou seja, ao final de cada interação (final do método execute do objeto Web_Pag) é executado o comando Save Work, criando assim, para cada interação, um *savepoint*. A figura 6.5.2.2.1 ilustra esta idéia.

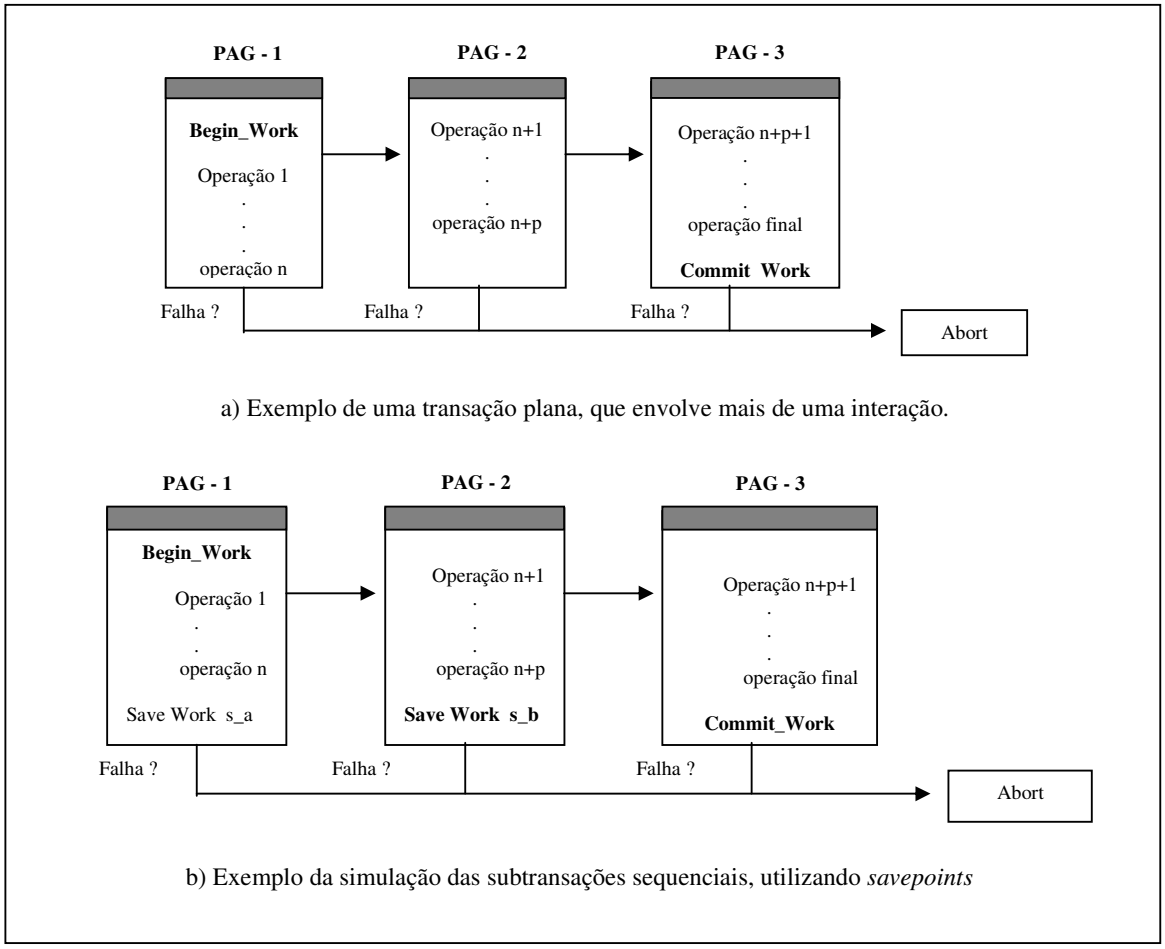

FIGURA 6.5.2.2.1 – EXEMPLO DE COMO UTILIZAR SAVEPOINTS PARA A SILUMAÇÃO DAS SUBTRANSAÇÕES SEQUENCIAIS

No gateway, a inserção dos *savepoints* é transparente. O usuário executa suas operações como se estivesse se servindo do modelo tradicional (transações planas) e o próprio gateway insere os *savepoints* ao final de cada interação.

Devido à utilização do *savepoint,* torna-se possível ao usuário desfazer apenas as operações executadas em cada interação. Por exemplo, no caso da figura 6.5.2.2.1.b, se o

usuário estiver na página 2 e resolver voltar para a página 1, é possível desfazer somente as operações da página 2. Para isto, basta executar o comando *Roolback Work* para o *savepoint* s_a.

6.5.2.2.2 Mecanismo de roolback de uma página

Definidos os vários pontos de reabilitação, surge um questionamento: Como relacionar um *savepoint* com uma página Web e como detectar um *backward* do usuário durante a navegação nas páginas da aplicação?

Para solucionar o problema criamos um identificador para cada página pertencente a aplicação, sendo o mesmo utilizado para identificar o *savepoint* criado durante a execução daquela página. Este identificador é um contador monotômico que corresponde a interação que a página foi criada. Por exemplo, se uma página for criada na segunda interação da aplicação, esta terá o identificador 2 armazenado no documento HTML e caso tenha ocorrida alguma operação junto ao SGBD, haverá um *savepoint* com o identificador 2.

Assim, se o usuário estiver navegando em uma página e utilizar o botão *back* do navegador, para retornar a página anterior, basta executar o comando *Rollback Work* com o identificador daquela página para voltar ao estado a que se encontrava quando foi criada.

Para armazenar o identificador no documento HTML, utilizou-se uma variável escondida (<hidden>), id_count, que contém o valor da interação.

Este identificador também é utilizado para detectar se ocorreu algum *backward* por parte do navegador. Durante a submissão da execução de uma página, o objeto Aplicação checa se o identificador da página que solicitou a interação é igual ao contador existente neste. Caso seja menor, o objeto Aplicação executa o comando *Roolback Work* com o identificador da página e atualiza o contador da aplicação com o mesmo valor.

Devido ao fato do evento *backward* somente ser tratado pelo gateway quanto o usuário submeter uma nova requisição para este, pode ocorrer o caso onde o estado do navegador esteja inconsistente com a do SGBD, entretanto esta situação é temporária até que o usuário solicite uma nova interação com a aplicação, caso isto não ocorra transação será abortada, eliminando assim qualquer inconsistência na base de dados.

A utilização deste mecanismo permitiu a simplificação do gateway em dois pontos:

1. O primeiro ponto é a não necessidade implementar qualquer mecanismo no navegador que informasse ao gateway cada ocorrência de um evento navegacional (*backward e forward*).
2. O segundo ponto é mais interessante, pois com a utilização deste mecanismo não há necessidade de se tratar o evento *forward*, pois nunca ocorrerá o caso em que o identificador da página Web seja maior que o do objeto Aplicação, já que sempre a última página terá o maior identificador.

Por fim, para finalizar o processo de simulação das subtransações seqüenciais, deve-se ressaltar que este mecanismo só é suportado quando se utiliza o objeto Web_Contexto para abrir a conexão com o SGBD e que só existe um objeto desta classe por objeto Aplicação.

6.5.2.3 Processo de simulação das subtransações compensatórias

Conforme apresentado no capítulo 5, uma transação C é dita compensatória de uma transação T, caso corresponda semanticamente ao efeito do *undo* de T, depois de T ter sido validada, não precisando necessariamente restaurar o banco de dados para o estado a que se encontrava quando T começou a executar [Gra81]. Transações compensatórias são úteis para atividades de longa duração, pois permite liberar os resultados das subtransações antes que a transação global finalize.

Para simular as transações compensatórias dois problemas devem ser resolvidos. O primeiro é como garantir que a transação compensatória será ativada caso seja necessário, e o segundo, como relacionar a transação compensatória com a página e a aplicação que a criou.

Para solucionar esses problemas criamos uma tabela no SGBD, que será utilizada para armazenar o código SQL referente à execução da transação compensatória. Assim, se ocorrer algum problema com o Gerenciador de Aplicação, as transações compensatórias não são perdidas, podendo ser executadas para desfazer o efeito das transações a que estavam associadas. A figura 6.5.2.3.a apresenta o código SQL da tabela criada.

```
CREATE TABLE Tab_Compensa (
      id_aplic    int          NOT NULL ,
      id_count    int          NOT NULL ,
      sql_comp    varchar      NULL ,
      PRIMARY KEY (id_aplic, id_count ),
      FOREIGN KEY (id_aplic) REFERENCES Tab_Aplicacao (id_aplic)
);
```

FIGURA 6.5.2.3.A – DEFINIÇÃO DA TABELA TAB_COMPENSA, USANDO SQL

A tabela criada possui três campos. O primeiro campo representa o identificador da aplicação. Este campo é chave estrangeira da tabela Tab_Aplicacao, definida na seção 6.5.2.1. O segundo campo é o identificador da página, a partir da qual a transação compensatória foi criada. O último campo é utilizado para armazenar o código SQL da transação compensatória.

Para criar uma transação compensatória o usuário indica ao objeto Consulta que a transação é compensável, utilizando o método compensa, passando como parâmetro o código SQL que será executado caso esta transação tenha que ser desfeita.

Durante a execução da transação, é inserido um comando SQL, que solicita a inserção do código passado pelo método compensa, dentro da tabela Tab_Compensa. A figura 6.5.2.3.b ilustra esta operação.

```
BEGIN TRANSACTION
//Comandos da execução da transação
operação 1
.
.
operação N
//Comando da execução da transação compensatórias
INSERT INTO Tab_compensa
VALUES (&id_aplic, &id_count, &sql_compensa);

COMMIT TRANSACIONT
```

FIGURA 6.5.2.3.B – DEFINIÇÃO DO ESQUEMA DE INSERÇÃO DA TRANSAÇÃO COMPENSÁTOIA

Desta forma é possível garantir que se a transação original for executada com sucesso, então a transação compensatória foi inserida na tabela Tab_compensa, podendo ser ativada.

A transação compensatória pode ser ativada em três casos:

1. Se o usuário solicitar que o objeto Aplicação execute o método roolback, para desfazer todas as operações executadas pela aplicação. Neste caso, o objeto Aplicação executa um comando SQL, selecionando as tuplas da tabela Tab_compensa, que possui o mesmo identificador do objeto Aplicação, classficado em ordem crescente do campo id_count. Selecionadas as tuplas, é montada uma transação contendo todos os valores do campo codigo_sql. Ao final desta transação é inserido um código que exclui as tuplas da tabela compensa que possui o mesmo identificador do objeto Aplicação.
2. Se o Gerenciador de aplicação falhar. Assim, quando o Gerenciador de aplicação for reiniciado, este seleciona todas tuplas da tabela Tab_compensa classificado em ordem crescente do campo id_aplic, e gera uma transação contendo todos o valores dos campos codigo_sql. Ao final da transação é inserido um código SQL para apagar todas as tuplas da tabela Tab_compensa.
3. O terceiro caso ocorre quando o usuário executa um backward durante a navegação nas páginas da aplicação. Assim, quando o objeto Aplicação detectar um backward, é executado um comando SQL selecionando as tuplas que possuem o mesmo identificador do objeto Aplicação e que possui o identificador de interação (id_count) maior que o existente na variável escondida da página HTML, classificado em ordem crescente do campo id_count. Selecionada as tuplas é montada uma transação contendo todos os valores do campo codigo_sql. Ao final da transação também é inserida uma operação apagando todas as tuplas da tabela, que possuem o mesmo identificador da aplicação e o identificador da interação maior que o existente na variável escondida id_count.

Por fim deve-se ressaltar que esta propriedade somente é suportada pelas consultas pertencente ao objeto Contexto.

6.6 Conclusão

Nesta seção iremos apresentar um breve resumo e os principais resultados obtidos.

1. **Aspectos transacionais:** são os aspectos mais relevantes na implementação do

Gateway, pois devido a manutenção da conexão é possível o usuário utilizar todos os recursos existentes no SGBD, como por exemplo, o controle de concorrência, uso de cursores, otimização de consultas, entre outros. Além disso, com a implementação do modelo de transações proposto, foram eliminadas várias restrições impostas pelo modelo de transações planas, permitindo um melhor controle do estado da aplicação e uma melhor adaptação as características do ambiente de integração.

2. **Segurança/Acesso:** O único mecanismo de segurança/acesso implementado no gateway foi o controle na ordem da execução das páginas Web, forçando o usuário a seguir a lógica da aplicação. Este mecanismo foi implementado no Módulo Aplicação e WebServer.

   Não foi implementado nenhum controle no acesso de usuários ao Gerenciador de Aplicação, e não existem garantias na autenticação e privacidade da operações. Entretanto vários mecanismos podem ser facilmente implementados no gateway, como por exemplo:

   - O servidor Web pode autenticar o usuário e passar ao Gerenciador de Aplicação o login do usuário para ser utilizado para acesso ao Banco de Dados.
   - O Gerenciador de Aplicação pode implementar uma tabela contendo o login e senha dos usuários que podem acessá-lo, e fazer o mapeamento deste para um usuário do Banco de Dados.

3. **Desempenho:** Vários mecanismos foram implementados para melhorar o desempenho:

   - O primeiro está relacionado com a manutenção da conexão. Assim não é necessário ficar abrindo e fechando conexão com o SGBD.
   - O segundo mecanismo está relacionado com o mapeamento de usuários de logins diferentes para um único login no Banco de Dados já iniciado, não necessitando criar uma nova conexão para cada novo usuário. Para implementar este mecanismo o usuário deve definir que a aplicação possui um objeto Contexto do tipo estático. Especificando assim, que este objeto

será compartilhado entre todos objetos Aplicação. Não é possível definir que um objeto Web_Contexto seja do tipo estático.

- Outro mecanismo que otimiza o desempenho é a existência de um Gerenciador de Aplicação para cada aplicação desenvolvida, permitindo assim um melhor balanceamento de carga entre os Gerenciadores de Aplicação.
- Para eliminar o problema do desempenho da linguagem Java, utilizamos uma ferramenta (Visual Café), que converte os bytes codes do Java em código nativo, para gerar o executável do Gerenciador de Aplicação.

4. **Desenvolvimento:** A complexidade no desenvolvimento das aplicações é uma das desvantagens da utilização da arquitetura de Gerenciador de Aplicação CGI, conforme [Lim97]. Entretanto, com a divisão das funcionalidades do gateway em módulos e objetos, permite ao desenvolvedores da aplicação voltar sua atenção somente para aqueles objetos aonde são implementados a lógica da aplicação, que neste caso são os objetos Web_Page, Aplicação e WebServer. Além disso, com a utilização do CORBA, fica transparente para os usuários detalhes de comunicação entre os objetos.
5. **Portabilidade:** A portabilidade do gateway é outro fator de destaque na implementação. Devido à utilização das tecnologias, CORBA, Java e CGI, que são padrões abertos e altamente portáveis, o gateway implementado é portável para qualquer plataforma, sistema operacional, e servidor Web. Além disso, o gateway foi implementado para ser utilizado em qualquer SGBD que suporte savepoints e que possua drivers para JDBC.

Capítulo 7

Estudo de Caso

7.1 Introdução

Neste capítulo, será analisado o desenvolvimento de uma aplicação Web Banco de Dados utilizando o gateway proposto no capítulo 6. Será enfatizado o processo de desenvolvimento da aplicação, com a definição das fases de desenvolvimento, ressaltando as funcionalidades dos Bancos de Dados no ambiente de integração.

O estudo de caso pretende discutir detalhes de características já abordadas e outras ainda não citadas de forma mais objetiva direcionadas ao gateway implementado. A efetiva implementação da aplicação a ser apresentada foi muito útil, pois serviu como um meio concreto para validar os resultados apresentados ao longo desta dissertação, mostrando através de um exemplo prático onde estes resultados podem ser aplicados.

Na seção 7.1 será apresentada a descrição da aplicação proposta para o estudo de caso, descrevendo a estrutura da base de dados e da lógica da aplicação.

Na seção 7.2 serão analisadas as fases de desenvolvimento da aplicação, definindo uma metodologia desenvolvimento, que envolve desde do processo de diagramação da aplicação até a fase de implementação.

7.2 Descrição da Aplicação

Para validar o modelo proposto e o gateway implementado foi decidido implementar uma aplicação que apresentasse os principais resultados desta dissertação e que não fosse

muito complexa a sua implementação.

Depois de analisar várias aplicações, ficou decido implementar uma aplicação clássica no estudo das transações, que é a aplicação de débito e crédito.

A aplicação de débito e crédito desenvolvida neste capítulo consiste em um sistema que permita a transferência de fundos entre contas correntes de clientes pertencentes a bancos diferentes, sendo que esta operação deve ser executada compreendendo várias páginas Web.

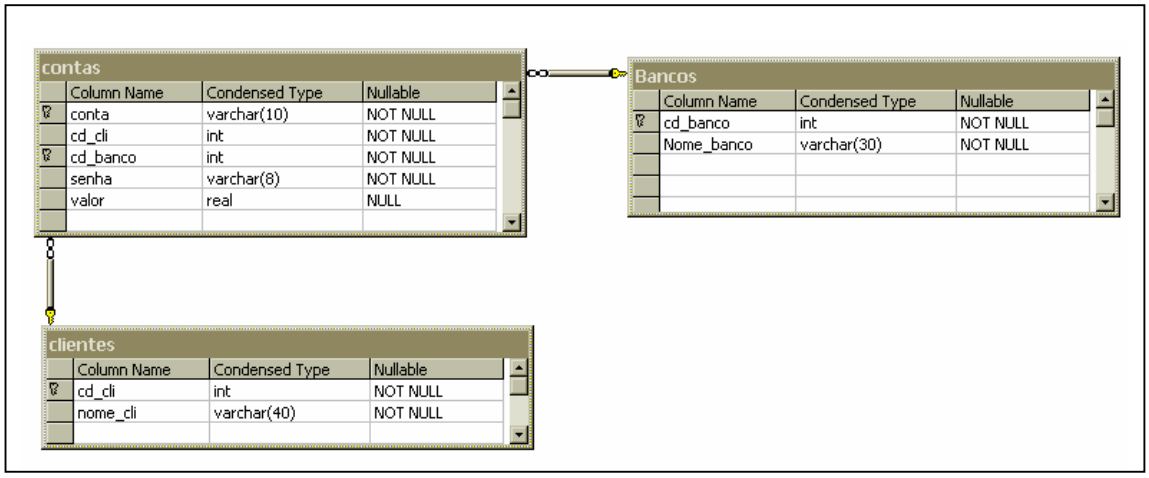

FIGURA 7.1 – ESQUEMA DA BASE DE DADOS DA APLICAÇÃO

A figura 7.1 ilustra o esquema do banco de dados especificado para esta aplicação. O esquema é constituído por três entidades:

1. A entidade *clientes* armazena informações sobre os correntistas dos bancos, sendo constituída por dois atributos. O atributo *cd_cli* é o código do cliente e chave primária da entidade. O atributo *nome_cli* representa o nome do cliente.
2. A entidade *bancos* armazena informações sobre os bancos cadastrados no sistema, sendo constituída por dois atributos. O atributo *cd_banco* é o código do banco e chave primária da entidade. O atributo *nome_banco* representa o nome do banco.
3. A entidade *contas* armazena informações das contas correntes dos correntistas dos bancos. A chave primária da entidade é formada pelo atributo *conta*, que identifica a conta corrente, e pelo atributo *cd_banco*, que identifica o banco,

chave estrangeira da entidade *bancos*. O atributo *cd_cli* é chave estrangeira da entidade *clientes*, utilizado para identificar qual o correntista daquela conta. O atributo *senha* é a senha de acesso da conta e o atributo *valor* é o saldo da conta.

A aplicação é constituída por seis páginas Web. A primeira página controla o acesso dos usuários ao sistema, conforme a figura A1 do anexo. Para o usuário ter acesso ao sistema é necessário que este identifique o banco, o identificador da conta e a senha de acesso. Uma vez autorizado a usar o sistema, o usuário pode realizar duas operações, conforme ilustra a figura A2 do anexo. Ele pode consultar o saldo da conta, pressionado o botão saldo, que gerará a página da figura A3 do anexo, ou poderá transferir um valor de sua conta corrente para outra, pressionado o botão destino. O valor a transferir é digitado na caixa de texto.

Selecionado o botão destino é gerada outra página, conforme ilustra a figura A4 do anexo. Nesta página o usuário seleciona o banco e conta para onde o dinheiro deverá ser transferido, depois pressiona o botão transferir, que criará uma nova página, como ilustra a figura A5 do anexo. Nesta página são apresentados as informações das contas origem e destino selecionadas nas páginas anteriores, sendo requerida a confirmação ou cancelamento da transferência. Se a transação for confirmada a transferência será validada e a página ilustrada na figura A6 do anexo aparecerá. Caso contrário, a transferência será cancelada e o sistema irá voltar para a primeira página da aplicação.

Deve-se ressaltar, que poderíamos ter implementado o sistema utilizando somente duas páginas, uma para controlar o acesso ao sistema e outras onde os usuários indicaria o valor a ser transferido e a conta destino. Entretanto, a aplicação foi quebrada em várias páginas para ilustra uma das principais vantagens do gateway, que é a capacidade de desfazer parte de uma transação, sem precisar desfaze-la toda, constituindo assim o caso onde a transação Web envolve mais de uma página, conforme citado na seção 4.3.2. Por exemplo, imagine a situação onde o usuário encontra-se na página da figura A5 e queira mudar a conta destino executando um back para a página anterior e selecionando uma nova conta. Se tivéssemos implementado esta aplicação utilizando um gateway baseando em transação planas esta ação seria impossível, pois a transação por completa teria que ser desfeita e usuários teria que começar tudo novamente. Entretanto, utilizado o gateway proposto, que implementa o modelo de transações definido no capitulo 6, esta operação é

possível, pois bastaria desfazer as operações da transação que foram executadas durante a criação daquela página. Como veremos nas seções seguintes, esta capacidade do gateway é totalmente transparente para o projetista da aplicação, para ele é como se tivesse usando um gateway baseado em transações planas.

7.3 Desenvolvimento da Aplicação

Definida a aplicação vamos nesta seção apresentar as fases de implementação envolvidas no desenvolvimento da aplicação, usando o gateway proposto no capítulo 7.

Antes de entramos em detalhes de implementação, iremos primeiro apresentar sucintamente um modelo diagramação para aplicações Web, que foi tomado como modelo lógico para o desenvolvimento do gateway.

7.3.1 Modelo de Diagramação da Aplicação

Apesar do gateway não apresentar uma interface que auxilie no desenvolvimento da aplicação, este foi desenvolvido tendo como base o trabalho de [RPF96].

Neste trabalho, a aplicação Web é vista como um autômato finito onde cada página Web é tratada como um estado, e a ordem de navegação das páginas como os eventos de transição de um estado para o outro, ou seja, a transição de um estado para o outro, vai depender de regras pré-definidas, baseadas na ordem de navegação das páginas Web.

Tendo como base esta idéia, podemos diagramar a aplicação proposta como um autômato finito, ilustrado na figura 7.2.1.

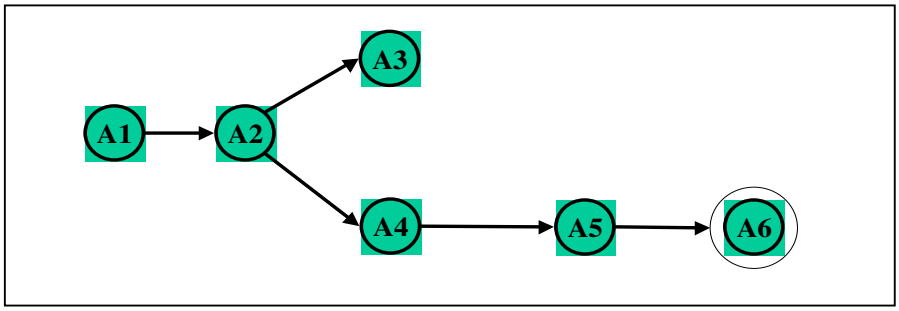

FIGURA 7.2.1 – DIAGRAMA DA APLICAÇÃO

Conforme ilustrado na figura, o autômato é constituído por seis estado, representados através de círculos, onde cada estado representa uma página Web. O círculo duplo representa o estado final válido, e as setas representam as transições válidas entre os estados.

7.3.2 Fases de Implementação

Definido o diagrama lógico da aplicação, iremos nesta seção definir o fases de implementação da aplicação, que farão o mapeamento do diagrama lógico para os objetos do gateway.

A implementação da aplicação pode ser dividida em três fases:

1. Definição do tipo de conexão com o SGBD
2. Definição dos objetos Web_Page .
3. Definição das regras de transição entre os objetos Web_Page

7.3.2.1 Definição do tipo de conexão com o SGBD

Como mencionado na seção 7.2.5, existem duas classes utilizadas para abrir conexão com o SGBD, a classe Contexto e a Web_Contexto. Portanto, esta fase consiste definir onde e quais classes utilizar.

Conforme apresentado no capítulo 7, a classe Contexto abre conexões com o SGBD do tipo "*autocommit*", ou seja, todas as consultas executadas pelo SGBD são tratadas como transações individuais, podendo ser compensável ou não. Portanto, esta classe deve ser utilizada quando a aplicação requisitar consultas que sejam executadas somente em uma interação.

Devido ao fato da classe Contexto tratar as consultas como transações individuais, então é possível que os objetos desta classe possam ser compartilhados entre os usuários da aplicação, permitindo assim que uma conexão com o SGBD possa ser compartilhada por vários usuários, não precisando abrir uma nova conexão para cada novo usuário da aplicação, que como já mencionado anteriormente, influência bastante no aumento do desempenho da aplicação.

No gateway, para que o objeto da classe Contexto seja compartilhado, basta defini-lo como sendo do tipo "*static*".

Já no caso da classe Web_Contexto, esta deve ser utilizada quando as operações de uma transação tiverem que ser executadas em mais de uma interação, ou seja, em mais de uma página Web. Neste tipo de conexão é o projetista da aplicação que controla o início (*begin_work*) e o fim da transação (*commit ou abort*). Ao contrário dos objetos da classe Conteto, que podem ser compartilhados entre vários usuários, os objetos da classe Web_Contexto devem ser criados uma para cada usuário da aplicação.

Com relação a aplicação proposta, foram criados um objeto da classe Contexto e um objeto da classe Web_Contexto.

O objeto da classe Contexto foi criado para executar consultas que envolvam apenas uma interação, como as consultas das páginas das figuras A1 e A3, onde são consultados os bancos cadastrados no sistema, para serem inseridos dentro de uma lista de seleção. Para otimizar e aumentar o desempenho, este objeto é do tipo "*static*", sendo portanto compartilhado por todos os usuários da aplicação.

Já o objeto da classe Web_Contexto foi criado para executar as consultas da transação de tranferência de fundos, que são executadas durante as páginas das figuras A2, A3, A4 e A5.

A figura 7.2.2.1 ilustra a definição desses objetos dentro da classe Aplicacao. Na linha 09 é definido o objeto default da classe Contexto e na linha seguinte o objeto Web_Contexto da classe Web_Contexto.

```
01.  public class AplicacaoImpl extends Gateway._AplicacaoImplBase {
02. //** Instancias defenidas pelo usuários
03. int cd_banco;
04. String nome_cli;
05. int cd_cli;
06. String conta;
07. String nome_banco;
08. float valor;
09. static sqlCORBA.Contexto Default;
10. sqlCORBA.Contexto Web_Contexto;
```

FIGURA 7.2.2.1 – FRAGMENTO DO CÓDIGO DA CLASSE APLICAÇÃO ONDE SÃO DEFINIDOS OS OBJETOS UTILIZADOS NA CONEXÃO COM O SGBD.

7.3.2.2 Definição dos objetos Web_Page

Definido os objetos utilizados para a conexão com o SGBD, vamos nesta seção, definir os objetos que irão gerar as páginas da aplicação, tendo base o diagrama da figura 7.2.1.

De modo geral, cada estado definido no diagrama do autômato finito irá corresponder a um objeto de uma classe filha da classe Web_Page, que receberá os dados do estado anterior, via o objeto Request, e gerará o novo estado, retornando-o através do objeto Response.

Para cada classe criada, o projetista da aplicação deve sobrecarregar o método execute da classe Web_Page, onde deve ser definido o código que irá gerar a página da aplicação.

No caso da aplicação proposta, foram criadas seis classes, uma para cada estado definido no diagrama. Para o primeiro estado foi criado a classe senha, que gera a página da figura A1. Para o segundo estado foi criado a classe origem, que gera a página da figura A2. Para o terceiro estado foi criado a classe saldo, que gera a página da figura A3. Para o quarto estado foi criada a classe destino, que gera a página da figura A4. Para o quinto estado foi criado a classe transferir, que gera a página da figura A5. Para o último estado foi criada a classe finalizar, que gera a página da figura A6. A figura 7.2.2.2 ilustra o código da definição da classe senha.

Na linha 2 da figura 7.2.2.2 é definida a classe senha que herda da classe Web_Page. Nas linhas de 3 a 8 são definidos os construtores da classe Web_Page. Da linha 9 até a 41 é definido código do método execute. Na linha 10 é instaciado o objeto cont que representa a conexão com o SGBD. Na linha 15 executa uma consulta que seleciona os bancos cadastrados no sistema. Da página 16 a 36 é gerado o código HTML que será retornado para navegador Web. Por fim na linha 39 é retornado o valor *true* indicando que todas as operações foram executadas com sucesso.

```
1.  package Gateway;
2.  public class Senha extends Gateway.Web_PageImpl {
3.  public Senha(java.lang.String name) {
4.  super(name);
5.  }
6.  public Web_PageImpl {
7.  super();
8.  }
9.  public boolean execute(Gateway.Aplicacao Aplic) {
10. String str;
11. sqlCORBA.Contexto cont =Aplic.Get_Contexto("default");
12. sqlCORBA.Consulta cons = cont.Create_Consulta("banco"+Aplic.Get_name());
13. String cursor = "rst"+Aplic.Get_name();
14. cons.setCursorName("rst"+Aplic.Get_name());
15. sqlCORBA.Resultado rst = cons.executeQuery("DECLARE "+cursor+" CURSOR READ_ONLY FOR
    select *
16. from bancos ORDER BY nome_banco open "+cursor+" FETCH NEXT FROM "+cursor,cursor);
17. Aplic.response().TipoConteudo("text/html");
18. Aplic.response().conteudo(" <!DOCTYPE HTML PUBLIC \"-//IETF//DTD HTML//EN\"><html> <head>
19. <meta http-equiv=\"Content-Type\"content=\"text/html; charset=iso-8859-1\"><meta name=
    "GENERATOR\"
20. content=\"Microsoft FrontPage 2.0\"><title>Untitled Normal Page</title></head><body
21. bgcolor=\"#FFFFFF\"><p align=\"center\"><font size=\"5\"><strong>Sistema de Transferência de
22. Cotas</strong></font></p><form action=\"/cgi-bin/gateway/cgi_teste.bat\"method=\"POST\"><p
23. align=\"center\">Login : <input type=\"text\" size=\"20\" name=\"T1\"></p><p align=\"center\">Senha:
    <input
24. type=\"password\" size=\"20\"name=\"T2\"></p><p align=\"center\"><input type=\"submit\"
    name=\"submit\"
25. value=\"origem\"></p></form><p align=\"center\"> </p><p> </p></body></html>");
26. while (rst.next()) {
27. str = str + "<OPTION value = \"" + rst.getIntStr("cd_banco") + "\">" + rst.getStringStr("nome_banco")+
28. "</OPTION>\n";
29. rst = cons.executeQuery("FETCH NEXT FROM "+ cursor,cursor);
30. }
31. str = str + "</select></p>    <p align=\"center\"><strong>Nº. da Conta: </strong><input ";
32. str = str + "type=\"text\" size=\"20\" name=\"conta\"></p><p align=\"center\">";
33. str = str + "<strong>Senha:</strong> <input type=\"text\" size=\"20\" name=\"senha\"></p>";
34. str = str + "<p align=\"center\"><input type=\"submit\" name=\"submit\" value=\"origem\"></p>";
35. str = str + "</form></body></html>";
36. Aplic.response().conteudo(str);
37. cons.execute("close "+ cursor+ " DEALLOCATE "+cursor);
38. cons.close();
39. return true;
40. }
41. }
```

FIGURA 7.2.2.2 – CÓDIGO DA CLASSE SENHA

7.3.2.3 Definição das regras de transição entre os objetos Web_Page

Definidas as classes que irão gerar as páginas da aplicação, falta apenas o último passo para a criação da aplicação, que consiste na definição de quais transições são válidas entre as páginas da aplicação.

A definição das regras de transição são especificadas no método Get_Web_Page da

classe WebServer, e são baseadas no nome da classe Web_Page que gerou a página corrente, e no nome da classe Web_Page que gerará a nova página.

O nome da classe Web_Page corrente é obtido através do valor da variável privada existente no objeto Aplicacao, denominada de estado_corrente, que é passado para o método Get_Web_Page através do atributo estado_corrente.

O nome da classe destino é especificado através do valor da variável submit do formulário HTML, passado para o método Get_Web_Page através do atributo novo_estado.

Utilizando estes atributos, basta o projetista da aplicação mapear cada seta do diagrama para uma regra, segundo o modelo da figura 7.2.2.3.a

```
if ((estado_corrente.compareTo("[nome da classe Web_Page que
gerou a página atual]")==0) && (novo_estado.compareTo("[nome
da classe Web_Page que gerará a nova página]")==0)){
      M_P = new Gateway.[nome da classe Web_Page]();
      return W_P;
    }
```

FIGURA 7.2.2.3.A – MODELO GERAL PARA DEFINIÇÃO DAS REGRAS DE TRANSIÇÃO

Como exemplo de definição destas regras, temos a figura 7.2.2.3.c que ilustra a definição da regra de transição do estado inicial, que corresponde a classe senha, para o segundo estado, que corresponde a classe origem, conforme o diagrama da figura 7.2.1.

```
if ((estado_corrente.compareTo("senha")==0) &&
(novo_estado.compareTo("origem")==0)){
      M_P = new Gateway.origem();
      return W_P;
    }
```

FIGURA 7.2.2.3.C – EXEMPLO DA DEFINIÇÃO DAS REGRAS DE TRANSIÇÃO

No caso da primeira página (primeiro estado), onde não há como determinar o estado anterior e o valor da variável submit, o projetista deve substituir o nome das classes pelo valor null, conforme a figura 7.2.2.3.b.

```
if ((estado_corrente.compareTo("null")==0) &&
(novo_estado.compareTo("null")==0)){
      M_P = new Gateway.[nome da classe Web_Page corresponde
ao estao inicial]();
      return W_P;
    }
```

FIGURA 7.2.2.3.B – REGRA DE TRANSIÇÃO PARA A PRIMEIRA PÁGINA

7.4 Conclusão

Ao longo deste capítulo foi descrito o processo de desenvolvimento de uma aplicação Web baseada em Banco de Dados, implementada utilizando o gateway proposto no capítulo 6, que além de servir como uma forma concreta de validação dos resultados desta dissertação, também serviu para ressaltar outra vantagem do gateway, a simplicidade no desenvolvimento da aplicação Web.

Este resultado é extremamente importante, pois como o gateway enquadra-se na arquitetura de Gerenciador de Aplicação CGI era esperado que este apresentasse um complexo processo de desenvolvimento, já que esta funcionalidade é tida como o ponto negativo desta arquitetura, conforme os estudos em [Lim97].

Numa avaliação mais objetiva, pode-se então afirmar que o gateway desenvolvido alcançou todos os resultados propostos na dissertação, apresentando soluções inovadoras, que tendem a ser um referencial bastante importante nas pesquisas relacionadas ao Ambiente de Integração Web e Banco de Dados.

Capítulo 8

Considerações Finais

8.1 Conclusão

O ambiente de Integração Web Banco de Dados é uma área de pesquisa que está em pleno desenvolvimento, onde a cada dia surgem novas tecnologias para simplificar a integração.

Apesar deste amplo dinamismo, observou-se durante alguns anos que as pesquisas nesta área restringiram-se basicamente ao problema de como mapear as operações *"stateless"* da Web para as operações *"statefull"* dos SGBDs. Entretanto, a medida que foram surgindo soluções cada vez mais eficientes, observou-se que o simples mapeamento não garantiria por completo a integração, era necessário que fossem solucionados outros problemas, como por exemplo, os relacionados com os aspectos transacionais.

Os problemas transacionais consistem basicamente em como suportar as propriedades ACID dos SGBDs, num ambiente hipertexto da Web, baseado em operações de longa duração.

É neste contexto que se apresenta à contribuição desta dissertação, através da especificação de um novo modelo de transações para o ambiente de integração, em substituição ao modelo de transações planas convencional que se mostrou inadequado por não suportar transações de longa duração e a capacidade de desfazer parte da transação.

O modelo de transações proposto fundamentou-se em tipos de (sub)transações, as páginas e as compensatórias.

As subtransações encadeadas possibilitaram a capacidade de desfazer parte da transação Web. Isto foi possível, porque a transação Web foi quebrada em um conjunto de subtransações encadeadas, onde cada uma delas envolve as operações executadas em cada página Web, que fazem parte da transação. Por exemplo, se a execução da transação Web envolver n-páginas, então haverá n-subtransações encadeadas, uma para cada página. Assim, caso seja necessário desfazer a execução de alguma página, basta desfazer a subtransação associada a ela.

Já as subtransações compensatórias foram utilizadas para dar suporte a necessidade de transações de longa duração, possibilitando ao usuário validar os resultados das subtransações encadeadas antes que a transação Web finalize.

Outro resultado bastante expressivo obtido nesta dissertação foi a simulação do modelo proposto utilizando somente os próprios recursos transacionais existente nos SGBDs tradicionais, não precisando assim de um SGBD especial para suportar o novo modelo, garantindo portanto, a restrição da não alteração das tecnologias.

Conclui-se, portanto, que esta dissertação cobriu todas as fases envolvidas num trabalho científico, pois além de apresentar a definição do problema e sua solução, também apresentou a formalização da solução, utilizando o modelo ACTA; implementou o gateway de integração; e desenvolveu um estudo de caso, utilizando o gateway, para validar os resultados desta dissertação.

8.2 Contribuição

As principais contribuições desta dissertação são:

- Levantamento dos principais problemas em aberto no ambiente de Integração Web Banco de Dados e um estudo detalhado das principais características transacionais do ambiente de Integração.
- Confirmação das limitações do modelo de transações planas para o suporte das características transacionais no ambiente de Integração, e da necessidade da definição de um novo modelo de transações para este ambiente. Deve-se ressaltar que as principais limitações das transações planas são a incapacidade de desfazer parte da transação e a falta de suporte a transações de longa duração.

- Definição de um novo de modelo de transações para o ambiente de integração, tendo como base as subtransações encadeadas, utilizadas para possibilitar desfazer parte da transação, e as subtransações compensatórias, utilizadas para suportar transações de longa duração.
- Simulação do modelo proposto utilizando somente os próprios recursos transacionais existentes nos SGBDs tradicionais. Para simular as transações encadeadas foi utilizado o mecanismo de *"save_point"* e para simular as transações compensatórias foram criadas tabelas adicionais.
- Implementação de um gateway de integração Web e Banco de Dados, onde foram apresentadas todas as fases envolvidas no projeto de um gateway, desde da definição da arquitetura e do diagrama de objetos, até questões relacionadas com a implementação. Como diferencial do gateway, pode-se citar: a simulação do modelo de transações Web proposto nesta dissertação; e a grande modularidade da gateway, obtida a partir da definição da arquitetura do gateway em cinco módulos e a pela definição dos objetos de cada módulo utilizando a linguagem IDL do CORBA.
- O desenvolvimento de um estudo de caso para validar de forma concreta o modelo de transações proposto e a implementação do gateway. Além disso, foi apresentada uma metodologia de desenvolvimento de aplicações Web Banco de Dados, que definiu um modo de modelar uma aplicação Web e mapear este modelo para os objetos do gateway.

8.3 Trabalhos futuros

Em virtude das pesquisas envolvendo o ambiente de integração Web e Banco de Dados serem relativamente novas, muitas questões técnicas, principalmente relacionadas com as características transacionais, devem ser estudadas para que esse ambiente seja mais confiável. Neste sentido, vários outros trabalhos podem ser desenvolvidos nesta área, sendo os trabalhos futuros decorrentes diretamente desta dissertação os seguintes:

- Realizar um levantamento bibliográfico dos trabalhos relacionados com a definição de modelos de transações para a Web e fazer um estudo comparativos entre estes modelos, analisando suas vantagens e desvantagens.
- Definir uma aplicação Web Banco de Dados que possua operações transacionais como, por exemplo, a aplicação de um shopping virtual, e desenvolvé-la utilizando o gateway proposto e um outro gateway de integração, que se enquadre na arquitetura de Servidor de Aplicação CGI, e fazer uma análise comparativa entre o desempenho destes gateways.
- Realizar um levantamento bibliográfico das soluções existente para a segurança na Web e para o acesso ao SGBD e propor uma solução para o problema da segurança no ambiente de integração e implementá-la no gateway proposto.
- Realizar um levantamento bibliográfico das metodologias de desenvolvimento de aplicações Web e das ferramentas existentes, e fazer um estudo comparativo entre as metodologias e as ferramentas, analisando suas vantagens e desvantagens.
- Aprofundar a especificação da metodologia de desenvolvimento de aplicações Web, proposta nesta dissertação, e implementá-la no gateway, através da criação de uma interface gráfica para auxiliar a geração da aplicação.
- Ao ínvês de simular o modelo de transações Web proposto nesta dissertação sobre um SGBD tradicional, poder-se-ia utilizar um Gerenciador de Transações que implementasse nativamente o modelo, para realizar um melhor analise das vantagens deste modelo sobre o modelo de transações planas.

Anexos

FIGURA A1 – TELA DE AUTORIZAÇÃO PARA USO DA APLICAÇÃO

FIGURA A2 – TELA UTILIZADA PARA CONSULTAR O SALDO OU INCIAR A TRANSFERÊNCIA

FIGURA A3 – TELA COM O SALDO DA CONTA DO CLIENTE

FIGURA A4 – TELA PARA A DEFINIÇÃO DA CONTA DESTINO

FIGURA A5 – TELA PARA A CONFIRMAÇÃO DA TRANSFERÊNCIA

FIGURA A6 – TELA DE FINALIZAÇÃO DA TRANSFERÊNCIA

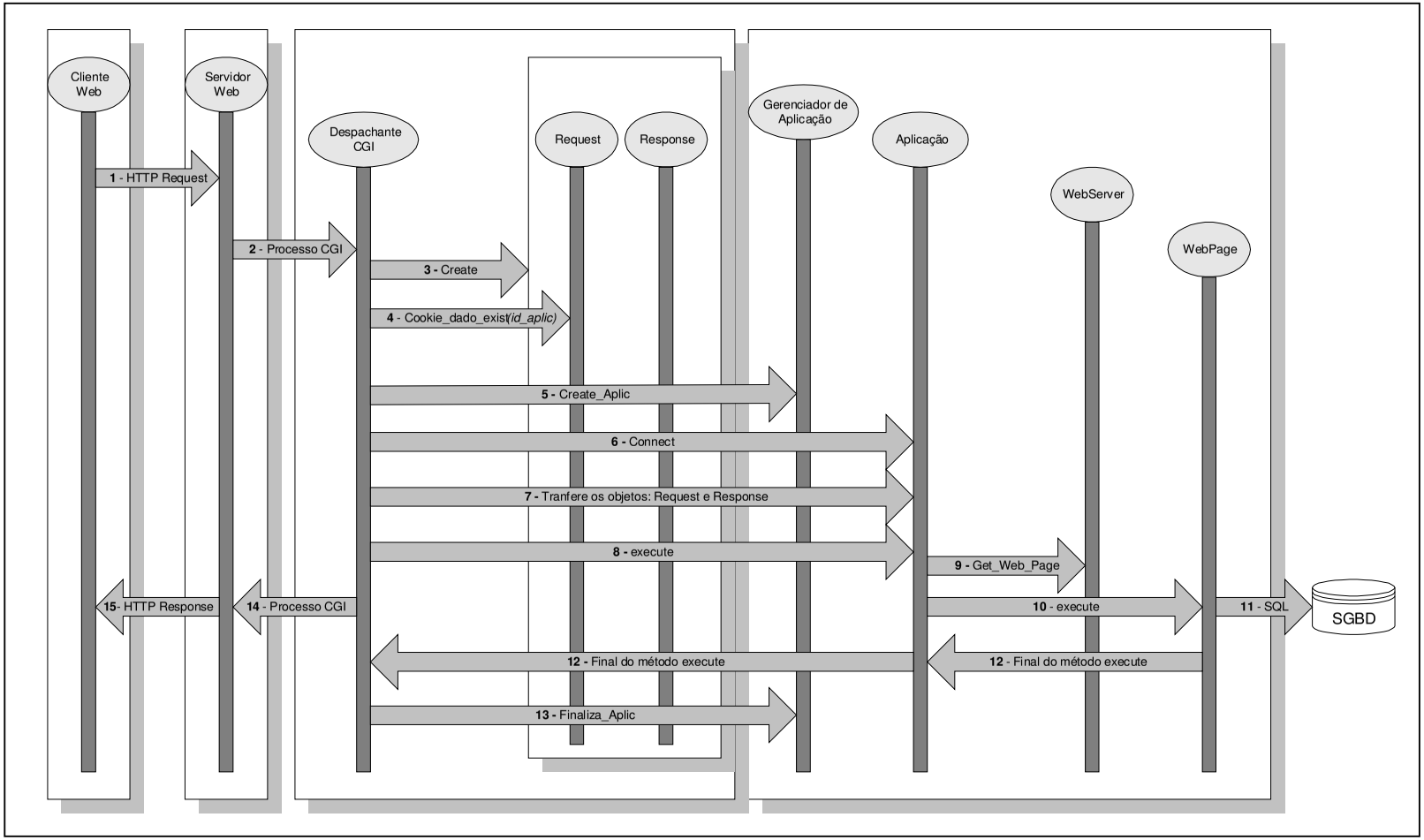

FIGURA A7 – DIAGRAMA DE SEQUENCI A DO FUNCIONAMENTO DO GATEWAY

Referências Bibliográficas

[AMJ95] Avi Silberschatz, Mike Stonebraker, Jeff Ullman: "Database Research: Achievements and Opportunities Into the 21st Century". Report of na NFS Workshop on the Future of Database System Research, maio 26-27, 1995.

[AV97] S. Abiteboul e V. Vianu: "Queries and Computation on the Web", Procs. Intl. Conference on Database Theory (ICDT), 1997 , pp. 262-275. http://www.db.stanford.edu/pub/papers/icdt97.www.ps.

[BC95] T. Berners-Lee, R. Cailliau: "Hypetext Markup Language Specification - 2.0". Internet RFC 1866, Nov. 1995.

[BCD95] G. Bossert, S. Cooper, W. Drummond, "Requirements for HyperText Protocol Security", Internet Draft, Rutgers University Network Services, Março, 1995

[BCL+94] T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen , A. Secret: "The world-wide web". Communications of the ACM, 37(8):76-82, 1994

[Ber94] T. Berners-Lee. "Uniform Resource Locators". Internet RFC 1738, Dez. 20, 1994.

[BFN96] T. Berners-Lee, R. T. Fielding, H. F. Nielsen. "Hypertext transfer protocol - http/1.0". Technical report, Internet RFC 1945, Maio 1996.

[Bil98] D. Billard: "Transactional Services for the Internet", Computer Science Departament, University of Geneva, 1998.

[BJM+94] E. Bina, V. Jones, R. McCool e M. Winslett. "Secure Acces to Data Over the Internet". Proc. Of the third ACM/IEEE Intl. Conf. On Parallel and Distributed Information Systems, Austin, Texas, 1994.

[Chr91] P. K. Chrysanthis. "ACTA, A framework for modeling and reasoning about extended transactions". Tese de Doutorado, submetido ao Departamento de Computação e Ciências da Informática da Graduate School of the University of Massachusets.

[CSJ+98] C. S. Peng, S. K. Chen, J. Y. Chung, A Roy-Chowdhury, V. Srinivasan: "Accessing existing business data from the World Wide Web". IBM. 1998.

[DI*95] A.Díaz, T. Isakowitz, V. Maiorana, G. Gilabert, RMC: A Tool do Design WWW Aplications, in Proc. Fourth Intrnational World Wide Web Conference, Dec. 11-14, 1995, Boston, Massachusetts, USA.

[Dua96] N. N. Duan: "Distributed Database Acess in a Corporate Environment Using Java". Fifth Intl. World Wide Web Conference, Paris, france, 1996. http://www5.conf.inria.fr/fich_html/paper-sessions.html

[EKR97] G. Ehmayer, D, Kappel e S. Reich. “Connecting Databases to the Web: A taxonomy of Gateways”. Departament of Information Systems, Johannes Kepler University, Austria, 1997

[ELLR90] A. Elmagarmid, Y. Leu, e S. Ostermann. Efects of autonomy on global concurrency control in interdase. In Proceedings of International Conference Vrey Large Data Bases, pag. 347-355, Amsterdam, The Netherlands, agosto 1989.

[Elm90] A. K. Elmagarmid. "Database Transaction Models for Advanced Applications". Morgan Kaufmann Publishers, San Mateo, California, 1990.

[EN94] R. Elmasri, S. B. Navathe: “Fundamentals of database Systems”, Second Edition. The Benjamin/Cummings Publishing Company, Inc. 1994.

[EVT88] F. Eliassen, J. Veijalainen, e H. Tirri. “Aspects of transaction modelling for interoperable information systems”. I Interim Report of the COST 11ter Priject, pages 39-55, 1988

[FP97] P. Fraternalli, P. Paolini, “A conceptual model and a tool environment for developing more scalable, dynamic, and customizable web applications, Politecnico di Milano, Dipartimento di Elettronica e Informazione, Technical Report 1997-X, August 1997. http://www.ing.unico.it/autoweb

[Fra95] M. Frank: “Database and the Internet”, DBMS On-Line Magazine, December, 1995. Http://www.dbmsmag.com/f19512.html

[Fra98] P. Fraternali. "Web Development Tools: A Survey". Proc. WWW7, Brisbane, Australia, April 1998.

[GGS97] Gerald Ehmayer, Gerti kappel, Siegfried Reich: "Connecting Databases to the Web: A Taxonomy of Gateways". Departament of Information Systems, Johannes Kepler University of Linz, Austria. Dexa 1997. Pg. 1-15.

[GMS87] H. Garcia-Molina e K. Salem. Sagas. In Proceedings of the ACM Conference on Management of Data, pages 249-259, May 1987.

[GR93] J. Gray e A. Reuter. “Transaction Processing: Concepts and Techniques”. Morgan Kaufmann Publishers, San Mateo, California, 1993.

[Gra81] J. Gray. The transaction concept: Virtues and limitations. In Proceedings of the International Conference on Very Large Data Bases, paes 144-154, Cannes, France, September 1981.

[Gun96] S. Gundavaram : “Using Cooking with CGI” World Wide Web Journal, Vol. 1(4), 1996. Http://www.w3.org/pub/WWW/Journal/4/s3.shishir.html.

[Hal??] P. M. Hallam-Baker: “Shen: A security Scheme for the World Wide Web”, CERN Programming Techniques Group. http://www.w3.org/hypertext/WWWShen/ref/shen.html

[Hal00] M. Hall. “Core Servlets and JavaServer Pages (JSP)”, Enterprise Edition Series. 2000.

[Hic95] K. E. B. Hickman: “The SSL Protocol”, Internet Draft, Netscape Communications Corporation, fevereiro de 1995. http://www.netscape.com/info/SSL.http

[HJH+95] J. L. Hosterler, J. Franks, P. Hallam-Baker, A. Luotonen, E. W. Sink, L. C. Stewart: “A proposed extension to HTTP: Disgest Access Authentication”, março de 1995. ftp://nic.nordu.net/internet-drafts/draft-ietf-http-digest-aa-01.txt

[HM96] S. P. Hadjefhymiades e D. I. Martakos: “A generic framework for the deployment of structured databases on the World Wide Web”, Fifth Intl. World Wide Web Conference, Paris, France, 1996. http://www5conf.inria.fr/fich_html/paper-sessions.html

[HM97] S. P. Hadjefhymiades e D. I. Martakos: “Improving the performance of CGI compliant database gateways”, Sixth Intl. World Wide Web Conference, Santa Clara, California, EUA, 1997. Http://www6.ntlabs.com/papers/PAPER44/PAPER44.html

[Int96] Internet Systems: “Internet tools product chart”, DBMS On-line Magazine, 1996. http://www.dbmsmag.com/9610i.html

[JGS96] Jack Jingshuang Yang, gail E. Kaiser , Stephen E. Dossick: "Na External Transaction Service for WWW". Columbia University. CUCS-047-96. http://www.psl.cs.columbia/papers/CUCS-047-96.html

[Kes95] M. Kesseler, A Schema-Based Approach to HTML Authoring, in Proc.Fourth World Wide Web Conference, Dec 11-14, 1995, Boston, Massachusetts, USA.

[Kim96] Pyung – Chul Kim. “A Taxonomy on the Architecture of Database Gateways for the Web”, Technical report, Database Laboratory Dept. of Information Communications Enginnering, Chungnam National University, 1996. http://grigg.chungnam.ac.kr/projects/UniWeb/documents/taxonomy/text.html

[KPE91] E. Kuehn, F. Puntigam, e Elmagarmid. Transaction specification in multidatabase ystems based on parallel logic programming. In Proccedings of the First International Workshop on Interoperability in Multidatabse Systems, IMS91, Japão, Abril 1991.

[KS95] D. Knopnicki e O. Shmueli: “W3QS: A Query System for the World Wide Web”, proc. Intl. Conf. on VLDB, Zurich, Suiça, 1995, pp. 54-65.

[KS97] H. F. Korth e A. Silberchatz: “Database System Concepts”, Segunda edição, McGraw-Hill, 1997.

[Lat94] L. Latham, "Client/Server Computing: Strategic Directions, Tactical Solutions," InSide Gartner Group This Week, Vol. X, No. 20, Gartner Group, May 18, 1994, pp. 1-5.

[LEK97] J. Lyon, K. Evans e J. Klein: “Transaction Internet Protocol”, Internet Draft, Microsoft Corporation, 1997. http://204.203.124.10/pdc/docs/TIP.txt

[Lim97] I. N. de Lima: “O Ambiente Web banco de dados: Funcionalidades e Arquiteturas de Integração”. Tese de Mestrado, Departamento de Informática da PUC-Rio, 1997.

[Loc88] F. Lochovsky. "Role-based Security in Data Base Management Systems". Database Security: Status and Prospcts, C. E. Landwehr, Editor, 1988.

[LSC+97] M. C. Little, S. K. Shivastava, S. J. Caughey, e D. B. Ingham: “Constructing Reliable Web Application Using Atomic Actions”, Sixth Intl. World Wide Web Conference, Santa Clara, California, EUA, 1997. http://www6.ntlabs/com/HyperNews/get/PAPER12.html.

[Luo93] A. Luotonen: “Basic Protection Scheme for the WWW”, Dezembro de 1993. http://www.w3.org/hypertext/WWW/AccessAuthorization/Basic.html

[Luo93b] A. Luotonen: “Public Key Protection Scheme for the WWW”, dezembro de 1993. http://www.w3.org/hypertext/WWW/AccessAuthorization/PubKey.html

[MAA96] C. D. Murta, J. M. de Almeida e V.A.F. Almeida. “Análise de Desempenho de um Servidor WWW” Anais do XXII Seminário Integrado de Software e Hardware, IX CTIC, Recife, PE, 1996, pp. 391-402.

[MAD97] Masum Z. Hasan, Alberto O. Mendelzon, Dimitra Vista: "Applying Database Visualization to the World Wide Web". Computer Systems Research Institute. University of Toronto. Toronto, Canada.

[MMM96] A. Mendelzon, G. Mihaila e T. Milo: Quering the Wolrd Wide Web”, Porc. Intl. Conf. on Parallel and Distributed Information Systems (PDIS), 1996, pp. 80-91. ftp://bd.toronto.edu/pub/papers/pdis96.ps.gz

[Mos81] J.E. Moss. Nested Transaction: Na approach to Reliable Distributed Computing. PhD thesis, Department of Electrial Engineering and Computer Science, MIT, 1981.

[MRKN91] P. Muth, T.C. Rakow, W. Klas, e E. J. Neuhold: “A trasaction model for na open publication environment”. IEEE Data Enginerring Bulletin, 14(1), Março de 1991.

[NCSA??] National Center for Supercomputing Applications: “Using PGP/PEM auth”. http://hoohoo.ncsa.uiuc.edu/docs/PEMPGP.html

[Net97] Netscape Communications Corporation: “Persistent Client State HTTP Cookies – Preliminary Specification”, 1997. http://home.netscape.com/newsref/std/cookie_spec.html

[Net97g] Netscape Communications Corporation: “JavaScript: Guide Authoring”, 1997. http://home.netscape.com/eng/mozilla/gold/handbook/javascript/index.html.

[Nex97] NeXT Software, Inc: “WebObjects Dynamic Applications for the Web”, 1997. http://www.next.com/WebObjects/WebObjectsDynamic.html.

[NF96] D. Newman, J. Fowler: “Web Servers: Digging Into Corporate Data”, data communications on the Web, 1996. Http://www.data.com/lab_tests/digging.html

[NZ84] M. Nodine e S. Zdonik. Cooperative transaction hierarchies: A transaction model to support design applications. In Proceedings of the International Conference on Very Large Data Base, pages 83-94, 1984.

[NZ90]. Nodine, M.H. e Zdonik, S.B. “Cooperative transction hierarchies: a transaction model to support design applications”. Proceedings of the Sixteenth Intl. Conference on ery Large Databases, Brisbane, Autralia, 1990

[ORA97] Oracle Corporation: “The Transacional Web Server for Business-Critical Applications”, White Paper, 1997.
http://www.olab.com/products/was/html/ows3_whitepaper.html

[Orf98] R, Orfali: “Client/Server Programming with Java and CORBA.” John Wiley & Sons, Inc., 1998.

[OV91] M. T. Özsu, P. Valduriez: “Principles of distributed database systems”, Prentice-Hall, 1991. Pp. 257-275.

[Per94] L. Perrochon: “Translation servers: Gateways between *stateless* and stateful information systems”, Insitut fur Informations systeme, ETH Zurich, Technical report, 1994, PA-nsc94.http://www.inf.ethz.ch/departament/IS/ea/publications/nsc94.html.

[Per95] L. Perrochon: “W3 ‘Middleware’: Notions and Concepts”, The Fourth Intl. Conf. On the World Wide Web (Workshop on Web access to legacy data), Boston, USA, 1995.

[PF98] P. Paolini, P. Fraternali. "A Conceptual Model and a Tool Environment for Developing More Scalable, Dynamic, and Customizable Web Applications", to apper in Proc EDBT98 Conference, Valencia, Spain. 1998. http://www.ing.unico.it/autoweb

[PHMS96] G. Port, C. Heath, P. Merrick e T. Segall: "Requirements for Taking Applications Beyond the Enterprise", World Wide Web Journal, Vol 1(1), 1996.
http://www.w3.org/pub/WWW/Journal/1/port.319/paper/319.html

[PKH88] C. Pu, G. Kaiser, e N. Hutchinson. Split-transaction for open-ended activities. In Proccedings of the 14th VLDB Conference.

[Pla96] S. W .Plain: “Web Database Tools”, PC Magazine, Vol 15 (15), September, 1996. Pp. 205-256.

[SDB+91] P.L. van der Spiegel, J.T.W. Diressen, P.D. Bruza, e Th.P. van der Weide. “A transaction Model for Hypertext”In D. Karagiannis, editor, Proceedings of the Data base and expert system Appications Conferrence (DEXA 91), pages 281-286, Berlin, Germany, 1991.

[RA97] Raghu Ramakrishnan, Avi Silberschatz: "Scalable Integration of Databse Collections on the Web". CS Dept. University of Wisconsin-Madison, and Lucent Bell Labs. 1997.

[Rag95] D. Raggett: “Mediated Digest Authentication”, Internet Draft, HP Labs, março de 1995. ftp://nic.nordu.net/internet-drafts/draft-ietf-http-mda-00.txt

[RC94] K. Ramamritham, P. K. Chrysanthis: In Search of Acceptability Criteria: Database Consistency Requeriments and Transaction Correctness Properties, Distributed Object Management, Morgan Kaufmann Publishers, San Mateo, California, 1994.

[Reu89] K. Reuter. Contract: A means for extending control beyond transaction boundaries. In Proceedings of the 2nd International Workshop on High Performance Transaction Systems, Setembro 1989.

[Row97] J. Rowe: “Accessing a database Server via World Wide Web”, Computer Sciences Corporation Computational Analysis and programing Suport Services (CAPSS), 1997. http://cscsunl.larc.nasa.gov/~beowulf/db/all_products.html

[RS91] Makek Rusinkiewicz e Amit Sheth. Polytransaction for managing interdependent data. IEEE Data Engineering Bulletin, 14(1), março 1991.

[RS94] E. Rescorla, A. Schiffman, “The Secure HyperText Transfer Protocol”, dezembro de 1994. ftp://nic.nordu.net/internet-drafts/draft-rescorla-shttp-00.txt

[RW91] A. Reuter e H. Wachter. The contract model. IEEE Data Engineering Bulletin, 14(1), Março 1991.

[Sch94a] R. Schulte, "Two-Tier vs. Three-Tier Trade-Offs," SMS:K-401-1564, SMS Research Note, Gartner Group, Dec. 1994.

[Sch94b] R. Schulte, "Three-Tier Software Architecture in Perspective," SMS:K-401-1565, SMS Research Note, Gartner Group, Dec. 1994.

[Sha96] R. Shah: "Integrating Databases with Java via JDBC", Java World On-line Magazine, 1996.

[Sim95] J. P. F. Simão. ‘Segurança no WWW – Panorâmica do estado da Arte’. Departamento de Informática. Faculdade de Ciências e tecnologia, Universidade Nova de Lisboa. Junha de 1995.

[SR95] D. Schawabe, G. Rossi, The Object-Oriented Hypermedia Desin Model, Communications

[Tel97] J. M. Telford: "Allaire Cold Fusion 2.0 professional". DBMS On-line magazine, Fevereiro, 1997.

[US91] R. Unland e G. Schlageter. A flexible and adaptable tool kit approach for transaction management in non-standard database systems. IEEE Data Engineering Bulletin, Março 1991.

[W3C97c] W3C Consortium "CGI-Common Gateway Interface",1997. http://www.w3.org/pub/WWW/CGI/.

[W3C97f] W3C Consortium "Overview of SGML and XML Resources",197. http://www.w3.org/MarkUp/SGML/.

[Wei91] G. Weikum. Principles and realization strategies of multilevel transaction management. ACM Transaction on Database Systems, 16(1):132-180, Março 1991.

[WS91] G. Weikum e H. Schek. Multi-level transaction and open nested transactions. Data Engineering Bulletin, março 1991.

[YK96] J. J. Yang e G. E kaiser: "Na Architecture for integrating OODBs with WWW", Fifth Intl. World Wide Web Conference, Paris, France, 1996. http://www5conf.inria.fr/fich_html/papers/P31/Overview.html

]
]