

**UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA
COMPUTAÇÃO**

ISABELA STANIZIO LESSA DE CARVALHO

**UM ESTUDO SOBRE MODELOS DE
TRANSAÇÕES PARA A INTERNET**

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de mestre em Ciência da Computação.

Orientador: Prof^o. Murilo Camargo, Dr.

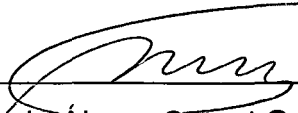
Computação Distribuída

Florianópolis, dezembro, 2000.

UM ESTUDO SOBRE MODELOS DE TRANSAÇÕES PARA A INTERNET

ISABELA STANIZIO LESSA DE CARVALHO

Essa dissertação foi julgada adequada para a obtenção do título de Mestre em Ciências da Computação Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo programa de Pós-Graduação em Ciência da Computação.



Prof.º Fernando Alvaro Ostuni Gauthier, Dr.
Coordenador do Curso

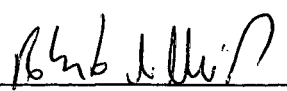
Banca Examinadora:



Prof.º Murilo Silva de Camargo, Dr.
Orientador



Prof.º Vitorio Bruno Mazzola, Dr.
Membro



Prof.º Roberto Willrich, Dr.
Membro

“...quando Colombo encontrou o seu caminho, não foi apenas pelos ventos e as águas do Atlântico que ele navegou, mas pelos ventos e as águas das aspirações de sua alma...”

Emanuel Swedenberg

Dedico especialmente este trabalho aos meus mestres da vida e do amor, meu pai, Pedro Rubens e minha mãe, Isa, que me ensinaram a ter perseverança e simplesmente nunca desistir. Ao meu noivo, Luis Alberto, que suportou longos momentos de ausência sempre me apoiando, e à minha irmã Daniela, meu maior tesouro.

AGRADECIMENTOS

Ao Professor Murilo Camargo, meu orientador, que no decisivo momento me estimulou a prosseguir e a vencer com alegria mais esta etapa, orientando e estimulando sempre, visando assim, meu amadurecimento como aluna e pesquisadora.

Aos professores que constituem a Banca Examinadora, o Professor Roberto Willrich e o Professor Vitorio Bruno Mazzola, por colaborarem diretamente para minha formação acadêmica no mestrado.

Ao meu querido professor do Curso de Graduação, Professor José Carlos Tavares da Silva, por sua serena capacidade de educar.

Ao meu querido amigo Nelson e a minha querida amiga Arlete, que foram os meus grandes incentivadores. Com certeza se não fossem as suas palavras de força e coragem, não teria chegado ao fim. Por me estimularem a prosseguir quando fiquei prisioneira da incerteza.

À Vera e à Valdete, que sempre com sua simpatia e prestatividade me ajudaram em qualquer assunto, quer seja assuntos de secretaria, quer seja com palavras de carinho e incentivo.

SUMÁRIO

Lista de Figuras

Lista de Abreviaturas

Resumo

Abstract

1 INTRODUÇÃO	1
1.1 Motivação	1
1.2 Objetivos	2
1.3 Principais limitações	3
1.4 Contribuições	5
1.5 Organização	6
2 TRANSAÇÕES GERAIS	9
2.1 Conceito de transações	9
2.2 Transações atômicas	10
2.3 Propriedades ACID	11
2.4 Recuperação	12
2.4.1 Recuperação de transações	14
2.4.2 Commit de duas fases	16
2.5 Modelos de transações avançadas	18
2.6 Conclusão	20

3 TRANSAÇÕES NA WEB – COLOCAÇÃO GERAL DO PROBLEMA	22
3.1 Os principais personagens nas transações <i>Internet</i>	24
3.1.1 Gerenciamento da execução de transações	25
3.1.2 Modelos de transação	26
3.2 Integrando a Web e Banco de Dados	27
3.2.1 Problemas de integração Web e Banco de Dados	30
3.2.2 Sistemas legados	31
3.2.3 Problemas transacionais	31
3.2.4 Segurança e acesso ao Banco de Dados	32
3.2.5 Controle de restrição de integridade	33
3.2.6 Desempenho	34
3.2.7 Desenvolvimento e portabilidade	35
3.2.8 Linguagens de Programação	36
3.3 Conclusão	37
4 FUNCIONALIDADES WEB SGBD	38
4.1 Sistemas <i>stateless</i> versus sistemas <i>stateful</i>	39
4.1.1 Gerenciamento da estado da aplicação	41
4.1.1.1 <i>Campos ocultos nas páginas Web</i>	43
4.1.1.2 <i>Cookies</i>	44
4.1.2 Desempenho do SGBD	45
4.2 Escopo de uma transação na Web	48
4.2.1 Uma página como unidade atômica	49
4.2.2 Várias páginas como unidade atômica	49
4.2.3 Páginas que não fazem parte de uma transação	50
4.2.4 Páginas com dupla funcionalidade	51
4.2.5 Recuperação	52
4.2.6 Consistência	58
4.2.7 Controle de concorrência e isolamento	60
4.3 Conclusão	62

5 ARQUITETURA	64
5.1 Taxonomia das arquiteturas de transação Web SGBD	65
5.1.1 Arquitetura de transações do lado servidor Web.....	67
5.1.1.1 <i>Interface CGI</i>	69
5.1.1.2 <i>Servidor Web estendido</i>	78
5.1.2 Arquitetura de transações do lado cliente Web.....	85
5.1.2.1 <i>Aplicação externa</i>	85
5.1.2.2 <i>Cliente Web estendido</i>	88
5.1.3 Arquitetura de transações do lado do cliente e servidor <i>Web</i>	91
5.1.3.1 <i>Servidor de transação</i>	91
5.2 Comparação de arquiteturas	97
5.3 Conclusão	101
6 TECNOLOGIAS ENVOLVIDAS	102
6.1 <i>Corba Transaction Service</i>	102
6.2 <i>Object Transaction Service (OTS)</i>	103
6.3 <i>Transaction Internet Protocol (TIP)</i>	106
6.4 <i>X/Open – DTP – Distributed Transaction Processing</i>	106
6.6 Conclusão	107
7 ESTUDOS DE CASO: TRANSAÇÕES NA WEB	108
7.1 Descrição da aplicação	109
7.2 Desenvolvimento	112
7.3 Segurança	113
7.4 Desempenho da aplicação	114
7.5 Desempenho do SGBD	116
7.6 Gerenciamento do estado da aplicação	116
7.7 Controle de concorrência	117
7.8 Consistência	117
7.9 Recuperação	119
7.10 Restrições de integridade	120

7.11 Portabilidade	121
7.12 Conclusão	122
8 BOLERO UMA FERRAMENTA DE MERCADO	123
8.1 A fábrica de aplicações para o comércio eletrônico	124
8.1.1 O Bolero e o Java	126
8.1.2 Integração com a <i>Web</i>	127
8.1.3 Aplicações alvo	128
8.1.4 Modelos de projeto	129
8.2 Módulos de Bolero	130
8.2.1 O Bolero <i>Application Server</i>	130
8.2.2 O Bolero <i>Component Studio</i>	132
8.2.3 O Bolero <i>GUI Builder</i>	134
8.2.4 O Bolero <i>Report Writer</i>	134
8.2.5 Repositório compartilhado	135
8.3 Mapeamento objeto-relacional	135
8.4 Linguagem OQL (<i>Object Query Language</i>)	136
8.5 Transações no Bolero	136
8.5.1 Controle de transações	137
8.5.2 Transação longa	139
8.6 Conclusão	141
9 CONCLUSÃO FINAL	143
9.1 Contribuições da dissertação	145
8.5 Trabalhos Futuros	146
REFERÊNCIAS BIBLIOGRÁFICAS	149

LISTA DE FIGURAS

Figura 3.1 : Arquitetura do ambiente de integração Web e BD	43
Figura 4.1 : Descontinuidade da sessão de usuário	54
Figura 4.2 : Gateway que não implementa sessão contínua	60
Figura 4.3 : Processo de execução de uma consulta via cursor	61
Figura 4.4 : Transação composta por uma página Web	63
Figura 4.5 : Transação composta por várias páginas Web	64
Figura 4.6 : Transação composta por páginas transacionais e por páginas Normais	65
Figura 4.7 : Páginas com dupla função	66
Figura 4.8 : Pontos de falhas no ambiente Web Banco de Dados	67
Figura 5.1 : Taxonomia das arquiteturas Web BD	80
Figura 5.2 : Arquitetura gateways que executam no lado servidor Web	82
Figura 5.3 : Arquitetura Programas Executáveis CG	84
Figura 5.4 : Arquitetura Gerenciador de Aplicação CGI	89
Figura 5.5 : Arquitetura Aplicação API	92
Figura 5.6 : Arquitetura Aplicação SSI	95
Figura 5.7 : Arquitetura Acesso Direto ao Banco de Dados	97
Figura 5.8 : Arquitetura Aplicação Externa	100
Figura 5.9 : Arquitetura Cliente Web Estendido	102
Figura 5.10 : Arquitetura Servidor de Transação	106
Figura 5.11 : Quadro comparativo das Arquiteturas	112
Figura 7.1 : Arquitetura da Aplicação	124

Figura 7.2 : Fluxo de dados dos programas da aplicação	125
Figura 7.3 : Ativação dos programas CGI da aplicação	129
Figura 8.1 : Estrutura do Bolero	171
Figura 8.2 : O Ambiente Bolero X Java	172
Figura 8.3 : Estrutura do Bolero Application Server	177
Figura 8.4 : Interface do Bolero Component Studio	180
Figura 8.5 : Objetos persistentes no Bolero	185
Figura 8.6 : Bolero: Transação longa de processamento de pedidos	186
Figura 8.7 : Editor de transações longas	187

LISTA DE SIGLAS

ACID - Atomicidade, Consistência, Isolamento, Durabilidade
API - *Application Programming Interface*
ASCII - *American Standard Code for Information Interchange*
CGI - *Common Gateway Interface*
CORBA - *Common Object Request Broker Architecture*
CPU - *Central Processor Unit*
DCOM - *Distributed Component Object Model*
DDM - *Data Driven Mode*
DLL - *Dynamic Link Library*
GIOP - *General Inter-ORB*
HTML - *Hyper Text Markup Language*
HTTP - *Hyper Text Transport Protocol*
IIOP - *Internet Inter-ORB*
JDBC - *Java DataBase Connectivity*
JDK - *Java Developer Kit*
JIT - *Just-In-time*
JTS - *Java Transaction Service*
JVM - *Java Virtual Machine*
LAN - *Local Area Network*
ODMG - *Object Data Management Group*
OLE - *Object Linking and Embedding*
OMG - *Object Management Group*
OO - *Orientação à Objetos*
OQL - *Object Query Language*
ORB - *Object Requested Broker*
OTS - *Object Transaction Service*
PC - *Personal Computer*

PHP - *Personal Home Page*

RMI - *Remote Method Invocation*

RPC - *Remote Procedure Call*

RPM - *Rapid Prototyping Mode*

SGBD - *Sistemas de Gerenciamento de Banco de Dados*

SQL - *Structured Query Language*

SSI - *Server Side Includes*

TIP - *Transacton Internet Protocol*

TP - *Transaction Processing Monitors*

URL - *Uniform Resource Locator*

WAN - *Wide Area Network*

WWW - *World Wide Web*

XML - *eXtensible Markup Language*

RESUMO

Este trabalho apresenta um estudo detalhado sobre modelos transacionais especificamente para o ambiente Web Banco de Dados, no que diz respeito a aplicações *Intranet/Internet*. São expostos os requisitos necessários para o desenvolvimento de transações que envolvam a Web e Bancos de Dados, além de reunir as funcionalidades das duas tecnologias. São abordados aspectos fundamentalmente transacionais do ambiente de integração, com ênfase nas propriedades ACID. É proposta uma classificação das arquiteturas para transações que envolvam a *Web* e Bancos de Dados, enfatizando sempre o comportamento das funcionalidades de Sistemas Gerenciadores de Banco de Dados sob cada arquitetura, como gerenciamento dessas transações, controle de restrições de integridade, segurança, acesso, desempenho, desenvolvimento e portabilidade. São abordadas as tecnologias envolvidas em transações na Web, como CORBA, TIP (*Transaction Internet Protocol*), X/OPEN, OTS (*Object Transactions Service*). Em seguida será apresentado um estudo de caso baseado em uma das principais arquiteturas de transações Web Banco de Dados, denominada *Programas Executáveis CGI*, e será apresentada também uma ferramenta de mercado, o Bolero, que se baseia em outra importante arquitetura de transação, denominada *Servidor de Transações*, os dois últimos exemplos são empregados para o desenvolvimento de aplicações *Internet/Intranet*.

Palavras Chave: Modelos de Transação, Banco de Dados, Web, *Internet*, *Intranet*

ABSTRACT

This work describes a study detailed on Transactions Models specifically developed for the Web Data base environment, on what regards to Intranet/Internet applications. We expose the needed requirements for the developing of transactions that involve the Web and Data bases, besides gathering the functionalities of both technologies. We approach fundamentally transactional aspects of the integration environment, giving emphasis to the ACID properties. It is also proposed a classificatory approach to the transactional architectures that involve Web and Data bases, always emphasizing the behavior of the functionalities of Data Base Management Systems under each architecture, like the management of the such transactions, the control of integrity restrictions, security, access, performance, development and portability. It is approach the technologies involved in Web transactions , like CORBA, TIP (Transaction Internet Protocol), X/open, OTS (Object Transactions Service). Next, it is introduce a case studybased on one of the main Web Data Base transactional architectures, called *CGI Executable Programs*. Eventually, it will also be introduced a market tool, *Bolero*, which is based on another important transactional architecture, called *Transactional Server*. The last two examples are applied to the development of Internet/Intranet applications.

Key Words: Transaction Models, Data base, Web, Internet, Intranet

1 INTRODUÇÃO

1.1 Motivação

Nos últimos anos a Internet e, em particular, a *World Wide Web* (WWW) ou simplesmente *Web*, tem se tornado um importante meio para transmissão e disseminação de informações, acesso a dados e comunicação pessoal. Inúmeras novas aplicações, distribuídas, como comércio eletrônico, bibliotecas digitais, educação a distância e sistemas hipermídia vêm ocupando espaço cada vez mais importante na *Web*. Torna-se urgente portanto, explorar os modelos transacionais que integram esta nova mídia e outras tecnologias já bem estabelecidas para acesso a informações.

Paralelamente, os Sistemas de Gerenciamento de Banco de Dados (SGBDs) apresentaram uma evolução nos últimos anos que permitiu torná-los o componente central de modernos ambientes de computação. No entanto, os SGBDs surgiram e evoluíram para atender necessidades características de seu ambiente, sem a preocupação com uma futura e eventual integração com outros sistemas.

Diante destes fatos, propostas para a integração das duas tecnologias estão aparecendo naturalmente. Esta nova área de pesquisa, denominada *Web Banco de Dados* (*Web Database*), vem crescendo muito na medida em que se verifica a enorme potencialidade desta integração para o desenvolvimento de várias aplicações *Web* baseadas em SGBDs.

Além disso, não é desprezível a taxa de crescimento da *Web* na Internet nos últimos anos. Muito investimento tem sido feito em infra-estrutura de comunicação, visando disponibilizar acesso à internet a um número cada vez maior de usuários.

Como conseqüência, o volume de informações disponível sob a *Web* vem aumentando consideravelmente e a tecnologia de Banco de Dados representa uma parte crítica deste crescimento, não só como fonte para armazenamento e gerenciamento de objetos *Web* como também para o desenvolvimento de novas aplicações sob sistemas de Banco de Dados na *Web*, justifica-se dessa maneira a importâncias dos modelos transacionais para este mundo.

1.2 Objetivos

O objetivo principal desta dissertação é realizar um estudo de modelos transacionais, analisar aspectos da interação entre Sistemas de Gerenciamento de Banco de Dados e a *World Wide Web* no que diz respeito ao desenvolvimento de aplicações *Intranet/Internet*. São propostas três abordagens para esta análise:

- Uma abordagem baseada em requisitos necessários para desenvolvimento de transações que envolvam *Web* Banco de Dados e que reúna as funcionalidades das duas tecnologias, mas sem limitar ou eliminar características de nenhuma delas.
- Uma abordagem transacional do ambiente de integração sob o ponto de vista de Banco de Dados, com ênfase nas propriedades ACID (Atomicidade, Consistência, Isolamento, Durabilidade) das transações *Web* Banco de Dados.

- Uma abordagem classificatória das arquiteturas das transações para integração das tecnologias, enfatizando o comportamento das funcionalidades dos Bancos de Dados sob cada arquitetura descrita.

A integração destas tecnologias encontra-se em constante desenvolvimento, dessa forma, pretende-se ainda neste trabalho, apresentar uma visão global de transações gerais e do ambiente de integração *Web* e Banco de Dados apontando alguns dos principais problemas da integração no que diz respeito a transações. Neste sentido, pretende-se descrever um estudo de caso que seja representativo no estágio atual de transações para o mundo *Web* e ainda um estudo detalhado de uma ferramenta do mercado atual de desenvolvimento de aplicações para *Web*, especialmente para *e-commerce* e *e-bussiness*, que se encaixa no conceito de tratamento de transações longas.

1.3 Principais limitações

O estado da arte das transações se dá através de pesquisas na área *Web* Banco de Dados que são bastante recentes e vêm abrangendo vários tópicos distintos. Em linhas gerais, três tópicos de pesquisa têm se destacado atualmente: a visão da *Web* como um enorme Banco de Dados distribuído e multi-plataforma¹, o acesso a Bancos de Dados através da *Web* e o desenvolvimento de técnicas e ferramentas para desenvolvimento de aplicações *Web* integradas a SGBDs.

Os trabalhos relacionados à visão da *Web* como um enorme Banco de Dados abrangem, entre outros, propostas de linguagens de consulta declarativas, muitas vezes baseadas em SQL, para recuperação de

¹ Neste Caso "multi-plataforma", seria um Banco de Dados que existisse em várias plataformas diferentes, como Unix, Maiframe , entre outros

informações na *Web* (Knopnicki & Shmueli, 1995; Mendelzon, 1996 e Arocena, 1997), aspectos teóricos relacionados com a computabilidade de consultas na *Web* (Abiteboul & Vianu, 1997), consultas a objetos não estruturados ou semi-estruturados na *Web* (Christophides, 1994; Quass, 1995 e Abitebou, 1997), implementação de sistemas (*robots/repository*) para indexação e localização de informações na *Web* [Eichmann, McGregor & Danley, 1994; Selberg & Etzioni, 1995 e , Yuwono et al) e técnicas para manutenção da integridade referencial (*links*) de objetos *Web* (Pitkow & Jones, 1996 e Ingham, 1996).

Os trabalhos relacionados com o acesso a Bancos de Dados através da *Web* incluem, entre outros, implementação de arquiteturas de integração das tecnologias (Varela, 1996; Yang & Kaiser, 1996 e Hadjefthymiades & Martakos, 1997), estudo das arquiteturas existentes (Shklar, 1995; Perrochon, 1995 e Kim, 1996), análise de desempenho de arquiteturas (Newman & Fowler, 1996 e Hadjefthymiades & Martakos, 1997), processamento de transações (Little et al, 1997 e Lyon, Evans & Klein, 1997), persistência de transações (Atkinson, 1996 e Glasgow 1997], gerenciamento de objetos *Web* (Mellman, 1996) e questões de segurança envolvendo as duas tecnologias (Bina et al, 1994; Stabile, Pang & Anand , 1997 e Rahmel, 1997).

Os trabalhos que dizem respeito à técnicas e ferramentas para desenvolvimento de aplicações *Web* integradas a SGBDs compreendem, entre outros, estudo de metodologias (Varela & Hayes, 1994; Varela & Hayes 1994b e Dobson & Burrill, 1995) e de requisitos (Port et al, 1996) para desenvolvimento de aplicações, propostas e/ou análise de *softwares* que ocupam uma camada intermediária na integração das tecnologias (Frank, 1995; Plain , 1996; North, 1996b; Nguyen & Srinivasan, 1996; Hadjefthymiades & Martakos, 1996; Hunter, Ferguson & Hedges, 1996; *Internet* , 1996 e Telford,1997) e linguagens de programação na *Web* integradas a Banco de Dados (Duan, 1996; Shah , 1996, Scherer, 1996 e LaGrow, 1996).

Este trabalho não aborda a *Web* como um enorme Banco de Dados, e sim analisa aspectos funcionais da interação SGBD e *Web* para o desenvolvimento de aplicações *Intranet/Internet*. Neste sentido, se aproxima das duas últimas áreas de pesquisa citadas acima, em particular, no que envolve o *estudo de*

arquiteturas existentes, processamento de transações, persistência de transações e estudo de requisitos para o desenvolvimento de aplicações Web Banco de Dados.

O trabalho mais abrangente que se aproxima dos objetivos desta dissertação, de acordo com o que foi pesquisado na literatura, propõe o estudo de alternativas para acesso a Banco de Dados Relacionais na *Web* e o desenvolvimento de uma ferramenta para desenvolvimento de aplicações *Web Banco de Dados* (Streit, 1996). A principal diferença entre o trabalho citado e este é sua abordagem funcional. Enquanto que em (Streit, 1996), estuda-se algumas técnicas para integrar SGBD e *Web* baseadas numa única arquitetura de integração das tecnologias, este trabalho aborda a questão segundo diversos aspectos funcionais de SGBDs referente a uma taxonomia das arquiteturas existentes.

1.4 Contribuições

Como contribuições deste trabalho, espera-se:

- Identificar os principais problemas em aberto no ambiente de integração *Web Banco de Dados* sob o ponto de vista de Banco de Dados e como alguns destes problemas podem ser resolvidos sem a alteração ou limitação das tecnologias envolvidas.
- Identificar os principais requisitos funcionais dos *gateways* de integração das tecnologias, principalmente no que diz respeito ao problema da característica *stateless* do ambiente *Web* em oposição à característica *stateful* dos sistemas de Banco de Dados e uma investigação de como se comportam as propriedades ACID de Banco de Dados no ambiente de integração.

- Propor uma extensão das taxonomias das arquiteturas dos *gateways* Web Banco de Dados existentes, tendo como base diversas funcionalidades de SGBDs. Em específico, será proposto um quadro comparativo das arquiteturas dos *gateways* segundo as funcionalidades levantadas ao longo da dissertação.
- Realizar uma análise funcional qualitativa da principal arquitetura de integração *Web* e Banco de Dados existente atualmente para o desenvolvimento de aplicações *Intranet/Internet* sob SGBDs, denominada aqui de *Programas Executáveis CGI*.
- Realizar uma análise também qualitativa da arquitetura de transações *Web* Banco de Dados, que atende e resolve a problemática da manutenção das propriedades ACID. Pode-se afirmar que esta arquitetura é a mais adequada à maioria das aplicações.
- Apresentar um extenso levantamento bibliográfico sobre o assunto. Se considerarmos a novidade e atualidade da pesquisa desenvolvida neste trabalho, ainda é muito difícil encontrar artigos técnicos sobre o tema. Pretende-se aqui, reunir uma ampla bibliografia que possibilite obter uma visão global do ambiente Web Banco de Dados.

1.5 Organização

O restante da dissertação é organizado nos seguintes capítulos:

- O capítulo 2 , introduz de forma sucinta o conceito de transações e o por que da importância das transações atômicas. E apresenta também uma breve descrição das propriedades ACID.

- O capítulo 3 apresenta uma visão geral da Web, destacando arquitetura e conceitos básicos necessários ao entendimento dos demais capítulos. Enumera-se as principais vantagens da integração Web e Banco de Dados. São descritos, ainda, os principais problemas neste ambiente de integração no que diz respeito ao desenvolvimento de aplicações *Intranet/Internet*.
- O capítulo 4 aborda aspectos funcionais do ambiente de integração. Dois tópicos são discutidos em detalhes. Primeiramente é analisada a questão da natureza *stateless* da Web em oposição à natureza *stateful* de Banco de Dados. Argumenta-se sobre a necessidade dos *gateways* implementarem uma sessão de usuário junto ao SGBD de forma a poder gerenciar o estado da aplicação e aproveitar melhor os mecanismos de otimização dos atuais SGBDs. Posteriormente é descrito o problema da extensão das propriedades ACID de Banco de Dados ao ambiente de integração.
- O capítulo 5 apresenta uma taxonomia das principais arquiteturas de integração das tecnologias existentes atualmente. São destacadas vantagens e desvantagens de cada uma delas com destaque na forma como as funcionalidades dos atuais SGBDs, descritas nos capítulos anteriores, são afetadas por cada arquitetura em particular. Os principais *gateways* disponíveis no mercado são incluídos em cada categoria e um quadro comparativo das arquiteturas existentes é apresentado.
- No capítulo 6 apresenta-se vários modelos de transações na *internet*, e são discutidos vários exemplos de tecnologias utilizadas no desenvolvimento de transações na *internet*
- No capítulo 7 faz-se um estudo de caso desenvolvido por Lifschitz & Lima (1998), abordando a principal arquitetura utilizada hoje em dia para desenvolvimento de aplicações Web Banco de Dados, no caso, *Programas Executáveis CGI*.

- No capítulo 8 analisou-se uma ferramenta de mercado , denominada Bolero, que se encaixa perfeitamente com uma arquitetura de transação denominada *Servidor de Transações*
- Por fim, o capítulo 9 contém a conclusão do trabalho em função dos temas abordados nos capítulos precedentes, contribuições da dissertação e futuros trabalhos.

2 TRANSAÇÕES GERAIS

Neste capítulo é introduzido de forma sucinta o conceito de transações e o por que da importância das transações atômicas. É apresentada uma breve descrição das propriedades ACID, que são elas: atomicidade, consistência, isolamento e durabilidade, descrevendo suas principais características. Em seguida descrevemos a importância da recuperação de dados em uma transação. Ao final abordamos algumas variações dos modelos tradicionais de transações, que são chamados de modelos de transações estendidas.

2.1 Conceito de transações

Uma transação é uma unidade lógica de trabalho, em geral envolvendo diversas operações de banco de dados (Date 2000). Um exemplo padrão envolve uma transferência de uma quantia de dinheiro de uma conta A para outra conta B. É claro que são exigidas duas atualizações neste caso, uma para poder retirar o dinheiro da conta A, e outra para depositá-lo na conta B. Se o usuário declarar que as duas atualizações são parte da mesma transação, então o sistema poderá efetivamente garantir que ambas serão realizadas ou nenhuma delas, ainda que, por exemplo, o sistema venha a falhar em meio ao processo.

As operações *begin transaction*, *commit* e *rollback* são fornecidas com a finalidade de informar ao sistema quando operações distintas fazem parte da mesma transação. Basicamente, uma transação começa quando uma operação *begin transaction* é executada, e termina quando uma operação *commit* ou *rollback* correspondente é executada. Por exemplo (em pseudocódigo):

```
BEGIN TRANSACTION ; /* move $$$ da conta A para conta B */
UPDATE conta A;           /* retirada           */
UPDATE conta B;           /* depósito            */
IF tudo funcionou bem
    THEN COMMIT ;         /* fim normal         */
    ELSE ROLLBACK ;      /* fim anormal        */
END IF ;
```

2.2 Transações atômicas

O conceito de transações atômicas surgiu para tentar resolver o problema de garantir a consistência de aplicações na presença de falhas. Esta é uma técnica de tolerância à falhas de extrema usabilidade, especialmente quando múltiplos, possíveis, remotos, recursos transacionais são envolvidos (Philip et al,1987).

Para controlar a manipulação de objetos persistentes, enfrentar falhas e problemas de concorrência em sistemas transacionais, existe um conjunto de propriedades , chamadas de propriedades ACID, que as transações devem satisfazer. A seguir essas propriedades são descritas.

2.3 Propriedades ACID

As propriedades ACID são as seguintes (Date 1992):

- a) **Atomicidade** - As transações tem a garantia de serem atômicas, isto é, elas têm a garantia (de um ponto de vista lógico) de serem executadas em sua totalidade ou de não serem executadas ao todo, mesmo que o sistema falhe no decorrer do processo.
- b) **Consistência** - As transações devem preservar a consistência do banco de dados. Ou seja, uma transação transforma um estado consistente do banco de dados em outro estado consistente, sem necessariamente preservar a consistência em todos os pontos intermediários.
- c) **Isolamento** - As transações também tem a garantia de estarem isoladas uma das outras, no sentido de que atualizações feitas no banco de dados por uma dada transação *T1*, não se tornarem visíveis para qualquer transação *T2* distinta, até e a menos que a *T1* execute com sucesso uma operação *commit*. A operação *commit* faz as atualizações efetuadas pela transação se tornarem visíveis para outras transações; dizemos que essas atualizações fizeram *commit* e tem a garantia de nunca serem canceladas. Por outro lado, se a transação executar a operação *rollback*, todas as atualizações feitas pela transação serão canceladas (ou desfeitas). Nesse último caso, o efeito será como se a transação nunca tivesse sido executada.
- d) **Durabilidade** - As transações também têm a garantia de serem duráveis (persistentes) no sentido de que, uma vez que uma transação execute uma operação *commit* com sucesso, suas atualizações terão a garantia de serem aplicadas ao banco de dados, mesmo que ocorra alguma falha subsequente do sistema em algum instante.

2.4 Recuperação

Para Bjork (1973) a recuperação, considerado o tópico mais importante na área de gerenciamento de transações, em um sistema de banco de dados significa basicamente a recuperação do próprio banco de dados: isto é, restaurar o banco de dados a um estado que se sabe ser correto, ou melhor consistente¹, depois que alguma falha o leva a um estado inconsistente. E os princípios subjacentes em que se baseia essa recuperação é muito simples, e podem ser resumidos em uma palavra: **redundância**.

Devemos deixar claros que as idéias de recuperação, de fato as idéias de processamento de transações em geral, são um tanto independentes de ser o sistema subjacente relacional outro qualquer.

O que não se deve permitir que ocorra, no caso de uma transação com várias atualizações, uma atualização seja executada e outras não, pois isso deixaria o banco de dados em estado inconsistente (Gray & Reuter, 1993). O ideal é ter uma garantia sólida de que todas as atualizações foram executadas. Infelizmente é impossível ter essa garantia, há sempre a possibilidade de que as coisas saiam erradas, e saiam erradas no pior momento possível. Por exemplo poderiam ocorrer uma queda no sistema entre as operações *insert* e *update*, ou poderia ocorrer um *overflow* aritmético na operação *update*². Porém um sistema que admite o gerenciamento de transações fornece algo quase tão bom quanto essa garantia. Especificamente ele garante, que se a transação executar algumas atualizações e ocorrer uma falha, antes de a transação atingir seu término planejado, então essas atualizações serão desfeitas. Assim

¹ No caso, *consistente* significa "satisfazendo todas as restrições de integridade conhecidas". "*Consistente*" poderia ser definido como "correto no que se refere ao sistema".

² A queda do sistema também é conhecida como uma falha global, ou do sistema; uma falha de um programa individual, como um *overflow*, também é conhecida como uma falha local.

a transação ou será executada integralmente ou será totalmente cancelada, será como se ela nunca tivesse sido executada. Desse modo uma seqüência de operações que fundamentalmente não é atômica pode parecer atômica de um ponto de vista externo.

O componente do sistema que fornece essa atomicidade, ou aparência de atomicidade, é chamado de gerenciador de transações, também conhecido como monitor de processamento de transações ou monitor TP (Bernstein et al, 1990), e as operações *commit* e *rollback* são a chave para se entender o modo como ele funciona:

- A operação *commit* indica o término de uma transação bem sucedida. Ela informa ao gerenciador de transações que uma unidade lógica de trabalho foi concluída com sucesso, o banco de dados está ou deveria estar novamente em um estado consistente e todas as atualizações feitas por essa unidade de trabalho podem agora ser validadas ou tornadas permanentes.
- Em contraste, a operação *rollback* assinala o término de uma transação malsucedida. Ela informa ao gerenciador de transações que algo saiu errado, que o banco de dados pode estar em um estado inconsistente, e que todas as atualizações feitas pela unidade lógica de trabalho até agora devem ser retomadas ou desfeitas.

Devemos salientar o fato de que uma aplicação realista não somente atualizará o banco de dados, ou tentará fazê-lo, mas também enviará alguma mensagem de volta ao usuário final indicando o que aconteceu. Por sua vez o tratamento de mensagens tem implicações adicionais para a recuperação (Gray & Reuter, 1993).

Vale a pena lembrar que é possível desfazer uma atualização. Isto ocorre pois o sistema mantém um *log* ou diário em fita ou mais comumente em disco, no qual são registrados detalhes de todas as operações de atualização, em particular, imagens do objeto atualizado antes e depois das operações. Assim,

se for necessário desfazer alguma atualização, o sistema poderá usar a entrada de *log* correspondente para restaurar o objeto atualizado a seu valor anterior.

Na prática, a *log* terá duas partes, uma parte ativa ou *on-line*, e uma parte de arquivo ou *off-line*. A parte *on-line* é a parte usada durante a operação normal; do sistema para registrar detalhes das atualizações à medida que elas são executadas, e em geral está contida em disco. Quando a porção *on-line* fica cheia, seu conteúdo é transferido para a porção *off-line*, que por ser sempre processada seqüencialmente, pode estar contida em fita.

Mais um ponto importante: o sistema deve garantir que instruções individuais sejam atômicas. Essa consideração se torna particularmente significativa em um sistema relacional, na qual as instruções são de nível de conjuntos e em geral operam sobre muitas tuplas ao mesmo tempo, não deve ser possível que uma determinada instrução falhe durante o processo e deixe o banco de dados em um estado inconsistente (Date, 2000). Em outras palavras, se ocorrer um erro em meio a uma certa instrução, então o banco de dados deve permanecer totalmente inalterado.

2.4.1 Recuperação de transações

Uma transação começa com a execução bem sucedida de uma instrução *begin transaction*, e termina com a execução bem sucedida de uma instrução *commit* ou de uma instrução *rollback* (Davies, 1978). Um *commit* estabelece o que se chama, entre muitas outras coisas, um ponto de *commit* ou validação. Um ponto de *commit* corresponde portanto ao fim de uma unidade lógica e, em consequência, a um ponto no qual o banco de dados está, ou deveria estar, em um estado consistente. Em contraste o *rollback* devolve ao banco de dados ao estado em que ele se encontrava em *begin transaction*. Isso significa o retorno ao ponto de *commit* anterior.

Quando um ponto de *commit* é estabelecido:

- Todas as atualizações feitas pelo programa em execução desde o ponto de *commit* anterior são validadas, isto é, elas se tornam permanentes. Antes do ponto de *commit*, todas as atualizações devem ser consideradas como apenas tentativas, no sentido de que podem ser desfeitas subsequente. Uma vez validada, uma atualização tem a garantia de que nunca será desfeita (essa é a definição de “validação”).
- Todos os posicionamentos do banco de dados são perdidos e todos os bloqueios de tuplas liberados. Nesse caso, “posicionamento de banco de dados” se refere à idéia de que, em qualquer instante dado, um programa em execução terá possibilidade de endereçamento a certas tuplas³ (por exemplo, através de determinados cursores no caso de SQL), essa possibilidade de endereçamento se perde em um ponto de *commit* .

Podemos entender agora que as transações não são apenas a unidade de trabalho, mas também a unidade de recuperação. Portanto, se uma transação for comprometida com sucesso, então o sistema garantirá que suas atualizações serão instaladas permanentemente no banco de dados, mesmo que o sistema caia no momento seguinte. Por exemplo é bastante possível que o sistema caia depois de uma instrução *commit* ser aceita, mas antes das atualizações terem sido gravadas fisicamente no banco de dados, elas ainda podem estar esperando um *buffer* de memória principal e serem perdidas no instante da queda (Lomet & Tuttle, 1995). Mesmo que isso aconteça, o procedimento de reinicialização do sistema ainda instalará essas atualizações no banco de dados, ele é capaz de descobrir os valores que devem ser

³ Alguns sistemas oferecem uma opção pela qual o programa de fato pode ser capaz de conservar a capacidade de endereçamento para determinadas tuplas de uma transação para a seguinte.

gravados através do exame das entradas relevantes no *log*. Assim o procedimento de reinicialização recuperará qualquer transação concluída com sucesso que não tenha conseguido fazer suas atualizações serem gravadas fisicamente antes da queda.

2.4.2 *Commit* de duas fases

Nesta seção, examinaremos rapidamente uma elaboração muito importante do conceito básico de *commit/rollback*, chamada *commit* de duas fases. O *commit* de duas fases é importante sempre que uma determinada transação pode interagir com vários “gerenciadores de recursos” independentes, cada uma gerenciando seu próprio conjunto de recursos recuperáveis e mantendo seu próprio log de recuperação. Por exemplo considere uma transação sendo executada em um *mainframe* IBM que atualiza um bando de dados IMS e também um banco de dados DB2 (Cruz, 1984). Se a transação se completar com sucesso, então todas as suas atualizações, tanto os dados do IMS quanto aos dados do DB2, terão de ser validadas; inversamente, se ela falhar, então todas as suas atualizações terão de ser retomadas. Em outras palavras, não deverá ser possível que as atualizações IMS sejam validadas e as atualizações DB2 sejam retomadas, ou vice-versa, pois então a transação não seria mais atômica (Davies, 1978).

Segue-se que não se faz sentido a transação emitir, digamos um *commit* para o IMS e um *rollback* para o DB2. E mesmo que ela emitisse a mesma instrução para ambos, o sistema ainda poderia cair entre as duas, com resultados desagradáveis. Assim, em vez disso, a transação emite uma única instrução *commit* (ou *rollback*) para todo o sistema. Essa instrução *commit* ou *rollback* “global” é por um componente do sistema chamado coordenador, cuja tarefa é garantir que ambos os gerenciadores de recursos façam validação ou a retomada das atualizações pelas quais são responsáveis e uníssono, e além

disso forneçam essa garantia mesmo que o sistema falhe no meio do processo. Então é o protocolo de *commit* de duas fases que permite ao coordenador oferecer tal garantia.

Vejamos como ele funciona. Para simplificar, suponha que a transação tenha concluído com sucesso seu processamento de banco de dados; assim, a instrução no âmbito do sistema que ela emite é *commit*, e não o *rollback*. Ao receber essa solicitação de *commit*, o coordenador executa o seguinte processo em duas fases:

- a) Primeiro, ele instrui todos os gerenciadores de recursos a ficarem prontos para “seguir qualquer caminho” na transação. Na prática, isso significa que cada participante do processo, ou seja, cada gerenciador de recurso envolvido, deve forçar todas as entradas de *log* de recursos locais utilizados pela transação em seu próprio *log* físico, isto é, para o armazenamento não volátil; aconteça o que acontecer daí por diante, o gerenciador de recursos terá agora um registro permanente do trabalho que realizou para a transação, e será capaz de comprometer suas atualizações ou retomá-las, conforme seja necessário. Supondo-se que a gravação forçada seja bem-sucedida, o gerente de recursos responderá agora “OK” ao coordenador. Caso contrário, ele responderá “Não OK”.

- b) Depois de receber respostas de todos os participantes, o coordenador força a gravação de uma entrada em seu próprio *log* físico, registrando uma decisão a respeito da transação. Se todas as respostas forem “OK”, essa decisão será fazer o *commit*; se qualquer resposta tiver sido “Não OK”, a decisão será retomar. De qualquer forma, o coordenador informará então cada participante de sua decisão, e cada participante deverá então validar ou retomar a transação de modo local, conforme tiver sido instruído. Observe também que o aparecimento do registro de decisão no *log* físico do coordenador assinala a transição da Fase 1 para Fase 2.

Agora se o sistema falhar em algum ponto durante o processo geral, o procedimento de reinicialização procurará o registro de decisão do log do coordenador. Se o encontrar, então o processo de *commit* de duas fases poderá continuar do ponto em que foi interrompido. Se não achar o registro, ele presumirá que a decisão foi retomar, e mais uma vez o processo poderá ser concluído adequadamente. Vale a pena observar que se o coordenador e os participantes forem executados em máquinas diferentes, como pode acontecer em um sistema distribuído, então uma falha por parte do coordenador, e enquanto ele estiver esperando, quaisquer atualizações feitas pela transação através desse participante deverão ser mantidas ocultas de outras transações.

2.5 Modelos de transações avançadas

De acordo com Korth (1995), requisitos de atomicidade, não são desejáveis em transações que podem permanecer por horas ou até dias em execução. A recuperação de uma falha, nestes casos, implica em um custo alto: uma falha não pode resultar em desfazer tudo o que foi feito durante horas de trabalho. Frente a esses novos requisitos, e relaxamento do modelo ACID tradicional tornaram-se necessários.

Várias pesquisas foram e vêm sendo realizadas no desenvolvimento de modelos de transações de longa duração. Estas pesquisas deram origem a modelos de transações estendidas. São exemplos destes modelos: sagas, transações aninhadas e os *workflows* transacionais.

Workflows transacionais permitem representar transações complexas, como um conjunto de várias tarefas (sub-transações), ordenadas através do uso de estruturas de controle como laços (*loops*) e comandos condicionais. O tratamento de falhas destas transações envolve regras de compensação e operações de desfazer semânticas, que são associadas a cada tarefa, de

forma a com por o *workflow* transacional. Uma tarefa é um conjunto de transições entre estados válidos e um conjunto de condições que disparam a execução desta tarefa.

Workflows transacionais são amplamente dependentes de bancos de dados e destinam-se a aplicações que manipulam de forma complexa e estruturada destes dados. Estes sistemas diferem, contudo, das aplicações de gerenciamento de *workflow*. Seu principal objetivo é estruturar transações complexas de maneira a garantir sua execução de modo transacional, através do tratamento adequado de suas falhas, e do uso de pré e pós condições de controle. É por este motivo que *workflows* transacionais não são classificados com uma abordagem para o problema de gerenciamento de *workflow*, mas como uma abordagem à estruturação e execução de aplicações (tarefas). Estes sistemas podem ser, portanto, empregados no contexto maior de gerenciamento de *workflow*.

São alguns exemplos de sistemas de *workflow* transacionais o sistema Contracts, da Universidade de Stuttgart, na Alemanha, e o FlowMark da IBM.

Para Molina & Salem (1987), um grande problema com as transações descritas neste capítulo é o fato de que elas são tacitamente consideradas de duração muito curta. Se uma transação demorara um longo tempo (horas, dias, semanas) então se ela tiver de ser retomada, será necessário desfazer um grande quantidade de trabalho e mesmo se tiver sucesso, ela ainda deverá depender de recursos do sistema por um tempo excessivo, bloqueando assim outros usuários. Infelizmente muitas transações reais tendem a ser de longa duração, em especial em algumas áreas de aplicações mais recentes, como as transações que acontecem na *internet*.

As sagas são um modo de atacar esse problema. Uma saga é uma seqüência de transações curtas com a propriedade de que o sistema garante que todas as transações na seqüência serão executadas com sucesso, ou certas transações de compensação serão executadas para cancelar os efeitos de transações concluídas com sucesso dentro de uma execução geral incompleta da saga. Por exemplo, em um sistema bancário, poderíamos Ter a transação " somar R\$ 100,00 à conta A". A transação de compensação seria

obviamente “ subtrair r\$ 100,00 da conta A” . Uma extensão da instrução *commit* permite que o usuário informe ao sistema que a transação de compensação a ser executada deve ser necessária mais tarde para cancelar os efeitos da transação concluída agora (Korth & Silberchatz, 1990). Observe que, no caso ideal, uma transação de compensação nunca deve terminar com um *rollback*.

2.6 Conclusão

Neste capítulo, apresentamos uma introdução necessariamente breve ao tópico de transações. Como vimos uma transação é uma unidade lógica de trabalho e também uma unidade de recuperação. As transações tem as propriedades ACID de atomicidade, consistência, isolamento e durabilidade. O gerenciamento de transações é uma tarefa de supervisionar a execução de transações de tal modo que se possa, de fato, garantir que elas têm essas importantes propriedades. Com efeito, a função geral do sistema poderia ser definida como a execução confiável de transações.

Vimos que o sistema garante as propriedades ACID de transações diante de uma queda do sistema. Para fornecer essa garantia, o sistema deve refazer todo o trabalho feito por transações concluídas com sucesso antes da queda e desfazer todo o trabalho feito por transações iniciadas, mas não concluídas antes da queda.

Os sistemas que permitem a interação de transações com dois ou mais gerenciadores de recursos, por exemplo, dois SGBDs diferentes, devem usar um protocolo chamado *commit* de duas fases se tiverem de manter a propriedade de atomicidade de transações.

Por último, identifiquei várias pesquisas que foram e vêm sendo realizadas no desenvolvimento de modelos de transações de longa duração. Estas pesquisas deram origem a modelos de transações estendidas. Apresentou-se

sucintamente as principais características destes modelos, que são eles: sagas, transações aninhadas e os *workflows* transacionais.

Levando em consideração que esta dissertação visa enfatizar os modelos de transações na *internet*, no próximo capítulo será abordado de forma bem direcionada ao ambiente *Web* SGBD, os modelos de transações no mundo *Web* e os personagens que participam dessas transações.

3 TRANSAÇÕES NA WEB - COLOCAÇÃO GERAL DO PROBLEMA

A demanda por de novos modelos de transações pela comunidade de Banco de Dados deve ser direcionada para o desenvolvimento de tecnologia que ajude a solucionar problemas surgidos nos últimos anos devido à explosão do crescimento da comunicação, incluindo a *Internet* e, em particular, a *Web*.

Este capítulo descreve as principais características dos personagens nas transações na *Internet* e as principais características do ambiente de integração SGBD e *Web*. A seção 3.2 apresenta argumentos favoráveis à integração das tecnologias no que diz respeito ao desenvolvimento de aplicações que envolvam transações no ambiente *Web* BD e descreve também a arquitetura do ambiente de integração. A seção 3.2.1 e as seções seguintes procuram detalhar aspectos deste ambiente, destacando os principais problemas e a forma como algumas funcionalidades dos SGBDs são afetadas pelo ambiente *Web*. Por fim, na seção 3.3 é apresentada uma breve revisão do capítulo e uma conclusão final.

Como já foi esclarecido no capítulo 2, transações são as unidades básicas de trabalho dos SGBDs. Uma transação é vista como uma unidade atômica de processamento consistente e confiável, composta por uma seqüência de operações elementares sobre os dados. Como produto da sua execução, uma transação pode mudar o estado do Banco de Dados ou não provocar qualquer alteração, dependendo dos efeitos de sua execução serem aceitos (*commit*) ou cancelados (*abort*).

Um das questões que podem ser levantadas no ambiente *Web Banco de Dados* é a definição formal de uma transação de Banco de Dados no contexto *Web*. As particularidades das transações *Web Banco de Dados* são as seguintes:

- Se iniciam no cliente *Web*, que pode estar localizado em qualquer parte do mundo passando na maioria das vezes pelo servidor *Web*, sob o protocolo HTTP *stateless* e pelo 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 dos SGBDs tradicionais. Na *Web*, um usuário pode, por exemplo, abandonar a navegação sem que isso seja notificado ao servidor *Web* ou ao SGBD.
- São baseadas em páginas *Web*, ou seja, mesmo que uma página de origem contenha vários comandos para serem realizados no Banco de Dados, seria desejável que todos eles fossem considerados como uma unidade atômica de processamento.
- Podem ser resultado de várias páginas de interação com o usuário. Isto indica que uma transação na *Web* é formada por um grafo de navegação por páginas *Web*, geradas dinamicamente ou não.
- São baseadas no modelo navegacional da *Web*, sendo que não existe a priori como impor um rígido controle na ordem das transações. O usuário pode, por exemplo, submeter uma transação e logo em seguida retornar à mesma transação por meio da opção *back* do *browser* e resubmetê-la.

Diante destas particularidades, o conceito de uma transação *Web Banco de Dados* se diferencia do conceito tradicional de transação de Banco de Dados como uma unidade básica de trabalho no SGBD. Ela é baseada no

modelo navegacional hipertexto, composta por uma ou mais páginas *Web*, com uma possível interação de longa-duração com o usuário.

3.1 Os principais personagens nas transações *Internet*

Os principais personagens nas transações na *Internet* são:

- **cliente**, que é o usuário da *Internet*, onde uma aplicação é capaz de se expandir pela *Internet*;
- os **servidores**, eles são nós na *Internet* oferecendo serviços, ou produtos, para os clientes;
- a ***Internet***, que oferece serviço de transporte de dados, tanto para clientes como servidores;
- **banco de dados**, é a união de todos os bancos de dados de serviços ou produtos que poderão ser encontrados na *Internet*.

Uma transação é iniciada pelo cliente que deseja executar, de um modo atômico, ações em diferentes servidores.

A *Web* frequentemente sofre por falhas que afetam a performance e consistência de aplicações que rodam sobre ela. Uma importante técnica de tolerância a falhas é o uso de ações atômicas para controlar operações em serviço. Estas ações atômicas garantem a consistência de aplicações a despeito de acessos concorrentes e falhas.

Técnicas para implementar transações em objetos distribuídos são conhecidas: em ordem para vir a ser uma transação consistente, um objeto requer facilidade para controle de concorrência, persistência, e habilidade para participar em um protocolo de *commit*. Enquanto for possível fazer aplicações de transações do lado do servidor, os *browser* tipicamente não possuem esta habilidade, uma situação como esta deve ser persistida para um previsto

futuro. O *browser* não será normalmente capaz de tomar parte em aplicações de transação.

Mas vamos abordar neste estudo também um projeto e implementação de um esquema que permite o *browser* não transacional para participar de aplicações transacionais, desse modo alimentando a necessidade de garantidas transações *end-to-end*.

3.1.1 Gerenciamento da execução de transações

As transações devem ser executadas concorrentemente, de forma transparente e confiável para a aplicação. A transparência refere-se à capacidade da manutenção do Banco de Dados em um estado consistente, mesmo na presença de várias transações executadas simultaneamente. A confiabilidade refere-se à capacidade do sistema de Banco de Dados em resistir aos diversos tipos de falhas que podem ocorrer no ambiente durante a execução de uma transação.

A avaliação da correção da execução concorrente de transações é feita com um critério que indica se o resultado final é ou não aceitável. O critério tradicionalmente empregado é o de seriabilidade do escalonamento das ações das transações, isto é, verifica-se se o resultado final é equivalente ao de uma execução serial dessas ações. Para isso podem ser usados diferentes mecanismos, sendo que o método mais difundido é o baseado em bloqueios (Korth & Silberwchatz, 1997 e Elmasri & Navathe, 1994).

Uma avaliação cuidadosa de transações *Web* Banco de Dados indica que os mecanismos tradicionalmente empregados para verificação da execução serial das transações em SGBDs tradicionais podem não ser apropriados ao ambiente *Web* Banco de Dados. De fato, os critérios de correção e seriabilidade de uma transação *Web* Banco de Dados devem levar em consideração os seguintes aspectos:

- Ser orientado a páginas *Web*, onde possivelmente uma página pode conter várias operações no Banco de Dados ou ser parte de uma única operação no Banco de Dados.
- Ser baseado num controle de concorrência para interação com transações de longa duração. Isto é decorrente do fato de que os mecanismos atuais, como bloqueios, podem diminuir a concorrência e até mesmo eliminá-la dependendo da granularidade do bloqueio, em virtude do tempo potencialmente "infinito" que uma transação *Web* Banco de Dados pode levar.
- Ser flexível o suficiente de forma a englobar combinação de várias páginas *Web* para 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.

Todos os critérios de correção e seriabilidade adotados atualmente pelos *gateways Web* Banco de Dados são baseados no critério utilizado pelo SGBD que se está utilizando, ou seja, baseados nos mecanismos tradicionais. Como será visto no item 4.1 estes mecanismos são insuficientes para garantir, em algumas arquiteturas de integração das tecnologias, a consistência e o isolamento de transações *Web* Banco de Dados.

3.1.2 Modelos de transação

Modelos de transação têm por objetivo fornecer um contexto definido para padrões de processamento em uma aplicação (Gray & Reuter, 1993). Estes padrões variam de acordo com os requisitos que motivaram a proposição do modelo. As transações gerenciadas pela maioria dos SGBDs existentes

seguem o modelo de transações planas. O modelo é assim chamado porque existe uma única camada de controle pela aplicação. Todas as operações entre os delimitadores de início e de término da transação possuem o mesmo nível, e são aceitas ou rejeitadas em bloco. As maiores restrições com relação às transações planas são a impossibilidade de se fazer referência a resultados intermediários e aceitar ou cancelar apenas parte das ações. Além disso, o modelo de transações planas é muito útil para transações de curta duração. Este modelo tem sido experimentalmente estendido para atender aos requisitos de novas e mais complexas situações, dando origem a modelos específicos, como o modelo de transações aninhadas e o modelo SAGAS (Gray & Reuter, 1993 E Silva, 1994).

No contexto *Web* Banco de Dados as dificuldades principais estão associadas à definição do modelo de transações que se adapte ao novo ambiente. O problema que precisa ser melhor estudado é o acesso ou atualização de muitos objetos no Banco de Dados por um período de tempo possivelmente longo, sem que suas transações tenham um compromisso (*commit*) e sem inutilizar ou parar as outras transações. Este fato sugere que o modelo de transações planas pode não ser o modelo mais adequado para o ambiente *Web* Banco de Dados.

3.2 Integrando a *Web* e Banco de Dados

Existem diversas aplicações de Banco de Dados que podem ser transportadas para o ambiente *Web* e várias aplicações *Web* que podem usar Bancos de Dados como mecanismos mais eficientes para o armazenamento de informações. Neste sentido, a *Web* está passando rapidamente da condição de troca irrestrita de documentos, como projetado inicialmente, para a condição de se tornar uma plataforma de desenvolvimento de inúmeras aplicações baseadas em Banco de Dados. Além disso, vêm crescendo as soluções

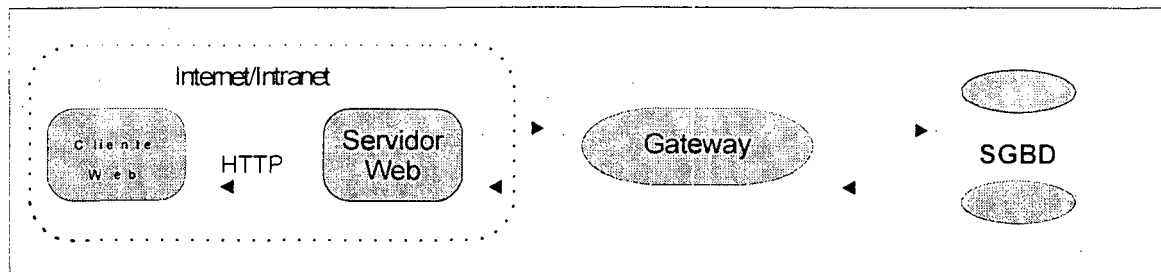
Intranet integradas a Banco de Dados: uma rede corporativa de computadores baseada nos padrões de comunicação da *Internet* pública e da *Web*, cujos dados são armazenados e gerenciados por SGBDs.

De modo geral pode-se enumerar as seguintes vantagens da integração *Web* e Banco de Dados:

- a) Os mecanismos de armazenamento e acesso de dados da *Web* não são muito eficientes nem padronizados. Muitas aplicações baseadas na *Web* armazenam dados em arquivos convencionais e, na maioria das vezes, usam linguagens de propósito geral como C, C++ ou *Perl*, para armazenamento e acesso aos dados. Quando o volume de informações previsto cresce muito estas linguagens podem se tornar inadequadas.
- b) Existe atualmente uma grande massa de dados residindo e sendo gerenciada por SGBDs. A *Web* veio abrir a possibilidade de dispor estas informações a um número ilimitado de usuários em redes locais ou remotas (Özsu & Valduriez, 1991). Além disso, a *Web* vem possibilitando o desenvolvimento de aplicações baseadas em SGBDs capazes de conectar membros de uma organização sob uma rede heterogênea, local ou remotamente.
- c) Um dos grandes problemas dos atuais SGBDs é que o desenvolvimento de aplicações dirigidas a usuários é limitado e muito dependente das ferramentas proprietárias distribuídas pelos fabricantes. Em particular, há o problema das interfaces do usuário para Banco de Dados.
- d) A *Web* está possibilitando o desenvolvimento e expansão de novas aplicações baseadas em SGBDs. São exemplos dessas aplicações compras eletrônicas via *Internet* e bibliotecas virtuais.

Uma arquitetura típica do ambiente de integração *Web* e Banco de Dados é mostrada na figura 3.1 abaixo. Nela estão os seguintes componentes necessários para se construir aplicações *Web* Banco de Dados:

Figura 3.1 - Arquitetura do ambiente de integração *Web* e BD.



- Cliente *Web*
- Servidor *Web*
- *Gateway*
- Servidor de Banco de Dados

O cliente e o servidor *Web* podem estar dentro dos limites de uma *Intranet* sendo somente usados por membros de uma organização ou fazer parte da Internet global. O *gateway* é o software responsável pelo gerenciamento da comunicação e por proporcionar serviços de aplicação entre o servidor *Web* e o servidor de Banco de dados. Ele reside tipicamente no lado servidor *Web*, e é composto por um ou mais programas *scripts*. Ao servidor de Banco de Dados cabe a tarefa de gerenciar os dados residindo no Banco de Dados. O servidor *Web* tem a tarefa de receber os pedidos dos clientes *Web* e retornar os dados enviados pelo *gateway* para serem exibidos pelo cliente *Web* ao usuário final.

Baseado na figura 3.1 anterior, é apresentado a seguir o fluxo de dados de uma aplicação no ambiente de integração:

- Inicialmente o cliente *Web* solicita um pedido ao servidor *Web* via protocolo HTTP.

- O servidor *Web* dispara um processo no *gateway* enviando os parâmetros recebidos do cliente *Web*.
- O *gateway* trata os parâmetros recebidos, formula o comando SQL, abre uma conexão com o SGBD e espera pela resposta.
- O SGBD atende a solicitação e retorna os dados ao *gateway*.
- O *gateway* trata os dados recebidos e os repassa ao servidor *Web* num formato que o cliente *Web* entenda.
- O servidor *Web* retorna os dados enviados pelo *gateway* ao cliente *Web*.
- O cliente *Web* identifica o formato dos dados e os exibe ao usuário cliente.

Toda essa dinâmica varia de arquitetura para arquitetura e será descrita com mais detalhes nas seções seguintes e nos demais capítulos.

3.2.1 Problemas de integração *Web* e Banco de Dados

Apesar das inúmeras vantagens proporcionadas pela integração SGBD e *Web* o desenvolvimento de aplicações nesse ambiente ainda está repleto de problemas técnicos cujas soluções não são triviais. Esta seção pretende discutir alguns dos problemas funcionais da integração das tecnologias. Entre outros, são destacados os aspectos transacionais relativos às aplicações *Web* Banco de Dados, aspectos de segurança neste novo ambiente, restrições de integridade, desempenho, desenvolvimento, portabilidade e linguagens de programação. Também são apontados problemas em aberto e, quando possível, soluções alternativas existentes.

3.2.2 Sistemas legados

O problema de sistemas legados (*legacy systems*) é inerente a qualquer tecnologia nova como a *Web*. Sistemas legados são sistemas que resistem a modificação e evolução. Os SGBDs representam, no contexto atual do ambiente *Web*, um importante subconjunto de sistemas legados (Shklar, 1995).

Até o momento, todos os SGBDs comerciais foram projetados e implementados sem levar em consideração as necessidades especiais do ambiente *Web*.

3.2.3 Problemas transacionais

Um dos grandes problemas da integração SGBD e *Web* é a natureza "sem estado" (*stateless*) das transações *Web* em decorrência do protocolo HTTP usado na comunicação cliente *Web* e servidor *Web*. Um servidor *Web* responde a uma solicitação do cliente *Web* retornando uma página HTML ou disparando uma aplicação externa via interface CGI, por exemplo. Uma vez que o pedido é atendido, a transação é completada e a conexão se encerra. O servidor *Web* não armazena nenhuma informação sobre a aplicação e/ou o usuário da aplicação

No entanto, surgem muitos problemas nos projetos de aplicação *Web* sobre Banco de Dados. De fato, é sabido que os atuais SGBDs foram projetados para atenderem pedidos de usuários que se conectam ao Banco de Dados em sessões contínuas, ou seja, o usuário fica conectado ao SGBD enquanto durar sua sessão (Gray & Reuter, 1993). A mudança para um ambiente baseado em transações *stateless* como o da *Web*, implica em vários questionamentos, como, por exemplo, uma definição formal de transação *Web* Banco de Dados, o gerenciamento da execução de transação neste ambiente

de integração e a necessidade de se verificar se o modelo de transações usados nos atuais SGBDs é adequado ao ambiente *Web* Banco de Dados. Estes elementos serão discutidos nas três sub-seções seguintes.

É importante ressaltar, no entanto, que o objetivo aqui não é propor soluções mas sim, identificar, como contribuição, alguns desafios técnicos a serem superados no contexto transacional de integração *Web* e Banco de Dados.

3.2.4 Segurança e acesso ao Banco de Dados

Outro problema importante quando se discute a integração *Web* e Banco de Dados é relativo à segurança nas transações entre o cliente *Web*, o servidor *Web* e o SGBD (Bina Et Al, 1994 e Rahmel, 1997). Três pontos se destacam: a segurança na comunicação entre cliente *Web* e servidor *Web*, a segurança no servidor *Web*, e a segurança e acesso ao Banco de Dados, que será analisada nesta seção.

A segurança e acesso ao Banco de Dados diz respeito a uma das importantes funcionalidades dos atuais SGBDs. Pode-se ver a *Web* como um usuário do SGBD como qualquer outro. Nestes termos, os mecanismos de segurança e acesso ao Banco de Dados poderiam funcionar como os já existentes. Em particular, o uso de visões (Elmasri & Navathe, 1994) proporcionado pelos SGBDs no ambiente de integração pode ser bastante útil. Além disso, alguns fatores devem ser levados em consideração com relação à segurança e acesso ao Banco de Dados, como os destacados a seguir:

- a) Gerenciador de acesso: a maioria dos SGBDs mantém permissões para acesso ao Banco de Dados (senhas de *login*) e aos objetos do Banco de Dados (*grants*). Caso o contexto da aplicação seja uma Intranet

pressupõem-se que os usuários sejam limitados e assim o gerenciamento de senhas e *grants* pode ser o habitual.

- b) **Acessos simultâneos:** o número de acessos simultâneos ao Banco de Dados é geralmente limitado. Dependendo dos requisitos da aplicação, um usuário pode não conseguir completar sua transação devido a sobrecarga no gerenciador de acessos do SGBD. Alguns *gateways* de integração podem oferecer um cache de acessos ao Banco de Dados e os compartilhar entre os usuários, se o SGBD permitir.
- c) **Autorização:** em alguns *gateways* de integração *Web* e Banco de Dados, a autorização para execução de uma aplicação *Web* que consulta ou atualiza o Banco de Dados é dada ao software usuário que ativou o *gateway* (geralmente o próprio servidor *Web*).
- d) **Recuperação:** um esquema de recuperação em Banco de Dados tradicionais é ativado como consequência de diversos tipos de falhas, como por exemplo, erros lógicos, paradas de sistema e falhas de disco.

3.2.5 Controle de restrição de integridade

As restrições de integridade presentes na grande maioria dos SGBDs têm por objetivo assegurar que as eventuais atualizações respeitem as regras estruturais e semânticas definidas para o Banco de Dados.

Existem basicamente dois tipos de restrições de integridade (Korth & Silberwchatz, 1997): restrições de modelo, também denominadas de restrições de chave e restrições de domínio. As restrições de modelo são geralmente fáceis de serem testadas e dizem respeito à organização interna do Banco de Dados, não sendo afetadas pelo ambiente *Web*. Já as restrições de domínio

são restrições sobre os valores que podem ser assumidos, por exemplo, por atributos de tabelas de SGBDs relacionais. Embora sejam fáceis de serem testadas pelo SGBD isoladamente, podem gerar problemas quando se usa o ambiente *Web*.

Cabe observar também que nem toda restrição de domínio imposta no Banco de Dados é passível de ser tratada no lado cliente *Web*. Por exemplo, restrições de domínio que são avaliadas pelo SGBD com base nos dados armazenados se enquadram nesta categoria. Neste caso não resta outra alternativa senão tratá-las no *gateway*.

3.2.6 Desempenho

O problema de desempenho relacionado ao ambiente *Web* refere-se ao tempo de resposta a uma requisição do usuário, o que geralmente depende do desempenho dos vários componentes da *Web* (Murta Et Al, 1996 e Newman & Holzbaur, 1996). Três desses componentes afetam diretamente uma transação *Web* Banco de Dados: 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*, considerando-se os diversos tipos de mídias possíveis, o gerenciamento do cache local e a execução de códigos associados aos dados enviados.

E por fim, o servidor *Web* que, como analisado em (MURTA et al, 1996), é um dos componentes determinantes no tempo de resposta de uma transação cliente, já que ele é afetado pela carga de pedidos, pela plataforma de hardware e pelo ambiente de software .

Além destes fatores pode-se identificar outros três que afetam particularmente o desempenho de aplicações *Web* Banco de Dados. São eles: otimização de consultas, número de usuários e balanceamento de carga. A

otimização de consultas será abordada no próximo capítulo. Os outros dois são descritos brevemente abaixo:

- Número de usuários: aplicações *Web* podem atingir um grande número de usuários simultâneos, o que pode degradar o desempenho do servidor *Web*, do *gateway* ou até mesmo do próprio SGBD.
- Balanceamento de carga: é imprevisível a carga de processamento que pode ser encontrada em aplicações *Web* Banco de Dados e seu gerenciamento envolve um grande número de fatores, entre os quais a habilidade da aplicação em lidar com consultas complexas sobre Bancos de Dados volumosos.

3.2.7 Desenvolvimento e portabilidade

No que diz respeito ao ambiente de desenvolvimento e portabilidade de aplicações *Web* baseadas em Banco de Dados os seguintes pontos merecem destaque:

- Aplicações *Web* Banco de Dados devem ser fáceis de serem estendidas a novas versões HTML. Adicionalmente, elas devem ser flexíveis caso ocorram mudanças, por exemplo, nas versões do protocolo HTTP ou da interface CGI.
- É recomendável a existência de mecanismos eficientes para transferir variáveis de entrada do cliente *Web* (e.g., dos formulários) para as consultas (e.g., SQL) no servidor de Banco de Dados.

- A estrutura para desenvolvimento de aplicações *Web Banco de Dados* deve ser flexível, pouco dependente do esquema do Banco de Dados, com modelagem voltada para o ambiente *Web*, sem grandes conhecimentos de interfaces, ou de programação do Banco de Dados, padronizada e portátil .
- A falta de metodologias para desenvolvimento de aplicações é uma característica do ambiente *Web* atualmente, por ser uma tecnologia muito recente. Hoje em dia alguns *gateways Web Banco de Dados* (Allaire, 1997 E Hadjefthymiades & Martakos, 1997) já proporcionam técnicas como SQL dinâmico (*dynamic SQL*) para tornar o ambiente de desenvolvimento mais flexível.
- A portabilidade no ambiente *Web Banco de Dados* não apresenta robustez. Mesmo padrões considerados abertos, uma característica prevista para o ambiente *Web*, ainda não se firmaram, gerando grandes problemas de portabilidade.

3.2.8 Linguagens de programação

Os benefícios proporcionados pela integração *Web Banco de Dados* também são afetados pelo tipo de linguagem que pode ser usado para o desenvolvimento. Algumas linguagens têm emergido para aplicações *Web Banco de Dados* como Java, C, C++, *JavaScript*, *VBScript*, Lua e PERL. Adicionalmente fabricantes de SGBDs continuam a usar suas linguagens proprietárias (PL da *Oracle*, por exemplo) como linguagens de desenvolvimento para aplicações *Web Banco de Dados*. Toda essa variedade de linguagens cria problemas de portabilidade e complexidade da aplicação *Web Banco de Dados*.

3.3 Conclusão

A tecnologia *Web* e, em particular, a tecnologia de integração *Web* e Bancos de Dados ainda está em pleno desenvolvimento. Como ressaltado ao longo deste capítulo ainda existem vários problemas técnicos para que a execução de transações neste ambiente seja satisfatória.

Alguns destes problemas são inerentes ao ambiente *Web* ou ao SGBD isoladamente e, portanto, sua solução depende de alterações funcionais das tecnologias envolvidas ou a busca por ferramentas proprietárias. Por exemplo, o suporte a um modelo de transações adequado ao ambiente *Web* implica em mudanças significativas no modelo de transações atualmente suportados pelos SGBDs. Outro exemplo é a solução para a limitação da interface *Web* (HTML) que não proporciona suporte a interfaces orientadas a dados como atualmente suportado pelas aplicações de Banco de Dados cliente/servidor. Neste caso, deve-se estender a linguagem HTML ou usar linguagens proprietárias gráficas suportadas por *browsers Web*, como Java.

A alteração das tecnologias envolvidas não é a solução mais adequada para os problemas abordados, a não ser se for patrocinada por organizações interessadas na padronização do ambiente de integração ou para efeitos de pesquisa de ponta. Diante deste fato, cabe ao desenvolvedor da aplicação identificar as limitações que a integração das tecnologias pode gerar e solucioná-las da melhor forma possível de acordo com as ferramentas disponíveis. Explicar esses problemas e explicitar essa situação foram objetivos desse capítulo.

No capítulo seguinte, discute-se em mais detalhes algumas funcionalidades particularmente importantes quando do desenvolvimento de aplicações *Web* Banco de Dados. Em particular, é discutido o problema decorrente da integração de sistemas *stateless* como a *Web* com sistemas *stateful* como os sistemas desenvolvidos sobre Banco de Dados e o problema da manutenção das propriedades ACID de Banco de Dados no ambiente de integração.

4 FUNCIONALIDADES WEB SGBD

A integração de dois ambientes distintos como a *Web* e os SGBDs pode levar à perda de algumas funcionalidades importantes. No capítulo anterior, foram abordados alguns problemas quando se considera o ambiente de integração para o desenvolvimento de aplicações *Intranet/Internet*. Procurou-se, sempre que possível, apontar os cuidados que se deve ter para que nenhuma das tecnologias envolvidas fossem limitadas de alguma forma.

Este capítulo trata detalhadamente de dois aspectos funcionais relacionados à natureza das tecnologias envolvidas. Na seção 4.1 analisa-se as conseqüências da natureza *stateless* da *Web* sobre o ambiente de integração e discute-se a extensão das propriedades ACID a serem verificadas pelos SGBDs no ambiente integrado.

Procurar-se-á apresentar argumentos de que a solução do problema *stateless* da *Web* no ambiente de integração deve, necessariamente, ser baseado numa solução em que o *gateway* que integra as tecnologias implemente uma sessão contínua de usuário junto ao SGBD enquanto a aplicação estiver sendo executada. Esta solução, como será descrito, não altera a natureza do ambiente *Web* ou dos atuais SGBDs, característica identificada como essencial para integração adequada das tecnologias.

A extensão das propriedades ACID para o ambiente de integração é um assunto bem mais delicado e ainda pouco estudado.

A abordagem do problema, neste trabalho, procura estabelecer um conjunto de requisitos a ser suportado por qualquer solução que requeira

propriedades similares às aplicações desenvolvidas sob sistemas de Banco de Dados cliente/servidor.

Novamente, procura-se abordar o problema de forma a não limitar ou alterar a natureza do ambiente *Web* ou dos SGBDs, individualmente.

4.1 Sistemas *Stateless* versus sistemas *Stateful*

Os atuais SGBDs são providos de mecanismos que permitem gerenciar várias conexões simultâneas de usuários distintos. Cada usuário, quando inicia uma aplicação, permanece, na maioria das vezes, conectado ao SGBD enquanto durar sua sessão de trabalho com o SGBD, ao fim da qual, é enviada uma mensagem de solicitação de finalização da sessão. Devido a esta característica, os SGBDs são classificados como sistemas orientados a sessão contínua de usuário (Gray & Reuter, 1993). Em particular, os SGBDs são considerados sistemas *stateful* pois mantêm o estado das informações sobre as ações do usuário enquanto durar sua interação com o SGBD.

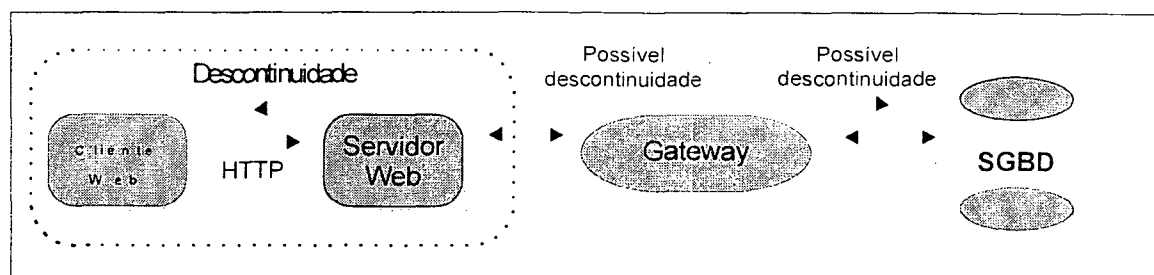
Como definido em (Comer & Stevens, 1994), "estado da informação é um conjunto de informações que é mantido por um sistema sobre o status das interações em andamento com seus usuários". O estado da informação inclui, entre outros, a situação de todas as variáveis e seus valores. Sistemas que não mantêm qualquer estado da informação são chamados sistemas *stateless*. A *Web* é um sistema tipicamente *stateless* já que é baseado no padrão de comunicação cliente/servidor em que a próxima ação do servidor *Web* não depende da ação tomada anteriormente (Perrochon & Fischer, 1995).

Os problemas decorrentes da natureza *stateless* da *Web* e suas conseqüências no ambiente de integração foram pouco estudados até o momento. Em (Perrochon, 1994) é descrito um projeto para o desenvolvimento de servidores de translação (*translation server*) que proporcionam acesso *stateful* a sistemas *stateless* como o da *Web*. Alguns grupos trabalham para

incorporar características *stateful* à linguagem Java (Atkinson Et Al, 1996 E Glasgow, 1997). No entanto, estes projetos estão mais direcionados ao ambiente *Web* do que propriamente ao ambiente de integração *Web* e Banco de Dados. De acordo com o que foi pesquisado, o único trabalho encontrado que aborda alguns aspectos do problema direcionado ao ambiente de integração é (Hadjefthymiades & Martakos, 1997), embora restrito ao uso da interface CGI.

No ambiente de integração SGBD e *Web* a manutenção da sessão contínua do usuário sob o SGBD é perdida. De fato, como mostrado na figura 4.1, a natureza do ambiente *Web* implica necessariamente numa descontinuidade entre uma interação do usuário e a próxima, devido ao fato do protocolo HTTP usado na comunicação entre o cliente *Web* e o servidor *Web* ser *stateless*. Adicionalmente, dependendo da forma como o *gateway* que integra os dois ambientes é implementado, pode haver outra descontinuidade entre o *gateway* e o SGBD.

Figura 4.1: Descontinuidade da sessão de usuário.



A característica *stateless* da *Web* em oposição à característica *stateful* dos SGBDs tem várias conseqüências no que diz respeito ao desenvolvimento de aplicações *Web* Banco de Dados. Duas se destacam, porque implicam na perda de funcionalidades desejáveis no ambiente de integração:

- O gerenciamento do estado da aplicação.
- O desempenho de aplicações no SGBD.

4.1.1 Gerenciamento do estado da aplicação

O estado da aplicação no contexto de Banco de Dados é um conjunto de informações mantidas pelo SGBD relativas à conexão atual do usuário, como por exemplo, quem é o usuário, qual a transação ainda em execução submetida por ele, quais objetos estão sendo gerenciados pelo SGBD e cursores SQL pendentes (Gray & Reuter, 1993).

O gerenciamento do estado da aplicação é uma funcionalidade particularmente importante no contexto de Banco de Dados e que é perdida no ambiente de integração devido à característica *stateless* da *Web*. De fato, os seguintes exemplos, tipicamente encontrados em aplicações de Banco de Dados convencionais, ilustram o problema:

- a) **Autenticação de usuário:** uma função importante do SGBD diz respeito ao gerenciamento da conexão do usuário da aplicação junto ao SGBD. Nos SGBDs o usuário inicia uma conexão onde é feita uma prévia verificação para acesso ao Banco de Dados segundo o par (*login*, *password*) e em todas as transações que se seguem o usuário não mais precisa se identificar junto ao SGBD até o fim de sua sessão.
- b) **Atualização de dados:** é sabido que muitos dos mecanismos para atualização de um Banco de Dados são na realidade seqüências de operações, que dependem da manutenção do estado da aplicação. Por exemplo, para alterar uma informação no Banco de Dados, primeiro deve-se autenticar a identidade do usuário, verificar se de fato ele pode realizar a operação, em seguida mostrar os dados a serem modificados, receber os dados alterados, confirmar a validade e por fim gravá-los no Banco de Dados, atualizando os índices se necessário.

- c) **Uso de cursores:** até mesmo uma simples aplicação para consulta aos dados pode se tornar problemática. A maioria dos Banco de Dados relacionais, por exemplo, dispõe de um mecanismo que preserva a posição do cursor na tabela para a recuperação posterior dos demais registros sem que seja necessário nova execução da consulta. Devido à característica *stateless* do ambiente *Web*, a implementação deste recurso deixa de ser trivial já que a conexão entre o cliente *Web* e o Banco de Dados é finalizada após a execução da consulta.

Os exemplos anteriores dão uma visão geral do problema de gerenciamento do estado da aplicação, que poderia ser em parte ou totalmente resolvido se as informações sobre o estado da aplicação do usuário pudessem ser mantidas no servidor *Web*.

No entanto, além do problema de ter que estender as funcionalidades dos atuais servidores *Web* que não foram projetados para gerenciar o estado de aplicações, esta solução seria ineficiente. Três fatores contribuem para isso:

- a) Pode existir um número arbitrário de clientes *Web* com um estado válido a qualquer momento. Manter a informação de todos os clientes no servidor *Web* pode sobrecarregá-lo.
- b) O servidor *Web* não tem uma maneira segura de saber se um usuário cliente já finalizou seus pedidos devido ao modelo navegacional da *Web*. Assim seria necessário um mecanismo auxiliar que determinasse quando liberar o estado da informação cliente.
- c) A quantidade de informação a ser mantida para alguns clientes pode ser muito grande e, portanto, pode ser outra forma de sobrecarga no servidor *Web*, afetando todos os outros usuários clientes.

Dessa forma, o problema do gerenciamento do estado da aplicação deve ser tratado ou no cliente *Web* ou no *gateway* integrador. Na seção a seguir são

apresentadas duas soluções que procuram resolver o problema para aplicações. No entanto, como será visto, são soluções com várias restrições. Uma solução mais abrangente e mais sofisticada envolvendo o *gateway* é apresentada mais adiante.

4.1.1.1 Campos ocultos nas páginas *Web*

A solução mais simples para a manutenção mínima do estado da aplicação é inserir o estado da aplicação nas próprias páginas que estão sendo submetidas pelo usuário. Existem duas formas para implementar esta técnica:

- a) Passar a informação em campos ocultos na página HTML. Estes campos não são visualizados pelo usuário quando o conteúdo da página lhe é exibido. A linguagem HTML disponibiliza a *tag* `<hidden>` que pode ser usada com esta finalidade.
- b) Pode-se colocar as informações na URL como parâmetros, de acordo com a especificação CGI.

Estas informações são enviadas para o *gateway* na próxima ação solicitada pelo usuário. O *gateway*, então, vai tratá-las e novamente incluí-las, se necessário, na próxima página de interação do usuário.

Para aplicações mais simples que precisem reter poucas informações durante a execução da aplicação, como por exemplo, qual o usuário da aplicação, esta solução é bastante eficaz. Além disso esta solução implica em um esforço adicional no desenvolvimento da aplicação que deve ser muito bem modelada e implementada, visto que o desenvolvedor deve prever quais as informações escondidas devem ser transportadas entre as páginas da aplicação durante sua execução.

4.1.1.2 Cookies

Um *cookie* é um item de dado que é armazenado em um arquivo texto no cliente *Web*. Quando o *browser* ativa uma página de uma determinada aplicação, esta página pode conter comandos *cookies* para armazenar dados no sistema de arquivos da máquina onde o *browser* é executado. Estes dados podem ser, por exemplo, uma informação entrada pelo usuário em um formulário HTML. Mais tarde, uma outra página poderá recuperar esta informação a partir do *browser* do usuário quando o *gateway* enviar um comando com o item de dado a ser recuperado. O *browser* recupera o *cookie* especificado e o retorna para o *gateway* na próxima interação.

Tecnicamente, o funcionamento dos *cookies* é o seguinte: quando o *gateway* identifica um novo usuário, ele adiciona um *header* extra na resposta enviada ao servidor *Web*, contendo um identificador para o usuário e outras informações que o *gateway* possa coletar a partir dos dados enviados pelo cliente. Este *header* informa ao cliente *Web* para adicionar as informações enviadas no arquivo *cookie* do cliente. Posteriormente, sempre que o *gateway* solicitar, serão enviadas as informações contidas no *cookie* com o *header* extra no pedido. O *gateway* usa estas informações para recuperar o estado corrente da aplicação de cada usuário quando ele realiza um novo pedido.

Um *Cookie* é um mecanismo relativamente simples e mais funcional do que a técnica anterior para gerenciar o estado da aplicação.

A técnica de manter uma sessão contínua com o SGBD pelo *gateway* é bastante útil para gerenciar o estado da aplicação do usuário uma vez que todo o estado do usuário já mantido pelo SGBD pode ser compartilhado com o *gateway*.

Apesar das vantagens desta solução para gerenciar o estado da aplicação, surgem alguns inconvenientes que precisam ser tratados pelo *gateway*. São eles:

- a) **Expiração do estado:** como HTTP é um protocolo *stateless*, é possível que um usuário interrompa a navegação sem notificar o servidor *Web*. O *gateway* deve lidar com tal término não previsto para não manter infinitamente o estado da informação no *gateway*. Este problema pode ser resolvido, por exemplo, se o *gateway* implementar mecanismos de *timeouts*.
- b) **Proteção do estado:** como o protocolo HTTP não relaciona dois pedidos sucessivos de um mesmo usuário, um pedido mal intencionado pode tentar acessar o estado de outros usuários se não houver um controle apropriado. O *gateway* deve criar mecanismos para proteger o estado do usuário, como por exemplo, usar técnicas de criptografia dos dados.
- c) **Evitar estados da transação desnecessários:** um típico *browser Web* disponibiliza as opções de navegação *back/forward* baseado no cache local. Na ausência de cópia no cache, o *browser* realiza um novo pedido ao servidor *Web*, o que pode gerar um novo estado da transação já mantido pelo *gateway* e, portanto, desnecessário ou até mesmo inconsistente.

A solução apresentada nesta seção aproveita a característica *stateful* dos SGBDs e, se implementado eficientemente, resolve o problema de gerenciamento do estado da aplicação. Na seção seguinte, será visto que esta solução é, também, a mais adequada para resolver a perda de outra importante funcionalidade no ambiente de integração decorrente da natureza *stateless da Web*: o desempenho do SGBD.

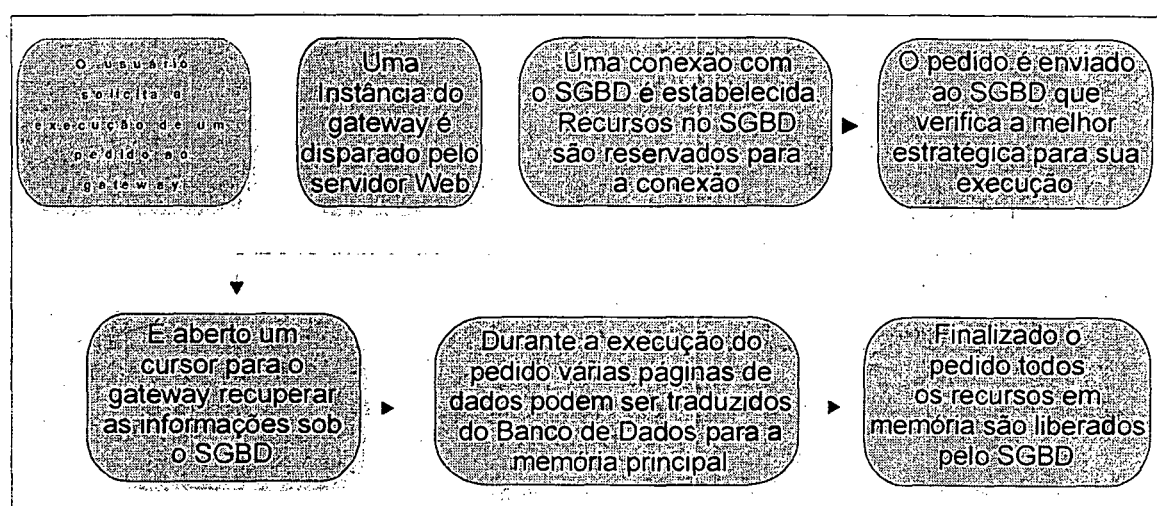
4.1.2 Desempenho do SGBD

O objetivo desta seção é discutir como o desempenho dos atuais SGBDs é afetado pela característica *stateless da Web* no ambiente de integração. Em

particular, será dada atenção especial na forma como os mecanismos de otimização de consultas dos SGBDs são influenciados pelo problema.

Como já destacado, existem duas alternativas para se implementar o *gateway* que integra as duas tecnologias. Ou ele implementa uma sessão contínua de usuário sob o SGBD enquanto durar a aplicação ou ele não implementa. A figura 4.2 a seguir ilustra o processo de execução de um *gateway* que não implementa uma sessão de usuário contínua sob o SGBD.

Figura 4.2 : *Gateway* que não implementa sessão contínua .



A figura 4.2 destaca as implicações sobre o SGBD da abertura e encerramento da conexão com o SGBD para cada interação do usuário. Primeiro, o estabelecimento de conexões com o servidor de Banco de Dados destina recursos substanciais, como memória, *threads* e processos de verificação de autorização, que requerem um tempo significativo para sua finalização. Estes recursos são liberados ao ser encerrada a interação com o SGBD. Na próxima interação do usuário, todo o mecanismo se repetirá.

Estudos de desempenho como feitos Em (Hadjefthymiades & Martakos, 1997) têm mostrado na prática que a estratégia de abrir e encerrar a conexão com o SGBD não é adequada no ambiente de integração, principalmente se várias interações são submetidas ao SGBD durante a aplicação. Assim, a alternativa de manter contínua a sessão do usuário de forma que a conexão

com o SGBD não seja finalizada para cada pedido do usuário passa a ser um requisito essencial para o desempenho de operações sob o SGBD.

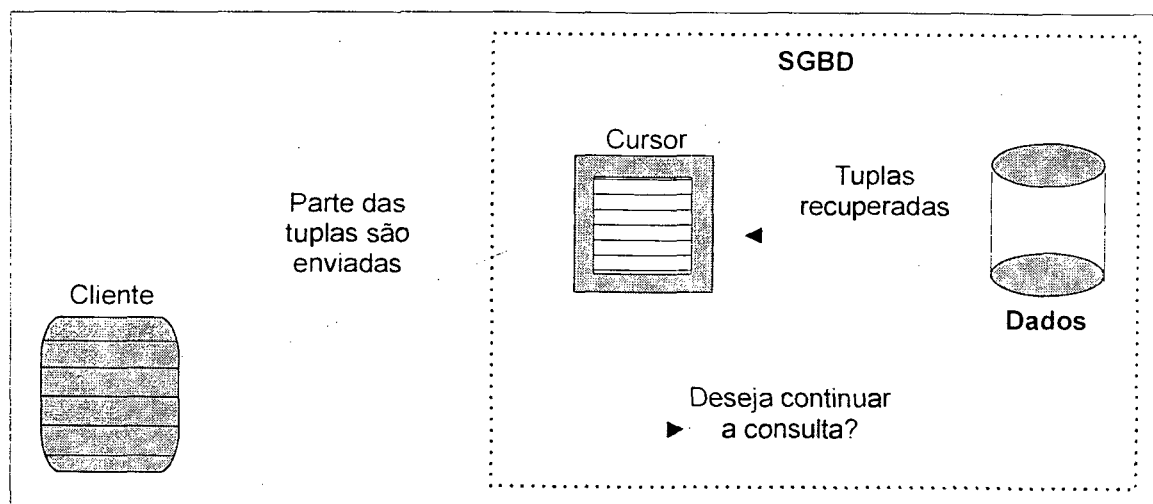
Além disso, duas importantes funcionalidades dos Bancos de Dados podem ser perdidas caso o *gateway* não mantenha uma sessão de usuário sob o SGBD: a otimização de consultas repetidas sob o Banco de Dados e a praticidade de cursores SQL para manipulação de dados armazenados no Banco de Dados.

A otimização de consultas refere-se aos mecanismos presentes nos atuais SGBDs que permitem um melhor gerenciamento do acesso aos dados armazenados (Korth & Silberwchatz, 1997).

Para resolver estes problemas os SGBDs relacionais proporcionam mecanismos de recuperação de dados através de cursores SQL. Um cursor é definido em relações a serem processadas pelas linguagens hospedeiras e funciona como um "ponteiro" para tuplas recuperadas pelo SGBD.

A figura 4.3 abaixo mostra o processo de execução de uma consulta usando-se um cursor SQL.

Figura 4.3 - Processo de execução de uma consulta via cursor.



O usuário submete a consulta mas somente um conjunto de dados é recuperado, geralmente através de uma chamada de busca (*fetch*), e enviado ao cliente. Assim, é possível não apenas controlar o número de tuplas

recuperadas, mas pode-se também periodicamente indagar ao usuário para determinar se ele deseja continuar ou não a consulta.

Toda a funcionalidade proporcionada pelo uso de cursores em SGBDs relacionais descrita acima só é possível se a conexão com o SGBD entre duas interações de um mesmo usuário não for finalizada.

Dessa forma, a solução mais adequada para a questão do desempenho sob o SGBD decorrente da característica *stateless* da *Web* é manter a sessão contínua do usuário sob o SGBD. Como o *gateway* é o responsável pela ligação entre o ambiente *Web* e o SGBD, deve caber a ele esta tarefa. Assim, preserva-se, no ambiente de integração, duas outras importantes funcionalidades dos SGBDs: otimização de consultas repetidas e a propriedade de cursores SQL para manipulação adequada dos dados armazenados.

4.2 Escopo de uma transação na *Web*

As transações gerenciadas pela maioria dos SGBDs existentes seguem o modelo de transações planas. O modelo é assim chamado porque existe uma única camada de controle pela aplicação: todas as operações entre os delimitadores de início (*begin transaction*) e término (*end transaction*) possuem o mesmo nível, e são aceitas ou rejeitadas em bloco (atomicidade). A expansão deste modelo para o ambiente de integração deve levar em consideração o escopo ou abrangência de uma transação *Web* Banco de Dados.

A primeira constatação é que uma transação no ambiente de integração deve ser baseada em páginas *Web*, já que elas são as unidades básicas de interação com o usuário.

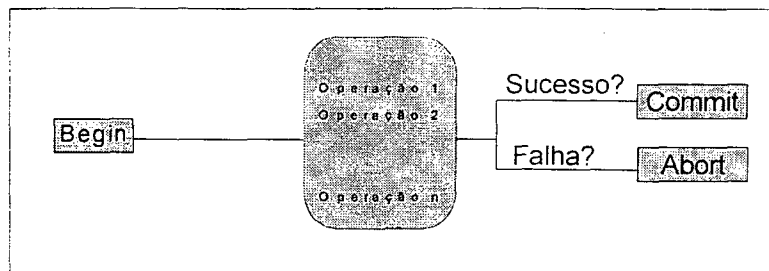
4.2.1 Uma página como Unidade atômica

A forma mais simples de uma transação *Web* é aquela em que uma página é composta por uma ou mais operações sob o SGBD. Uma vez que uma página *Web* deve ser considerada como uma unidade básica de operação, se a página contiver um conjunto de operações sob o SGBD.

Tendo em vista esta consideração, é importante restringir alguns mecanismos dos atuais SGBDs que possibilitam que cada operação no SGBD seja considerada uma unidade atômica, tais como a configuração de variáveis ambientes do SGBD que indica como ele deve proceder quando um conjunto de comandos for submetido através de uma interação do usuário.

A figura 4.4 ilustra uma página *Web* composta por várias operações a serem efetivadas pelo SGBD. Neste caso, as operações *Web* contidas na página são consideradas como uma unidade atômica de processamento para o SGBD.

Figura 4.4 : Transação composta por uma página *Web*.

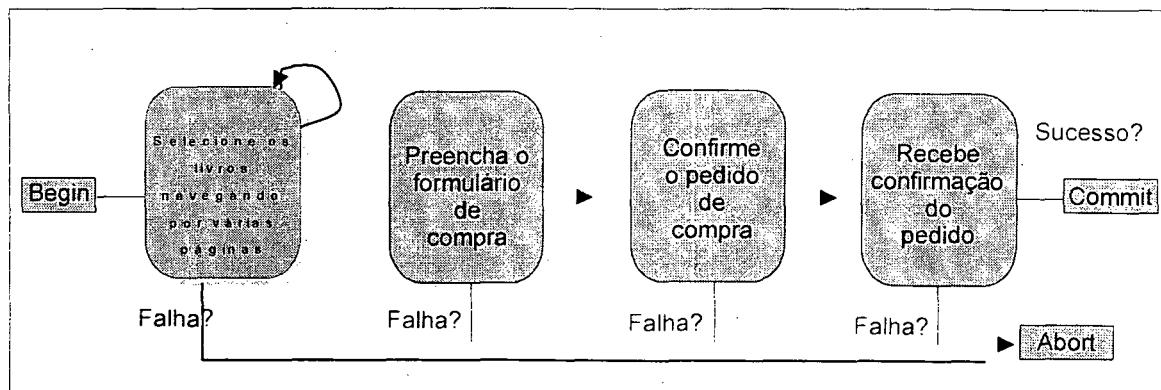


4.2.2 Várias páginas como unidade atômica

A limitação do escopo de uma transação *Web* Banco de Dados a uma única página *Web* é desejável para efeitos de desempenho, já que assim é minimizado, por exemplo, o tempo de duração de bloqueio aos dados no

SGBD. No entanto, desenvolvedores de aplicações freqüentemente precisam que várias interações com o usuário sejam realizadas como se fossem uma única transação atômica. Para que isso seja possível 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. Veja na figura 4.5 um exemplo dessa transação.

Figura 4.5 - Transação composta por várias páginas *Web*: compra de livros.



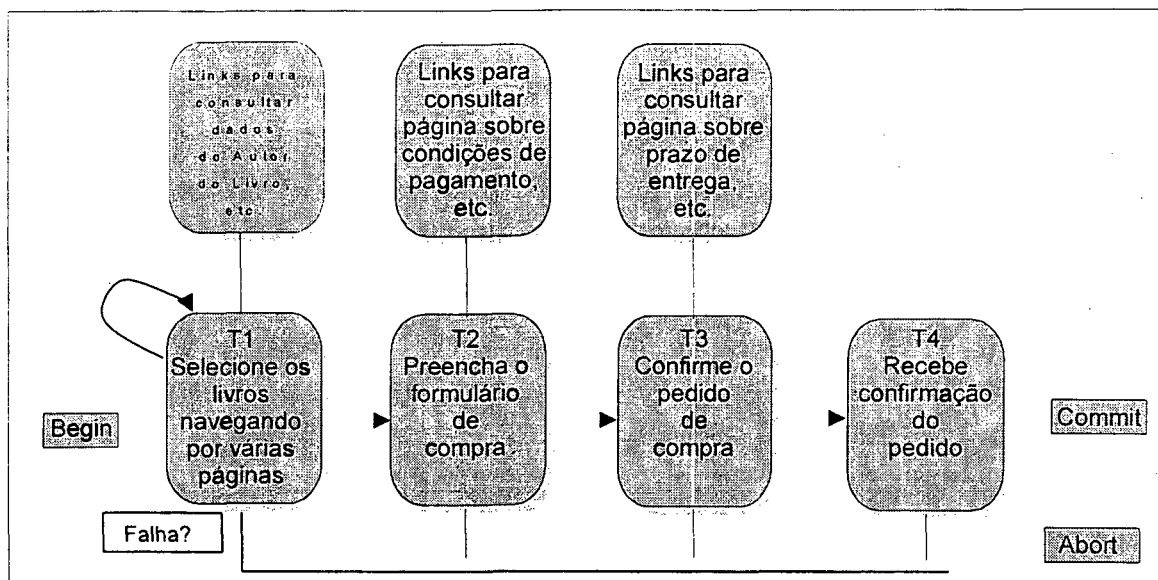
A figura 4.5 mostra uma seqüência de páginas *Web* que devem ser manipuladas no contexto de uma transação atômica. Primeiramente o usuário seleciona os livros navegando por diversas páginas *Web*, em seguida preenche os dados para compra dos livros, confirma o pedido de compra e recebe uma ordem de pagamento com o total a ser pago quando os livros forem entregues.

4.2.3 Páginas que não fazem parte de uma transação

A possibilidade de inclusão de várias páginas formando uma única transação atômica é útil para resolver o problema de transações que exijam várias interações com o usuário. No entanto, a solução de delimitar um conjunto de páginas para compor uma transação atômica pode ser muito restritiva no ambiente *Web*.

Veja na figura 4.6 um exemplo de uma, com estas características.

Figura 4.6: Transação composta por páginas transacionais e por páginas normais



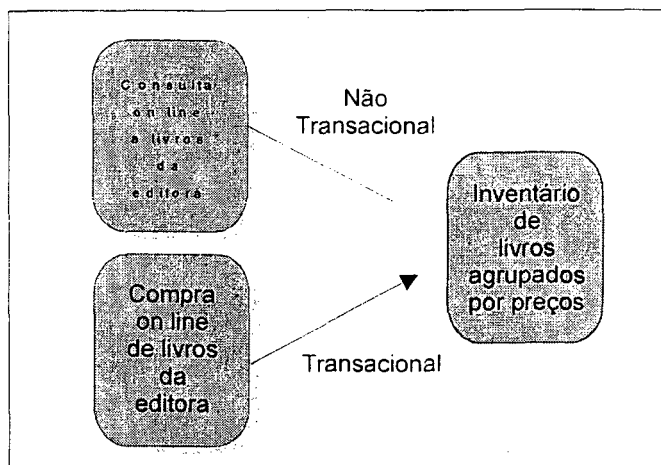
Neste exemplo, existem páginas que não fazem parte diretamente do contexto transacional sendo submetido pelo usuário, podendo inclusive serem páginas estáticas armazenadas no sistema de arquivos na plataforma do servidor *Web*. Isto nos permite identificar dois tipos de páginas no ambiente *Web*: aquelas que devem ser marcadas como transacionais e aquelas que não precisam ter estas características.

4.2.4 Páginas com dupla funcionalidade

A solução de contextualizar uma página *Web* como parte de uma transação ou não ainda pode ser restritiva no contexto navegacional da *Web*. De fato, existem situações na *Web* que nos permite identificar momentos em

que uma mesma página pode ora fazer parte de uma transação *Web Banco de Dados*, ora ser considerada como qualquer outra página do ambiente *Web*. Veja na figura 4.7 um exemplo de páginas com estas características:

Figura 4.7 - Páginas com dupla função



Este exemplo mostra páginas que ora fazem parte do contexto transacional *Web Banco de Dados* (compra *on line* de produtos por clientes cadastrados) e ora não precisam ter a conotação transacional (consultas a produtos *on line* por clientes não cadastrados). Isto permite dar uma semântica de dupla funcionalidade para páginas *Web*: sob certas ações do usuário cliente uma página deve compor uma transação atômica e sob outras a mesma página não deve fazer parte do contexto transacional.

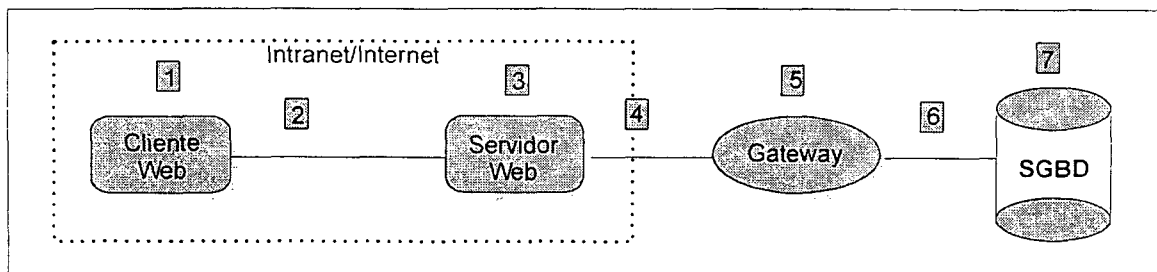
4.2.5 Recuperação

O objetivo desta seção é discutir como devem ser implementados mecanismos de recuperação no ambiente *Web Banco de Dados*. Para isso serão destacados os potenciais pontos de falha de uma transação *Web Banco*

de Dados e discutidas semânticas adequadas para quando ocorrer alguma falha num determinado ponto.

Uma transação entra num estado de falha depois de ser determinado que ela não pode mais prosseguir com sua execução normal. Por exemplo, uma transação pode falhar devido a erros de hardware em um dos componentes do sistema, erros na comunicação dos componentes do sistema (*links* perdidos), erros lógicos (dado não encontrado) e erros do sistema como na detecção de *deadlocks*¹ (Korth & Silberwchatz, 1997). Nesse caso, devem ser ativados mecanismos de recuperação do sistema evitando estados inconsistentes no SGBD. Nos atuais SGBDs isoladamente, este é um problema já bem estudado e resolvido. Uma típica arquitetura *Web* integrada a Banco de dados é ilustrada na figura 4.8 a seguir. Nela são destacados os possíveis pontos em que podem ocorrer uma falha durante a execução normal de uma transação.

Figura 4.8: Pontos de falhas no ambiente *Web* Banco de Dados.



Diante destas considerações iniciais a implementação do *gateway* deve proporcionar a consistência de aplicações no ambiente de integração considerando os sete pontos de falha assinalados na figura 4.8 anterior:

- a) **Falha na máquina cliente ou no *browser***, onde podem ocorrer falhas de hardware ou de software tornando a aplicação cliente inoperante por um determinado tempo. Neste caso o *gateway* deverá manter a sessão do usuário junto ao SGBD por um determinado tempo. Se a conexão com o

¹ Um *deadlock* é uma situação na qual duas ou mais transações estão em estado de espera simultânea, cada uma esperando que uma das outras libere um bloqueio antes de poder prosseguir.

cliente não se reiniciar, deve-se garantir que todas as ações da transação submetida pelo usuário sejam desfeitas (*rollback*).

- b) **Falha na rede entre o cliente *Web* e o servidor *Web***, onde *links* de comunicação podem ser perdidos ou ocorrer falhas na comunicação entre o cliente *Web* e o servidor *Web*. Neste caso, o *gateway* não receberá qualquer pedido do cliente e novamente, depois de um determinado tempo, um *rollback* deve ser acionado. O cliente saberá que não conseguirá submeter nenhuma outra ação e que, se após um tempo a conexão não for restabelecida a transação ativa será desfeita pelo *gateway*.
- c) **Falha no servidor *Web***. Um *rollback* será realizado pelo *gateway* se o servidor não conseguir se restabelecer após um determinado tempo. O cliente novamente fica esperando por resposta do servidor, mas se isso não ocorrer num determinado tempo ele deverá resubmeter a transação que foi desfeita pelo *gateway*. Os servidores *Web* têm capacidade para atender um número limitado de pedidos de usuários simultâneos que depende, entre outros, do projeto do servidor *Web* e da capacidade de processamento da máquina onde executa o servidor *Web*.
- d) **Falha na conexão entre o servidor *Web* e o *gateway***: Análogo à falha na rede.
- e) **Falha no *gateway***. Neste caso ocorreu um erro no coordenador da aplicação. Nenhuma transação ativa ficará disponível para o cliente de forma que a responsabilidade para garantir que a transação se realize consistentemente deve ser deixada ao SGBD. Novamente, o SGBD realizará um *rollback* da transação em andamento depois de um determinado tempo (*timeout* do SGBD).
- f) **Falha na conexão entre o *gateway* e o SGBD**. Neste caso cabe também ao SGBD garantir o estado consistente da transação em andamento. O

problema é saber se a transação foi finalizada com sucesso ou foi abortada. Assim, o usuário deve ser informado do estado de sua transação quando a conexão for restabelecida. Cabe ao *gateway* proporcionar mecanismos neste sentido fazendo uma verificação no SGBD para ver se a transação foi ou não completada. Atualmente esse tipo de mecanismo não é garantido por nenhum dos *gateways* levantados neste trabalho (vide capítulo seguinte).

- g) **Falha no SGBD. Do ponto de vista do *gateway*** este tipo de falha é análogo ao anterior. Novamente o SGBD garante mecanismos de consistência e recuperação de transações submetidas quando o SGBD for restabelecido.

Desde o momento que se inicia uma transação no cliente até seu fim, uma transação *Web Banco de Dados* pode estar em três possíveis estados. São eles:

- a) **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.
- b) **Transação em processo:** o usuário recebeu a confirmação de que uma transação se iniciou e está em execução e portanto, uma sessão para ele será mantida pelo *gateway* junto ao SGBD. Nenhuma confirmação de fim de transação (*commit*) foi solicitada.
- c) **Transação finalizada mas não confirmada:** o usuário decidiu confirmar o fim da transação (enviou um *commit*) 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.

De acordo com estes três estados, se ocorrer uma das sete falhas enumeradas anteriormente, deve-se implementar uma solução para que seja ativado o processo de recuperação da transação. São enumeradas abaixo, de acordo com as falhas listadas, as semânticas consideradas mais adequadas para serem implementadas pelo *gateway*:

1. Falha no Cliente:

- a) Transação iniciada mas não confirmada: neste caso ou a transação não se iniciou de fato, o que implica que não há nada para ser feito, ou a transação se iniciou e o cliente não pode continuar, de forma que a sessão deve ser automaticamente finalizada pelo *gateway* após o *timeout*.
- b) Transação em processo: aqui o cliente não poderá continuar e toda a transação deve ser desfeita (*rollback*) após o *timeout*.
- c) Transação finalizada mas não confirmada: neste caso uma ação de *commit* pode ter ocorrido ou não. O Banco de Dados deve ser consultado mais tarde para que seja determinado o que ocorreu com a transação. Em particular, se um *commit* foi enviado pelo cliente e nenhuma confirmação deste *commit* foi recebida, o usuário cliente deve ter algum mecanismo para saber posteriormente se a transação foi ou não compromissada.

2. Falha na Rede:

- a) Transação iniciada mas não confirmada: como o cliente não recebeu a confirmação de sessão iniciada ele não poderá se reconectar caso a falha na rede seja temporária, mesmo que a sessão tenha se iniciado com sucesso. É uma situação análoga ao caso 1 A.
- b) Transação em processo: se a falha na rede for temporária o cliente poderá se reconectar e continuar a transação. Se ocorreu um *timeout* então a sessão será finalizada e toda a transação ativa desfeita.
- c) Transação finalizada mas não confirmada: neste caso o cliente só terá condições de SABER se a transação foi compromissada ou não após o

restabelecimento da falha, por meio de uma consulta ao Banco de Dados.
Situação análoga ao caso 1 C.

3. Falha no servidor *Web*:

- a) Transação iniciada mas não confirmada: situação análoga ao caso 2 A.
- b) Transação em processo: situação análoga ao caso 2 B.
- c) Transação finalizada mas não confirmada: situação análoga ao caso 1 C.

4. Falha na comunicação servidor *Web* e *gateway*:

- a) Transação iniciada mas não confirmada: situação análoga ao caso 2 A.
- b) Transação em processo: situação análoga ao caso 2 B.
- c) Transação finalizada mas não confirmada: situação análoga ao caso 1 C.

5. Falha no *gateway*:

- a) Transação iniciada mas não confirmada: a sessão pode ou não ter sido iniciada no SGBD. Em qualquer dos casos, o cliente não poderá continuar e um *timeout* finalizará a sessão se ela foi iniciada.
- b) Transação em processo: com a falha no *gateway*, nenhuma transação posterior será aceita. Cabe ao SGBD retornar a transação para um estado consistente.
- c) Transação finalizada mas não confirmada: situação análoga a 1 C.

6. Falha na comunicação entre o *gateway* e o SGBD:

- a) Transação iniciada mas não confirmada: o SGBD pode ou não ter iniciado uma sessão. Situação análoga ao caso 5 A.
- b) Transação em processo: situação análoga ao caso 5 B. Toda o processo de recuperação é deixado para ser realizado pelo SGBD.

c) Transação finalizada mas não confirmada: situação análoga a 1 C.

7. Falha no SGBD:

- a) Transação iniciada mas não confirmada: neste caso, nenhuma operação foi compromissada e portanto a aplicação deve ser reiniciada. O SGBD deve retornar o erro apropriado quando possível.
- b) Transação em processo: todas as operações não compromissadas pelo SGBD serão desfeitas. A transação deve ser reiniciada.
- c) Transação finalizada mas não confirmada: análogo ao 1 C.

Observe que o caso 1 C se repete em todos os demais. Este caso não permite determinar se a transação submetida pelo usuário foi tornada persistente ou não no Banco de Dados. Somente depois que todos os componentes forem restabelecidos se terá condições de determinar se as transações pendentes se tornaram ou não persistentes. Neste caso, seria adequado que o *gateway* tivesse algum mecanismo para que o usuário cliente soubesse o que aconteceu com sua transação.

O problema citado no parágrafo anterior não é, na realidade, um problema particular só do ambiente de integração SGBD e *Web*. O objetivo é o de garantir que o SGBD estará num estado consistente mesmo na presença dos mais diversos tipos de falhas.

4.2.6 Consistência

Uma transação deve deixar o Banco de Dados num estado consistente ao final de sua execução. 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.

No entanto, a orientação a páginas de dados do ambiente *Web* em detrimento à orientação a conjunto de tuplas dos SGBDs, pode originar sérios problemas de consistência das transações *Web Banco de Dados*. A seguir são apresentados alguns exemplos que ilustram o problema e suas implicações na questão da consistência das transações:

- a) **Inserção de tupla:** um exemplo típico é a inserção de tuplas no Banco de Dados. Após o usuário inserir uma tupla, geralmente o *gateway* retorna uma página informando que a operação foi completada com sucesso. Se o usuário usar a opção *back* do seu *browser* ele poderá re-submeter a mesma operação de inclusão com os mesmos dados.
- b) **Exclusão de tupla:** a exclusão de tuplas é outro exemplo. Se o usuário excluir uma tupla por meio de uma interação e voltar na página da operação novamente re-submetendo-a, então a tupla não mais existirá. Um erro de falha da transação vai ser retornado pelo SGBD, o que é indesejável.
- c) **Atualização de tupla:** podem existir situações mais complexas que implicam, por exemplo, em atualizações em várias tuplas cujos danos podem ser difíceis de serem revertidos futuramente.
- d) **Páginas inconsistentes:** uma vez recuperado um documento *Web* o cliente *Web* usa o cache local para armazená-lo de forma que o usuário posteriormente possa consultá-lo novamente sem que para isso seja necessário uma nova conexão com o servidor *Web*. Esta característica *Web* permite melhor desempenho do sistema. No entanto, cria um problema potencial para aplicações cujas páginas de dados são geradas dinamicamente como nas aplicações *Web Banco de Dados*.

A primeira tentativa para solução do problema seria desabilitar as opções *back/forward* do *browser*. Isto, de fato, é possível em alguns deles, mas para isso seria necessário retirar todas as opções presentes nos *browsers*

impedindo a navegação hipertexto normal e a maioria das outras funcionalidades dos *browsers*.

Uma boa solução está no *gateway* que integra as tecnologias. O *gateway* pode garantir que uma transação não será re-submetida ao SGBD. Mais ainda, o *gateway* pode controlar as interações do usuário de forma a garantir que se consiga determinar onde o usuário atual está na aplicação e para que ponto ele pode prosseguir. Para isso, novamente, o *gateway* deve gerenciar o estado da aplicação, ficando ativo, no mínimo, até que a aplicação seja finalizada.

Um 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. Isto é possível, devido à opção *stop* presente nos *browsers* atuais.

Para se ter uma solução compatível com a semântica *stop* dos *browsers* deve-se implementar algum mecanismo no *gateway* que permita identificar este tipo de ação do usuário e, uma vez identificado, ser de responsabilidade do *gateway* retornar o Banco de Dados a um estado anterior ao da transação submetida. Segundo a pesquisa realizada nessa dissertação, não foi encontrado nenhum *gateway* que implementasse este tipo de mecanismo. Este é um problema em aberto no ambiente de integração de difícil solução.

4.2.7 Controle de Concorrência e Isolamento

O objetivo desta seção é discutir como as técnicas de controle de concorrência aos dados implementadas pelos atuais SGBDs podem afetar o ambiente de integração. Será analisada, em particular, a técnica de bloqueio (*lock*) e cuidados que se deve ter relativo ao nível de granularidade do bloqueio a ser configurado no SGBD. Novamente, procurar-se-á analisar o problema de forma a não modificar a natureza das tecnologias envolvidas.

Como já destacado, o isolamento de uma transação implica que cada transação deve ver um estado consistente do Banco de Dados sempre que for executada concorrentemente. Isto requer que os resultados intermediários de uma transação não sejam observáveis por outras transações. Para garantir esta propriedade os SGBDs são dotados de técnicas que garantem a seriabilidade das transações concorrentes, questão já amplamente difundida na literatura (Gray & Reuter, 1993 E Elmasri & Navathe, 1994). O problema aparece quando várias páginas originam uma transação atômica. Novamente, a discussão inicial baseia-se nas duas alternativas para se implementar o *gateway*: ou ele mantém uma sessão contínua de usuário sob o SGBD enquanto durar a aplicação ou ele não mantém.

O problema pode ser reduzido se aplicações *Web* Banco de Dados levarem em consideração os seguintes pontos:

- A granularidade (Korth & Silberwchatz, 1997) dos itens de dados no SGBD para técnicas de concorrência baseadas em bloqueio deve ser muito bem configurada. Níveis de granularidade para tabelas inteiras ou mesmo para páginas de dados podem ser inadmissíveis no ambiente de integração
- A configuração do valor do *timeout* que o SGBD deve esperar para liberar os bloqueios de uma transação no ambiente de integração deve ser um valor que leve em consideração a aplicação (Internet/Intranet), o tráfego na rede, a sobrecarga no servidor *Web* e a capacidade de processamento do *gateway* integrador.
- O uso, sempre que possível, de cursores, que podem ser úteis para aumentar a concorrência aos dados.

Além disso, deve-se também garantir que a navegação via opções *back/forward* dos *browsers* não gerem situações indesejáveis do ponto de vista do controle de concorrência. Por exemplo, situações em que um usuário submete uma página que solicite um bloqueio a um item de dado e, logo em

seguida, o usuário re-submete a mesma página via opção *back* do *browser*. Como já ressaltado na sessão anterior devem existir mecanismos no *gateway* que lidem com este tipo de ação por parte do usuário.

4.3 Conclusão

Este capítulo abordou diversas funcionalidades desejáveis no ambiente de execução de transações *Web* BD, agrupadas sob dois grandes tópicos: a característica *stateless* da *Web* e a extensão das propriedades ACID para o ambiente de integração.

No que diz respeito à característica *stateless* da *Web* analisou-se, em particular, o problema do gerenciamento de estado da aplicação bem como as soluções existentes.

5 ARQUITETURA

Nos capítulos anteriores procurou-se destacar os aspectos funcionais do ambiente de integração, apontando os problemas mais significativos dentro do contexto de sistemas de Banco de Dados. Este capítulo discute o ambiente de integração para a execução das transações sob a abordagem das arquiteturas dos *gateways Web Banco de Dados* existentes, enfatizando-se as principais funcionalidades descritas nos capítulos precedentes para cada arquitetura em particular.

As transações no ambiente SGBD e *Web* já é hoje uma realidade dando suporte aos mais diversos tipos de aplicações, comerciais ou não. Existem centenas de *softwares* ou *gateways* de integração Web e Banco de Dados (Plain, 1996) que visam tirar proveito do ambiente aberto e multi-plataforma da Web e das facilidades de gerenciamento e mecanismos de otimização para acesso a dados presentes nos SGBDs.

Um dos objetivos deste capítulo é apresentar uma classificação das arquiteturas destes *gateways*, que são os responsáveis pela administração das transações.

Pretende-se analisar também como os aspectos funcionais dos atuais SGBDs são afetados pela arquitetura do *gateway*, como por exemplo, controle de gerenciamento do estado da aplicação, propriedades ACID das transações Web Banco de Dados, controle de restrições de integridade, segurança, desempenho e portabilidade. Pretende-se ainda, mencionar os principais *gateways* disponíveis no mercado para cada arquitetura descrita, a título de exemplificação.

São poucos os trabalhos que tratam da classificação das arquiteturas dos *gateways* Web Banco de Dados existentes. Em (Perrochon, 1995), (Spider, 1997), (Kim, 1997) e (Lifschitz & Lima, 1998), são propostas três classificações das arquiteturas dos *gateways*, Web Banco de dados. A Taxonomia das arquiteturas apresentadas a seguir parte dos estudos relacionados em (Perrochon, 1995), (Kim, 1997) e (Lifschitz & Lima, 1998), estendidos de forma a englobar as novas arquiteturas que surgiram recentemente, decorrentes do amadurecimento e da rápida evolução da integração das tecnologias.

Este capítulo está organizado da seguinte forma. A seção 5.1 categoriza os *gateways* que podem ser desenvolvidos segundo a arquitetura de transações que implementam. Nas seções 5.1.1, 5.1.2 e 5.1.4 é apresentado o funcionamento de cada arquitetura em particular, divididos segundo a taxonomia proposta na seção 5.1. São catalogados oito arquiteturas de transações distintas para integração Web e Banco de Dados, com destaque para vantagens e desvantagens de cada uma delas. Na seção 5.2 é mostrado um quadro comparativo das arquiteturas catalogadas baseadas nas funcionalidades de Banco de Dados discutidas ao longo dos capítulos anteriores. Por fim, a seção 5.3 é finalizada com uma breve revisão do capítulo e conclusão final.

5.1 Taxonomia das arquiteturas de transações Web SGBD

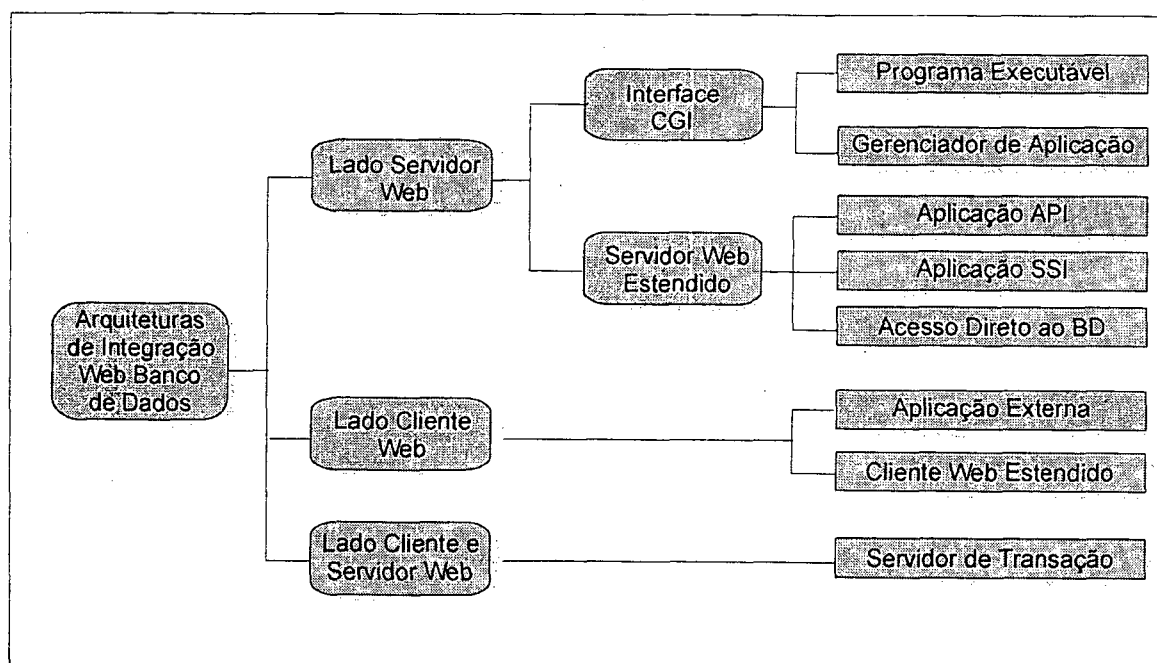
Os *gateways* de integração Web e Banco de Dados, também chamados *middlewares*, se referem à extensa classe de softwares que ocupam uma posição intermediária na arquitetura de integração dessas tecnologias. Como analisado no capítulo 2, quando um *gateway* é ativado sua principal função é executar, junto ao SGBD, o pedido que lhe foi solicitado e devolver ao servidor Web o resultado do pedido em um formato especificado, como por exemplo, em formato HTML.

Estes *gateways* podem ser classificados em categorias segundo a arquitetura de integração que implementam. Esta classificação ou taxonomia pode ser baseada em vários critérios. Em (Spider, 1997b), por exemplo, o critério utilizado baseia-se na arquitetura cliente/servidor da aplicação que acessa o Banco de Dados: a arquitetura de duas ou de três camadas. Em (Perrochon, 1995) o critério utilizado baseia-se na forma como a aplicação que acessa o Banco de Dados interage com o cliente *Web*: se por meio de interface padrão, de interface proprietária ou por meio de sistemas de código móvel (Java, por exemplo).

O critério utilizado para a classificação das arquiteturas dos *gateways* *Web* Banco de Dados geralmente é influenciado pelos objetivos a que se propõem o trabalho da classificação. Neste sentido, utilizou-se aqui uma classificação das arquiteturas existentes baseadas na localidade em que o *software* integrador é executado: se no lado cliente ou no lado servidor *Web*. Esta classificação é semelhante à utilizada por Kim (1996) e por (Lifschitz & Lima, 1998) e é útil, primeiro, por ser mais abrangente que as demais e segundo, porque permite estudar melhor como as funcionalidades dos atuais SGBDs são afetadas pela arquitetura de integração proposta.

A figura 5.1 mostra um diagrama da taxonomia das arquiteturas dos *gateways* *Web* Banco de Dados utilizada aqui.

Figura 5.1: Taxonomia das arquiteturas dos *gateways* *Web* BD.



Os *softwares* de integração *Web* e Banco de Dados são primeiramente enquadrados em três categorias: *softwares* que executam no lado servidor *Web*, *softwares* que executam no lado cliente *Web* e *softwares* que executam no lado cliente e servidor *Web*. Os *softwares* que executam no lado servidor *Web* podem ser classificados em duas categorias: aqueles que usam a interface CGI (*Common Gateway Interface*) padrão e aqueles que usam um servidor *Web* estendido de forma a suportar características não previstas inicialmente pelo servidor *Web* padrão. Quando o *software* integrador usar a interface CGI pode-se ainda separá-lo nas categorias Programas Executáveis, onde o software é composto por um ou mais programas executáveis 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 *software* integrador usar um servidor *Web* estendido pode-se ainda separá-lo em três categorias: ele é uma Aplicação API do servidor *Web*, é uma Aplicação SSI, ou o servidor *Web* é projetado para realizar o Acesso Direto ao Banco de Dados. Os *softwares* que executam no lado cliente *Web* podem ser incluídos na categoria Aplicação Externa ao cliente *Web*, que se conecta diretamente ao Banco de Dados ou na categoria Cliente *Web* Estendido. Por fim, o software que executa no lado cliente e servidor *Web* é incluído na categoria Servidor de Transação. Todas essas arquiteturas são descritas com mais detalhes nas seções seguintes.

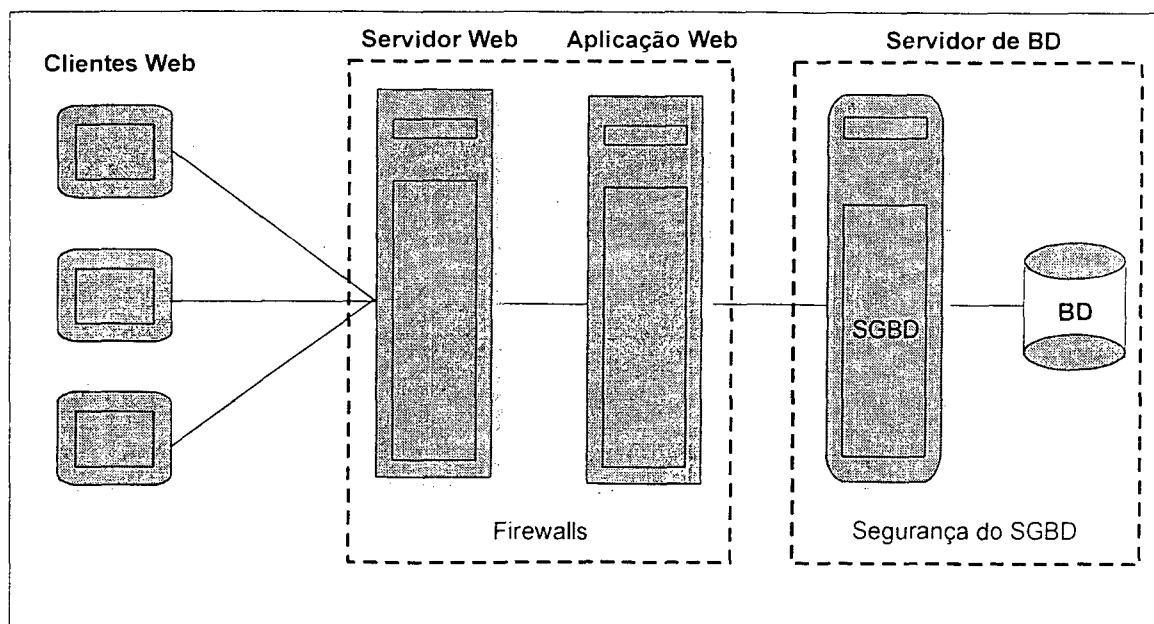
5.1.1 Arquiteturas de transações do lado servidor Web

As arquiteturas de integração *Web* e Banco de Dados localizadas no lado servidor *Web* tem como principal característica o fato de serem semelhantes à arquitetura cliente/servidor de Banco de Dados em três camadas. As três camadas são:

- **Cliente Web:** Onde reside a lógica da apresentação, que controla a interação entre o usuário e o computador. Nesta camada pode-se ter alguma lógica da aplicação, tais como validação de dados implementados por ferramentas proprietárias como Java, *JavaScript* ou *VBScript*.
- **Servidor Web/Aplicação cliente do Banco de Dados:** Esta camada é composta pelo servidor *Web* e a aplicação propriamente dita que acessa o Banco de Dados. Ela contém a lógica da aplicação, responsável pelas decisões, cálculos e operações que a aplicação deve realizar, mecanismos de segurança para acesso à aplicação, e controle de acesso multiusuário à aplicação
- **Servidor de Banco de Dados:** Esta camada consiste do servidor de Banco de Dados, tipicamente rodando em máquinas de alto desempenho. Ela contém todos os dados, metadados e regras para integridade referencial dos dados definidos pelo desenvolvedor da aplicação.

A seguir é apresentado um esboço genérico das arquiteturas localizadas no lado servidor *Web*, segundo as três camadas descritas anteriormente.

Figura 5.2: Arquitetura dos *gateways* que executam no lado servidor *Web*.



É importante observar que todo o código da aplicação reside num único local, de forma que toda a manutenção no código da aplicação se reflete imediatamente a todos os clientes *Web*.

A seguir são apresentadas as arquiteturas que usam a interface CGI padrão dos servidores *Web* para execução da aplicação e logo após são descritos as arquiteturas que usam servidores *Web* estendidos .

5.1.1.1 Interface CGI

Como descrito no capítulo sobre a interface CGI possibilita a execução de aplicações externas ao servidor *Web*. Assim, a interface CGI é a maneira mais natural para o desenvolvimento de aplicações *Web* Banco de Dados, já que são aplicações externas ao ambiente cliente/servidor *Web*. Apesar da implementação da interface CGI diferir em alguns servidores *Web*, as arquiteturas dos *gateways Web* Banco de Dados que usam a interface CGI são consideradas os mais portáveis que existem atualmente.

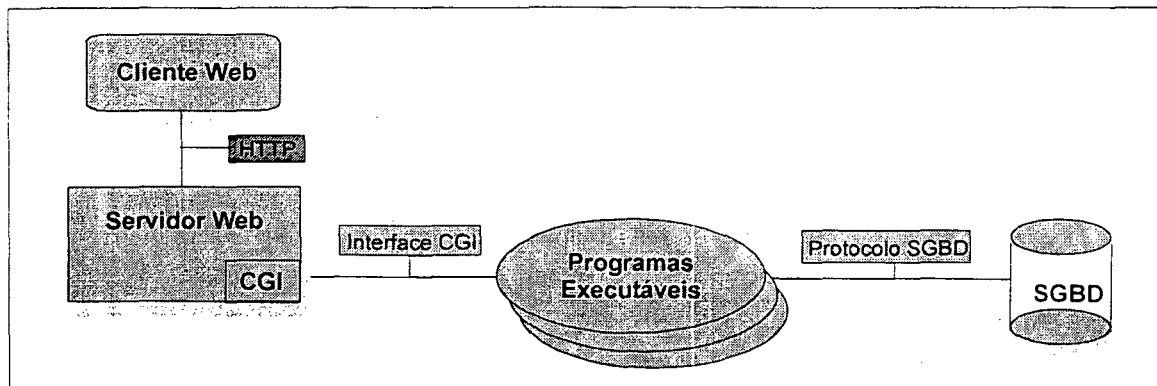
a) Programas Executáveis CGI

Esta arquitetura é composta por um ou mais programas CGI, geralmente implementados usando-se alguma linguagem de programação hospedeira do SGBD. Esta é a arquitetura mais imediata para se desenvolver aplicações *Web* Banco de Dados.

De fato, para se usar a interface CGI implementada pelos servidores *Web* é necessário uma linguagem que possa ler da entrada padrão, obter as variáveis de ambientes e escrever na saída padrão. Como a grande maioria dos SGBDs existentes incorporam alguma linguagem de programação com

estas características, praticamente todos eles podem se comunicar com a *Web* por meio de Programas Executáveis CGI sem grandes custos adicionais. A figura abaixo ilustra esta arquitetura.

Figura 5.3: Arquitetura Programas Executáveis CGI.



O funcionamento da arquitetura é mostrado a seguir:

1. O cliente *Web* solicita ao servidor *Web* a execução de uma aplicação externa, que neste caso é um programa executável na plataforma do servidor *Web*.
2. O servidor *Web* dispara um processo para execução do programa através da interface padrão CGI, enviando os dados recebidos do cliente *Web* de acordo com a implementação do servidor *Web* para a interface CGI.
3. O programa recebe os dados passados pelo servidor *Web* via interface CGI, decodifica-os, formula o comando SQL e abre uma conexão com o Banco de Dados para sua execução.
4. SGBD retorna os dados e a conexão é finalizada. O programa executável trata os dados recebidos e os repassa ao servidor *Web* via interface CGI, num formato que o cliente *Web* entenda, como por exemplo, no formato HTML. O processo iniciado pelo servidor *Web* para execução do programa é finalizado neste momento.

5. O servidor *Web* retorna os dados repassados pelo programa ao cliente *Web*.

A maioria dos *gateways* existentes hoje em dia estão implementados segundo esta arquitetura. São exemplos os *gateways* *DB2WWW Connection* (Nguyen & Srinivasan, 1996), *IBM97*, *ColdFusion* (Allaire, 1997), (Craig, 1996, Olympia, 1996 e Telford 1997), *NetImpact Dynamo* (Sybase, 1997C), *Web.(Sql Sybase*, 1997), *LiveWire* (Netscape, 1997J), *DataRamp* (Dataramp, 1997), *DBWeb* (Maia & Pinheiro, 1997) e *QSF* (Hadjefthymiades & Martakos, 1997).

A seguir uma análise mais detalhada da arquitetura Programas Executáveis CGI.

- **Aspectos transacionais:** A arquitetura possibilita pouco controle das transações. De fato, como a conexão com o Banco de Dados é finalizada após a execução de cada programa e o processo do programa se encerra após o envio de dados ao servidor *Web*, os mecanismos de gerenciamento de estado da aplicação do usuário. O controle de concorrência e demais mecanismos para garantir as propriedades ACID são gerenciados pelo próprio SGBD a partir do momento em que o programa CGI solicita um pedido ao SGBD. Ou seja, não há nenhum controle transacional fora do ambiente SGBD.
- **Segurança/Acesso:** A implementação de segurança nesta arquitetura é muito limitada. De fato, o acesso ao Banco de dados fica restrito aos mecanismos de acesso tradicionais dos SGBDs. Além disso, como é o servidor *Web* que solicita a execução dos programas CGI, o acesso ao Banco de Dados é realizado na maioria das vezes, sob a conta do servidor *Web* e não do usuário cliente em particular. Ou seja, o SGBD reconhece como usuário o servidor *Web* e portanto os *grants* aos objetos de dados devem ser dados ao servidor *Web*, o que pode ser perigoso como analisado no capítulo 3.

- **Desempenho:** A arquitetura é muito ruim sob o ponto de vista do desempenho. Isto se deve ao fato de que, para cada pedido de um cliente *Web* um novo processo da aplicação é disparado em separado pelo servidor *Web*. Não há compartilhamento de recursos, como por exemplo, espaço em memória. Para uma aplicação *Web* que envolva um grande número de processos simultâneos indo ao Banco de Dados, pode-se degradar em muito o desempenho da aplicação.
- **Desenvolvimento:** O ambiente de desenvolvimento não é considerado ideal já que nesta arquitetura os programas da aplicação tornam-se menos legíveis e mais difíceis de serem depurados. Juntamente com complexas estruturas de dados e lógicas de programação deve-se intercalar textos da linguagem HTML no programa a fim de formatar as saídas dos dados. Pode-se inclusive intercalar textos das linguagens *JavaScript* ou *VBScript*, usadas, por exemplo, para consistência dos campos de entrada de dados em formulários HTML da aplicação e que são tratados no cliente *Web*.
- **Portabilidade:** A arquitetura é baseada na interface padrão CGI e portanto é suportada por todos os servidores *Web* que utilizem a especificação CGI padrão. Esta é a principal vantagem da arquitetura.

Existem formas diferenciadas de se implementar a arquitetura. Elas se diferenciam na maneira como os comandos SQL são gerados. Assim, o comando SQL pode estar explicitamente gerado dentro da linguagem de programação hospedeira que se está usando, pode estar dentro da página HTML da aplicação, ou pode estar implícito dentro de uma linguagem de macro.

Estes três variantes são detalhadas a seguir:

1. SQL dentro da linguagem de programação genérica.

Neste caso os executáveis CGI incorporam dentro de si o código do comando SQL, tipicamente usando-se a interface proprietária para programação do SGBD, como uma linguagem hospedeira do SGBD. Um dos problemas desta forma de implementação da arquitetura é que é necessário criar um executável para gerar cada página de interação com o usuário. Isto implica em problemas de manutenção do código. Qualquer mudança na aplicação implica em compilar o código correspondente novamente.

2. Extensão de *tags* HTML para suportar comandos SQL .

Uma segunda variante da arquitetura é desenvolver um aplicativo CGI genérico capaz de interpretar comandos SQL inseridos em páginas HTML.

Esta proposta é bem mais flexível que a anterior pois pode-se mudar características da interface e a consulta ao Banco de Dados sem ter que mudar o código do aplicativo. Elimina-se assim a necessidade de reestruturar e recompilar o código toda vez que for necessário alterar características da interface ou mudança na consulta SQL. Além disso, a aplicação do usuário é desenvolvida segundo páginas HTML, que serão interpretadas pelo aplicativo CGI genérico, permitindo que os próprios usuários desenvolvam a aplicação mais eficiente sem que se tenha necessariamente de entender qualquer mecanismo da interface CGI.

Esta proposta tipicamente estende as *tags* HTML em dois tipos: *tags* para acesso a um SGBD através de comandos SQL e *tags* para construção de controles da lógica da aplicação tal como interação entre variáveis e estruturas de *loops*.

3. SQL implícito em linguagem de macro.

Uma terceira maneira de se implementar a arquitetura é desenvolver um aplicativo CGI capaz de interpretar uma linguagem de macro. Trata-se de uma linguagem pré-definida contendo diretivas/comandos que serão interpretados pelo aplicativo CGI. A proposta é bem parecida com a anterior. A diferença é que ao invés de se estender tags HTML para suportar sentenças SQL, os

comandos para gerar as sentenças SQL ficam residentes em macros, que podem estar armazenadas em arquivos convencionais ou podem ser objetos do Banco de Dados.

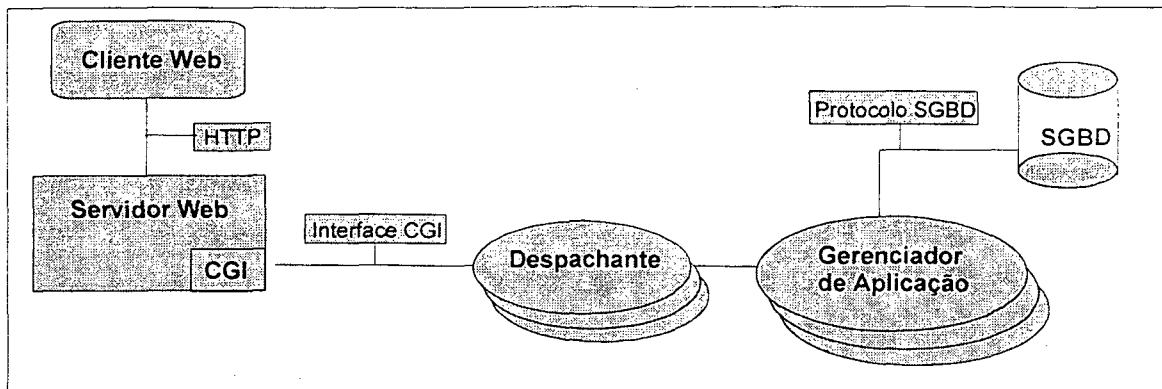
Esta forma de se implementar a arquitetura tem as seguintes vantagens:

- Requer pouco esforço para desenvolvimento de aplicações *Web* Banco de Dados, acarretando rápido desenvolvimento.
- É suficientemente flexível para uma variedade de aplicações *Web* que não requer extensiva programação lógica.
- É facilmente portátil para várias plataformas. Uma macro escrita numa plataforma funciona em qualquer outra plataforma. Basta reescrever o aplicativo CGI genérico na plataforma específica.
- Qualquer mudança no esquema do Banco de Dados não implica em mudança no código do CGI. Basta a mudança nas macros correspondentes o que pode ser bem mais simples.

b) Gerenciador de Aplicação CGI

De forma a suplantiar os diversos problemas da arquitetura CGI anterior como a dificuldade de gerenciar eficientemente o estado da aplicação, segurança no acesso ao Banco de Dados e otimização de consultas, pode-se implementar uma arquitetura de integração *Web* Banco de Dados dividido em dois módulos: vários despachantes (*dispatchers*), que são pequenos programas executáveis, e (um ou mais) gerenciador que coordena a aplicação. Esta arquitetura dá origem à arquitetura de integração denominada Gerenciador de Aplicação CGI. A figura a seguir ilustra esta arquitetura.

Figura 5.4: Arquitetura Gerenciador de Aplicação CGI.



O funcionamento da arquitetura é mostrado a seguir:

1. O cliente *Web* solicita ao servidor *Web* a execução de um aplicativo externo que neste caso é um Gerenciador de Aplicação CGI.
2. O servidor *Web* inicia um processo para um dos despachantes 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 a interface CGI.
3. O despachante identifica qual Gerenciador de Aplicação pode processar o pedido do servidor *Web*, transfere em seguida os dados ao gerenciador escolhido e fica esperando pela resposta.
4. O Gerenciador de Aplicação escolhido começa a executar sob o SGBD o pedido solicitado pelo despachante e, ao contrário da arquitetura Programas Executáveis CGI, não finaliza a conexão com o Banco de Dados. O Gerenciador de Aplicação fica esperando por novos pedidos de despachantes relativos ao usuário cliente que está utilizando a aplicação.
5. O despachante recebe os dados do Gerenciador de Aplicação e repassa-os ao servidor *Web* via interface CGI. O processo iniciado pelo servidor *Web* para a execução do despachante é finalizado neste momento.
6. O servidor *Web* retorna os dados ao cliente *Web*.

Um despachante CGI pode ser implementado sem nenhum conhecimento do SGBD. Analogamente, os Gerenciadores de Aplicação são implementados sem conhecimento da interface CGI. Como os Gerenciadores de Aplicação mantêm a conexão com o Banco de Dados aberta após a execução de algum pedido do despachante, pode-se tomar vantagens integrais do esforço de otimização do SGBD.

São exemplos de *gateways* que se enquadram nesta arquitetura os softwares *Web Connect* (Informix, 1997), *O2Web* (O2, 1997), *NetDynamics* (Plain, 1996), (Spider, 1997b) e *WebObjects* (Plain, 1996 e Next, 1997).

A seguir uma análise mais detalhada da arquitetura Gerenciador de Aplicação CGI.

- **Aspectos transacionais:** O estado da transação do usuário pode ser melhor gerenciado nesta arquitetura e dependendo das características da aplicação, pode-se implementar as soluções para os problemas técnicos descritos no capítulo anterior, como manter o estado, a proteção do estado e evitar manter estado de transação desnecessário. A manutenção da conexão com o Banco de Dados pelo Gerenciador de Aplicação permite que mecanismos de controle de concorrência presentes nos atuais SGBDs sejam viáveis no ambiente *Web*.
- **Segurança/Acesso:** Pode-se implementar um rígido controle de segurança da aplicação. Os vários mecanismos que podem ser usados incluem:
 - 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 controle do fluxo da navegação pode ser gerenciado pelo Gerenciador de Aplicação
 - O desenvolvedor define o ponto de início da aplicação e todo o caminho que o usuário deve seguir, página por página.

- Gerenciador de Aplicação pode controlar os privilégios dos usuários (consulta, inserção, atualização, etc) a nível de página..
 - Para acesso ao Banco de Dados o Gerenciador de Aplicação pode verificar o *login/password* do usuário recebido do servidor *Web* com o *login/password* do usuário no SGBD, antes mesmo de se iniciar a conexão com o Banco de Dados.
 - O servidor do Banco de Dados pode usar os mecanismos padrão de segurança para que o Gerenciador de Aplicação se conecte a ele, segundo os métodos previsto pelo fabricante do SGBD.
-
- **Desempenho:** Alguns mecanismos que deterioram a eficiência da aplicação, implementados segundo a arquitetura Programas Executáveis CGI, são eliminados aqui. Não é necessário vários executáveis que a todo momento abrem e fecham uma conexão com o Banco de Dados. Como o Gerenciador de Aplicação sempre está residente em memória principal, ele pode realizar cache de cada aplicação sob um *login* de usuário.
 - **Desenvolvimento:** A complexidade da implementação desta arquitetura é muito maior que na arquitetura anterior. Exige-se bons conhecimentos técnicos da plataforma em que vai ser implementado o Gerenciador de Aplicação. Por exemplo, no ambiente Unix, para a comunicação entre o despachante e o Gerenciador de Aplicação pode-se usar *sockets* (Comer & Sttevens, 1994) cuja implementação exige muito conhecimento de *hardware* e linguagens de programação específicas.
 - **Portabilidade:** A arquitetura pode ser implementada dentro do escopo dos padrões *Web* já que a interface de comunicação servidor *Web* e Gerenciador de Aplicação é feita via CGI. No entanto, o Gerenciador de Aplicação pode ser extremamente dependente de ferramentas e da plataforma para a qual foi implementado.

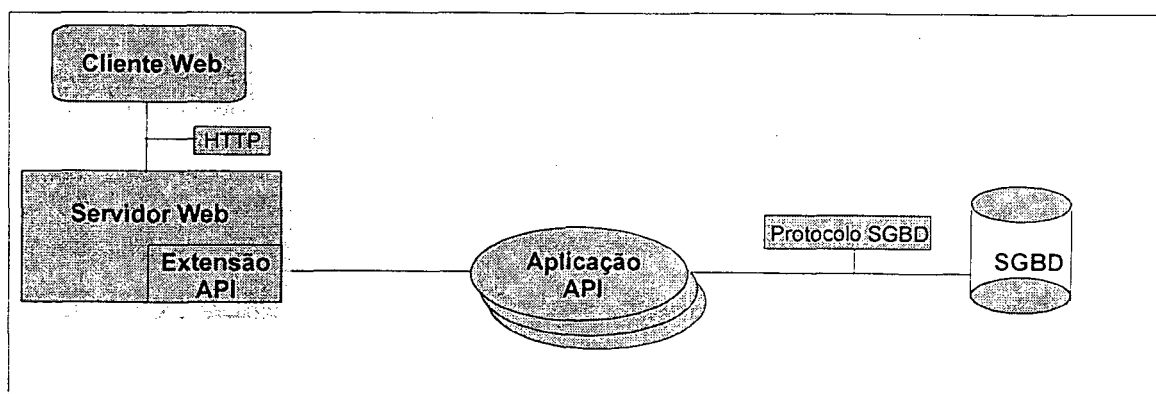
5.1.1.2 Servidor Web estendido

As arquiteturas que se encontram nesta categoria não usam a interface padrão CGI para execução das aplicações *Web* Banco de Dados. Ao contrário, as funcionalidades do servidor *Web* são estendidas de forma a proporcionar acesso mais otimizado e mais rápido a aplicações externas. Ganha-se portanto em desempenho quando comparado às arquiteturas que usam a interface CGI, mas perde-se em portabilidade da aplicação, ficando o desenvolvedor limitado às características do servidor *Web* proprietário.

a) Aplicação API

As APIs possibilitam a extensão das funcionalidades dos servidores *Web*. Uma destas funcionalidades é a possibilidade de se escrever aplicações APIs para acesso a um Banco de Dados dando origem à arquitetura denominada Aplicação API. A figura abaixo ilustra esta arquitetura.

Figura 5.5 : Arquitetura Aplicação API.



O funcionamento da arquitetura é mostrado a seguir.

1. O cliente *Web* solicita ao servidor *Web* a execução de uma aplicação API, que pode ou não já estar executando no mesmo espaço de memória em que o servidor *Web* está executando.
2. O servidor *Web* inicia a execução da aplicação API se ela ainda não estiver executando e repassa os dados enviados pelo cliente *Web* de acordo com a implementação API do servidor *Web*, ou seja, não se usa a interface CGI.
3. A aplicação API inicia a conexão com o Banco de Dados ou, dependendo da implementação da aplicação, pode-se substituir um despachante CGI por um despachante API que envia um pedido para um Gerenciador de Aplicação.
4. A aplicação API realiza a conexão com o Banco de Dados e, em seguida, repassa os dados ao servidor *Web* por canais próprios da implementação API do servidor *Web*.
5. O servidor *Web* retorna os dados ao cliente *Web*. A aplicação API normalmente fica residente em memória esperando por novo pedidos até que o servidor *Web* decida quando ela pode ser finalizada para liberar recursos do sistema.

Geralmente, nesta arquitetura o servidor *Web* oferece bibliotecas DLLs (*Dynamic Link Librarys*) específicas para conexão com o Banco de Dados. A conexão com o Banco de Dados pode ou não ser finalizada ao final de um pedido do cliente *Web*.

Isto vai depender das funcionalidades presentes na API proporcionada pelo servidor *Web*.

Muitos produtos comerciais que usam um das duas arquiteturas descritas anteriormente possuem uma versão de forma a suportar uma ou mais APIs de servidores *Web* proprietários, como por exemplo, *ColdFusion*, *NetImpact*, *Dynamo*, *Web.sql*, *NetDynamics*, que já suportam as APIs.

A seguir uma análise mais detalhada da arquitetura Aplicação API.

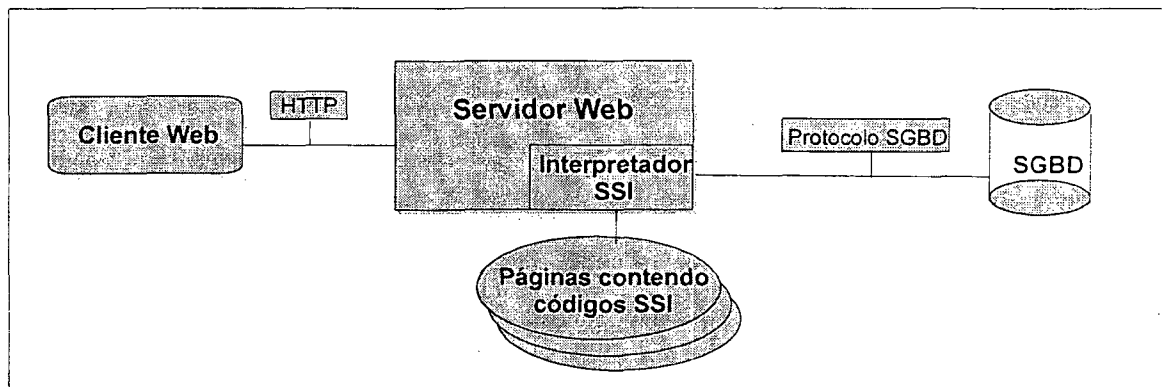
- **Aspectos transacionais:** A aplicação API pode ou não implementar mecanismos eficientes para a gerência do estado da aplicação. Existem diversos níveis de complexidade possíveis, que vão desde aplicações que se equivalem à arquitetura Programas Executáveis CGI até o nível mais sofisticado da arquitetura Gerenciador de Aplicação CGI.
- **Segurança/Acesso:** Pode-se também implementar diversos níveis de segurança/acesso ao Banco de Dados equivalentes à arquitetura Gerenciador de Aplicação CGI.
- **Desempenho:** O desempenho é melhor do que nas duas arquiteturas anteriores . De fato, o servidor *Web* e a aplicação API compartilham recursos de memória e, devido ao fato dos programas APIs ficarem residentes, eles são executados mais rapidamente do que os programas CGI ou os despachantes CGI.
- **Desenvolvimento:** Do ponto de vista de desenvolvimento o uso de APIs de servidores *Web* proprietários está dentre as tarefas de codificação mais difíceis em relação às outras arquiteturas.
- **Portabilidade:** A implementação da arquitetura é extremamente dependente da interface API do servidor *Web*. Como não existe nenhum padrão para APIs de servidores *Web* atualmente, a aplicação não é portátil nem entre servidores que seguem a norma padrão.

b) Aplicação SSI

A funcionalidade SSI implementada por alguns servidores *Web* permite a inserção de códigos (ou *tags*) especiais em documentos HTML que são

interpretados e executados automaticamente pelo servidor *Web* quando solicitado pelo cliente *Web*. Alguns desses códigos podem incluir funcionalidades para execução de comandos em um SGBD, dando origem à arquitetura de integração denominada Aplicação SSI. A figura 5.6 ilustra esta arquitetura.

Figura 5.6: Arquitetura Aplicação SSI.



O funcionamento da arquitetura é mostrado a seguir:

1. O cliente *Web* solicita uma página que possui, além dos elementos da linguagem HTML, códigos SSI para conexão e execução de comandos em um SGBD.
2. O servidor *Web* recupera a página e reconhece que a página contém códigos SSI para serem executados dinamicamente. Neste momento o servidor *Web* ativa o interpretador SSI, que é um de seus componentes.
3. O servidor *Web* se conecta ao SGBD apropriado e solicita a consulta. Os dados são retornados pelo SGBD e o servidor *Web* os formata de acordo com outros códigos SSI de formatação presentes no restante da página HTML.
4. O servidor *Web* encerra a conexão com o Banco de Dados e retorna a página de dados ao cliente *Web*.

Um exemplo de servidor *Web* que se enquadra nesta arquitetura é o servidor *WebQuest* (Webquest, 1997).

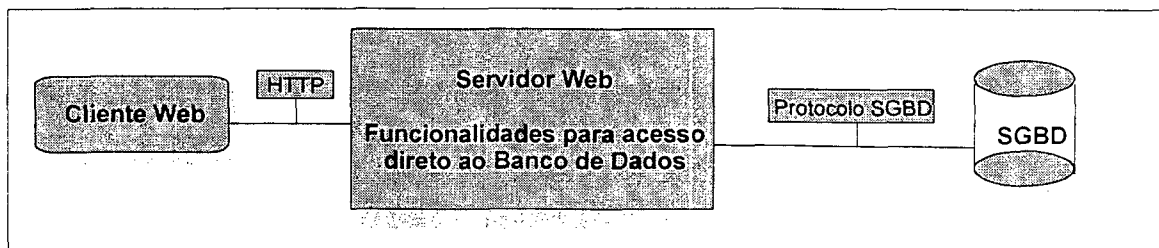
A seguir uma análise mais detalhada da arquitetura Aplicação SSI.

- **Aspectos transacionais:** A arquitetura é bastante limitada neste aspecto e apresenta os mesmos problemas da arquitetura Programas Executáveis. O estado da aplicação é perdido entre um pedido de cliente *Web* e outro e, para cada pedido de algum cliente *Web*, é realizada uma nova conexão com o Banco de Dados. Não há nenhum controle transacional fora do SGBD.
- **Segurança/Acesso:** O acesso ao Banco de Dados é feito como nas aplicações convencionais. O servidor envia o *login* do usuário e a *password* recebidos do cliente *Web*. Se tudo estiver correto o SGBD realiza o comando SQL e retorna os dados ao servidor *Web*.
- **Desempenho:** Como ressaltado no capítulo 3 o servidor *Web* deve examinar todo o documento a procura de códigos SSIs para execução dinâmica. Se for necessário realizar esta ação para todas as páginas HTML pode haver sobrecarga no processamento de pedidos de clientes *Web*. Caso contrário, elimina-se o problema de desempenho associado à verificação do código SSI.
- **Desenvolvimento:** É uma arquitetura excelente para o desenvolvimento de aplicações baseadas em Banco de Dados com rapidez e que não exijam sofisticação. Basta o conhecimento do código SSI, que na maioria das vezes é bem simples.
- **Portabilidade:** As aplicações não são portáveis já que as SSIs são implementadas de forma específica por cada servidor *Web* proprietário.

c) Acesso direto ao BD

O servidor *Web* pode ser implementado para, além de atender pedidos HTTP, atender pedidos que exijam acesso ao Banco de Dados sem nenhuma interferência da interface CGI ou dos mecanismos API ou SSI já descritos. Ele incorpora o protocolo de comunicação do SGBD e adiciona mecanismos próprios para o desenvolvimento de aplicações *Web* Banco de Dados totalmente integradas ao SGBD. Ou seja, o servidor *Web* funciona como um cliente do SGBD que suporta o protocolo HTTP. Esta forma de conexão origina a arquitetura denominada Acesso Direto ao Banco de Dados. A figura a seguir ilustra esta arquitetura.

Figura 5.7: Arquitetura Acesso Direto ao Banco de Dados.



O funcionamento da arquitetura é mostrado a seguir:

1. O cliente *Web* solicita ao servidor *Web* a execução de uma aplicação incorporada ao próprio servidor *Web*.
2. O servidor *Web* faz uma série de verificações internas (e.g., *login* e *password*) e inicia uma conexão com o Banco de Dados como se fosse mais um cliente do SGBD.
3. SGBD atende o pedido e retorna os dados ao servidor *Web*.
4. servidor *Web* formata os dados de acordo com a aplicação desenvolvida no servidor *Web* e retorna-os ao cliente *Web*. Normalmente o servidor *Web* fica

esperando por novos pedidos do cliente e não libera os recursos da sessão do usuário no SGBD enquanto o cliente *Web* estiver utilizando a aplicação.

Esta arquitetura pode ser usada também para transpor para o ambiente *Web* as aplicações de Banco de Dados cliente/servidor (Msa-Infor, 1997). Neste caso o servidor *Web* emula uma sessão de usuário para uma aplicação já existente como se fosse um cliente normal da aplicação. Toda a conversão de dados para o formato HTML é feita automaticamente pelo servidor *Web*. São exemplos desta arquitetura os servidores *CorpWeb Server* (Msa-Infor, 1997) e *AOLServer* (America, 1997).

A seguir uma análise mais detalhada da arquitetura Acesso Direto ao Banco de Dados.

- **Aspectos transacionais:** Como o servidor *Web* é projetado para ser um cliente do SGBD pode-se implementar bons mecanismos para controle de concorrência, gerenciamento do estado de transações *Web* e recuperação na presença de falhas.
- **Segurança/Acesso:** Dependendo da implementação do servidor *Web* esta arquitetura é mais segura que as demais para acesso ao Banco de Dados. De fato, pode-se compartilhar integralmente os mecanismos de segurança presentes nos SGBDs já que o acesso ao Banco de Dados e seus objetos pode ser implementado de acordo com o projeto de cada SGBD.
- **Desempenho:** O desempenho da arquitetura é similar ou superior à arquitetura Aplicação API dependendo das características da arquitetura: se distribuída ou não, se usa processamento paralelo ou não. No entanto, caso o servidor *Web* suporte todas as funcionalidades de um servidor *Web* comum e seja de fato exigido para estes fins, pode-se ter perda de desempenho devido à dupla funcionalidade do servidor *Web*.

- **Desenvolvimento:** O desenvolvimento de aplicações é dependente da implementação do servidor *Web*, o que vai exigir bons conhecimentos da arquitetura do servidor *Web*. No entanto, geralmente estão presentes ferramentas para desenvolvimento de aplicações geradas automaticamente.
- **Portabilidade:** Tem-se os mesmos problemas da arquitetura Servidor API. A aplicação é extremamente dependente do servidor *Web*.

5.1.2 Arquitetura de transações do lado do cliente *Web*

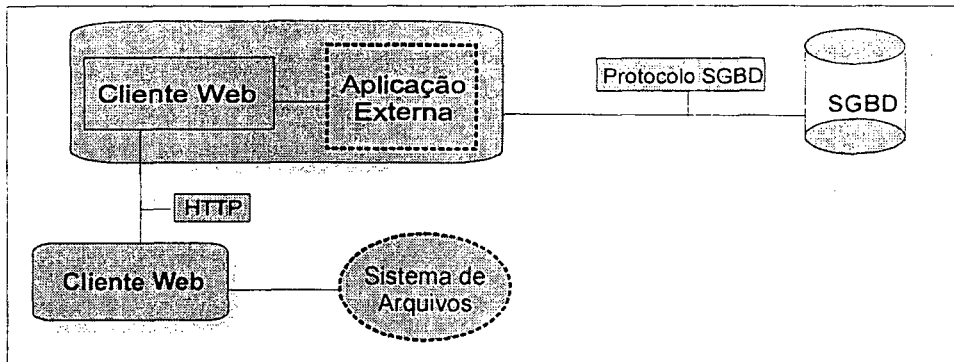
As arquiteturas de integração *Web* Banco de Dados que se enquadram nesta categoria têm como principal característica o fato do aplicativo que acessa o Banco de Dados usar recursos do lado cliente *Web*. Embora o código resida no servidor *Web* todo o processamento da lógica da aplicação é realizado na máquina do cliente *Web*. Assim, o cliente *Web* alivia o processamento do lado servidor *Web*, realizando o processamento da lógica da aplicação *Web* Banco de Dados.

5.1.2.1 Aplicação externa

Os clientes *Web* podem ser configurados para utilizar outros aplicativos para apresentação de dados que eles não sejam capazes de lidar diretamente. Por exemplo, a maioria dos clientes *Web* invoca um aplicativo externo específico quando ele recebe um tipo de dado sonoro. De forma análoga pode-se usar esta configuração para associar aplicações de Banco de Dados e a *Web*. Ou seja, a arquitetura de integração pode consistir de uma Aplicação

Externa ao cliente *Web* para acesso ao Banco de Dados. A figura 5.8 ilustra esta arquitetura.

Figura 5.8 - Arquitetura Aplicação Externa



O funcionamento da arquitetura é mostrado a seguir:

1. O cliente *Web* solicita um pedido ao servidor *Web*.
2. O servidor *Web* atende ao pedido enviando instruções para o cliente *Web* executar uma Aplicação Externa. Geralmente são enviados também arquivos de configuração da aplicação com uma extensão que a Aplicação Externa consiga interpretar.
3. O cliente *Web* recebe o pedido e imediatamente inicia a execução da Aplicação Externa para acesso ao Banco de Dados.
4. A Aplicação Externa assume o controle da aplicação, se conecta ao Banco de Dados, solicita um pedido e recebe os dados enviados pelo SGBD.
5. Os dados são exibidos no cliente *Web* pela Aplicação Externa. A execução da Aplicação Externa é finalizada quando o usuário seleciona uma opção apropriada no cliente *Web*.

Esta arquitetura de integração pode ser implementada configurando-se programas de Banco de Dados já existentes para serem executados no cliente

Web. Cabe ao cliente *Web* solicitar a execução dos programas de aplicação. Um exemplo seria um interpretador de formulários de Banco de Dados.

Um exemplo de como a funcionalidade de acesso a um Banco de Dados pode ser implementada nesta arquitetura é a introdução do conceito *plug-in* na *Web*, de modo a permitir que aplicações externas executem dentro do cliente *Web*. Um exemplo de *gateway* nesta arquitetura é o *PowerBuilder Plug-In* (Powersoft, 1997).

A seguir uma análise mais detalhada da arquitetura Aplicação Externa.

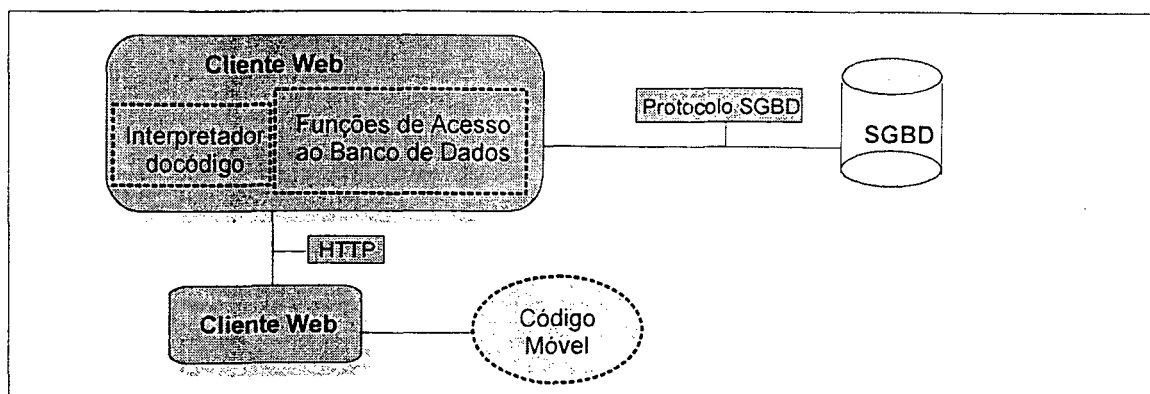
- **Aspectos transacionais:** A comunicação entre o Banco de Dados e a Aplicação Externa, que executa no cliente *Web*, geralmente é implementada pela própria Aplicação Externa e, neste caso, o controle transacional será gerenciado pelos mecanismos já existentes em Banco de Dados cliente/servidor. O grande problema é que os clientes *Web* não podem estar, em alguns casos, em pontos não alcançados por LANs ou WANs implementados pelos atuais Banco de Dados cliente/servidor, limitando a abrangência da aplicação. Neste caso, uma Intranet seria mais adequada à arquitetura.
- **Segurança/Acesso:** Aqui também os mecanismos para segurança e acesso ao Banco de Dados seguiriam os mecanismos já existentes para SGBDs sob a arquitetura de Banco de Dados cliente/servidor.
- **Desempenho:** Os aplicativos externos constituem uma técnica de ampliação do cliente *Web* que apresenta baixo desempenho, já que faz uso intenso dos recursos do sistema no cliente *Web*. Todos os aplicativos externos que forem executados durante o funcionamento da aplicação ficarão co-residentes na memória principal, o que poderá sobrecarregar a máquina cliente. Para evitar isso é necessário uma ampliação da funcionalidade do *browser*. Assim que é finalizada a aplicação, os recursos de memória da aplicação no cliente *Web* são liberados.

- **Desenvolvimento:** Pode-se aproveitar as ferramentas de desenvolvimento para as aplicações de Banco de Dados cliente/servidor tradicionais. Pode-se inclusive transpor a aplicação cliente/servidor de Banco de Dados para dentro do *browser* sem nenhuma modificação (Plain, 1996). No entanto, o desenvolvimento de novas aplicações na *Web* pode ficar limitado.
- **Portabilidade:** O cliente *Web* deve ser capaz de executar a Aplicação Externa e isto é naturalmente uma funcionalidade proprietária. Por exemplo, a estrutura *plug-in* da *Netscape* é proprietária, embora seja suportada pelas versões mais recentes do *browser* Internet Explorer da Microsoft.

5.1.2.2 Cliente Web estendido

A grande maioria dos clientes *Web* disponíveis no mercado possuem outras funcionalidades diferente daquela básica de apresentação dos dados recebidos do servidor *Web*, como previsto inicialmente pelos idealizadores da *Web*. Essa característica surge diante da grande limitação do cliente *Web* e de seus componentes, como a linguagem HTML. Esta forma de conexão, onde o *browser* é estendido de forma a ser capaz de acessar um Banco de Dados via linguagem de código móvel, dá origem à arquitetura Cliente *Web* Estendido. A figura 5.9 ilustra esta arquitetura:

Figura 5.9 - Arquitetura Cliente *Web* Estendido.



O funcionamento da arquitetura é mostrado a seguir:

1. O cliente *Web* solicita um pedido de um documento que contém ou faz referência ao código da aplicação para acesso ao Banco de Dados.
2. O servidor *Web* envia o documento e o código da aplicação. Juntamente com o código da aplicação pode ser enviado o código para gerenciamento da conexão com o Banco de Dados, como por exemplo, um *driver* gerente de conexão semelhante a um *driver* ODBC (North, 1996b; North, 1996c e Reed, 1996).
3. O cliente *Web* recebe o documento e o código da aplicação e imediatamente reconhece que deve iniciar a execução do código da aplicação com o auxílio de um interpretador que é um componente do *browser Web*.
4. O cliente *Web* se conecta ao Banco de Dados usando as funcionalidades de acesso ao Banco de Dados enviados junto com o código da aplicação ou incorporadas no cliente, executando o pedido do usuário.
5. O SGBD envia o resultado ao cliente *Web* e este, de acordo com o código da aplicação, exibe o resultado da operação ao usuário. A conexão com o Banco de Dados é, geralmente, finalizada após cada pedido ter sido atendido.

Esta arquitetura apresenta duas variantes principais. A primeira delas consiste em estender uma linguagem *script* interpretada pelo cliente, como *JavaScript* ou *VBScript*, para acesso a um Banco de Dados. Neste caso a linguagem *script*, inserida em páginas HTML, contém comandos para manipulação de comandos SQL. O interpretador da linguagem *script*, acoplado ao *browser Web*, executa o código acessando o Banco de Dados. Como

exemplo desta arquitetura existe a interface API JDBC (*Java DataBase Connectivity*) (Linthicun, 1996; Shah, 1996, Sun, 1997 E Sun 1997b) da *Sun Microsystems* que incorpora funções de acesso a Banco de Dados implementadas na linguagem Java semelhante à interface ODBC da Microsoft. O *gateway jConnect* (Sybase, 1997b) da *Sybase Corporation* é um exemplo da implementação de JDBC para o SGBD *Sybase*.

A seguir uma análise mais detalhada da arquitetura *Cliente Web Estendido*.

- **Aspectos transacionais:** Na maioria dos casos analisados para esta dissertação, a conexão com o Banco de Dados é iniciada e finalizada para cada pedido do usuário. No entanto, dependendo de como as funções de acesso ao Banco de Dados foram implementadas, pode haver suporte para gerenciar o estado da aplicação, presença de cursores para atualizações, recuperação de dados e suporte a funções importantes de Banco de Dados como *rollback* e *commit* em pontos específicos da aplicação.
- **Segurança/Acesso:** Como o acesso ao Banco de Dados é feito diretamente pelo usuário cliente, fica mais fácil a implementação das funcionalidades de segurança e acesso presentes nos SGBDs tradicionais. Um cuidado particular deve ser tomado no que diz respeito à segurança na rede, já que há transporte do código da aplicação.
- **Desempenho:** O desempenho da arquitetura depende muito da carga da rede. De fato, como o fluxo da comunicação segue o caminho cliente *Web*, servidor *Web*, cliente *Web*, Banco de Dados, cliente *Web*, há sérios problemas de desempenho se a rede estiver sobrecarregada.
- **Desenvolvimento:** O desenvolvimento de aplicações pode ser relativamente fácil se funções para acesso ao Banco de Dados já estiverem presentes, o que ocorre na maioria das vezes. Ferramentas proprietárias visuais também podem ajudar.

- **Portabilidade:** De forma que a arquitetura seja largamente utilizada a grande maioria dos *browsers Web* disponíveis deveriam ser capazes de entender o código da linguagem *script* ou da linguagem completa multi-plataforma. O primeiro caso é menos portátil atualmente devido ao problema de que não se conseguiu chegar ainda em um padrão de linguagem *script*.

5.1.3 Arquitetura de transações no lado cliente e servidor *Web*

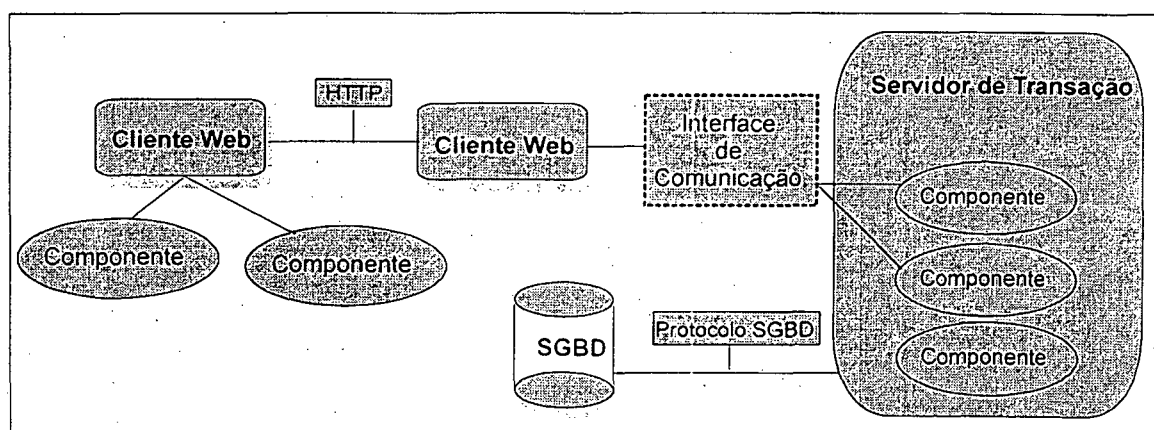
As arquiteturas de integração *Web* e Banco de Dados que se enquadram nesta categoria têm como principal característica o fato de que parte da aplicação que acessa o Banco de Dados executa no lado cliente *Web* e parte no lado servidor *Web*. O código da aplicação reside no servidor *Web* e a maior parte da lógica da aplicação é realizada no lado servidor *Web*. No entanto, pode-se ter uma parte da lógica da aplicação executando no lado cliente *Web* como complemento da aplicação que roda no servidor *Web* e, diferentemente das arquiteturas anteriores, os dois lados podem trocar mensagens entre si, sem a intervenção direta do servidor *Web*.

5.1.3.1 Servidor de Transação

A busca por mecanismos que possibilitem gerar um ambiente para execução de transações *Web* que conservem as propriedades ACID vem merecendo atenção especial por parte de fabricantes e pesquisadores interessados no desenvolvimento de aplicações complexas no ambiente *Web*. Em particular, o ambiente de integração *Web* e Banco de Dados deve ser

baseado numa arquitetura que solucione os dois principais problemas da integração destas tecnologias: o suporte ao gerenciamento do estado da aplicação e o suporte às propriedades ACID das aplicações baseadas em Banco de Dados. A busca por soluções eficientes para estes problemas tem dado origem à nova arquitetura de integração *Web* e Banco de Dados denominada aqui de Servidor de Transação. A figura 5.10 ilustra esta arquitetura e logo a seguir são descritos seus elementos principais.

Figura 5.10: Arquitetura Servidor de Transação.



Os componentes são as unidades básicas da aplicação. Estes componentes são projetados para serem interoperáveis em diferentes plataformas. Para isso os fabricantes disponibilizam o que se convencionou chamar de modelo de componente de *software* (*software component model*) (Hughes, 1997), que é, resumidamente um *framework* para a execução dos componentes (Berghel, 1996).

Os componentes de aplicação podem ser referenciados em páginas *Web* por meio de *tags*, de forma semelhante aos *applets* Java. Para que os componentes possam ser executados no cliente *Web* é necessário que o *browser Web* suporte o modelo de execução do componente. Esta limitação, aliada ao fato de serem tecnologias muito recentes, ainda tem restringido a execução de componentes nos clientes *Web*. Entretanto, em virtude do fato da tecnologia de componentes estar crescendo rapidamente e ser baseada em especificações já bastante difundidas como CORBA/IIP (*Common Object*

Request Broker Architecture/Internet Inter-ORB Protocol) (Object, 2000b e Oracle, 1996) e DCOM (*Distributed Component Object Model*) (Roy & Ewald, 1997 e Richards, 1996), espera-se que este problema diminua num curto intervalo de tempo. É importante ressaltar ainda que um componente no cliente *Web* pode se comunicar com um componente no servidor *Web*. O Servidor de Transação, descrito abaixo, proporciona o ambiente de execução dos componentes de aplicação no lado servidor *Web*.

O Servidor de Transação é um *framework* para execução de aplicações *Web* baseadas em componentes no servidor *Web* (Berghel, 1996). Suporta mecanismos para gerenciar, por exemplo, a manutenção do estado dos componentes, autenticação de um pedido de um componente, registros de *log* e serviços transacionais que possibilitam manter a atomicidade, consistência, isolamento e durabilidade das transações realizadas pelos componentes. Como definido em (Berghel, 1996), um *framework* "é um ambiente de *software* que é projetado para simplificar o desenvolvimento e o gerenciamento do sistema para um domínio de aplicação especializado". O domínio de aplicação aqui é o desenvolvimento de aplicações complexas no ambiente *Web*. Neste sentido, o Servidor de Transação pode, juntamente com o SGBD, propiciar ao ambiente *Web* um bom modelo transacional, não implementado em nenhuma das outras arquiteturas discutidas anteriormente.

Pode-se ver o Servidor de Transação semelhante aos Monitores de Processamento de Transações (*Transaction Processing Monitors - TP Monitors*) (Gray & Reuter, 1993 e Bernstein, 1990), muito utilizados em aplicações de Banco de Dados cliente/servidor em três camadas. De forma análoga a estes monitores, a principal função do Servidor de Transação é coordenar o fluxo de pedidos entre o cliente *Web* e os programas de aplicação (compostos por vários componentes de aplicação), que podem processar os pedidos junto ao SGBD. Isto permite que os desenvolvedores dirijam seus esforços para a lógica da aplicação sem ter que se preocupar com a infraestrutura para a execução da aplicação. O funcionamento desta arquitetura é mostrado a seguir:

1. O cliente *Web* solicita ao servidor *Web* a execução de um componente da aplicação no Servidor de Transação. Geralmente, alguns componentes são descarregados no cliente *Web* e tornam-se ativos neste momento, podendo ser acionados para realizarem ações em algum ponto da aplicação.
2. O servidor *Web* atende ao pedido disparando o componente via interface de comunicação. Esta interface pode ser algum mecanismo desenvolvido sob uma API do servidor *Web* ou algum mecanismo próprio suportado pelo Servidor de Transação para comunicação com um determinado servidor *Web* proprietário.
3. Em algumas implementações os componentes são autenticados para se verificar se podem ser disparados pelo usuário da aplicação. Eles são tratados como unidades atômicas de processamento e coordenados pelo Servidor de Transação.
4. O componente em execução pode ou não realizar uma ação sobre um ou mais Bancos de Dados. Caso seja solicitado uma ação sobre um Banco de Dados o Servidor de Transação trabalha em cooperação com o SGBD para que o pedido seja atendido e que sejam garantidas as propriedades ACID. O Servidor de Transação pode manter um pool de conexões com o BD.
5. Os componentes podem executar concorrentemente se usuários diferentes estiverem submetendo as mesmas ações ao mesmo tempo. O Servidor de Transação garante o isolamento e consistência das ações realizadas. Ao final de uma solicitação do usuário os dados são retornados via servidor *Web* ao cliente *Web*.

Como exemplos são citados a seguir 3 (três) Servidores de Transação. Para efeitos de comparação das tecnologias que implementam, cada um deles é descrito resumidamente.

- **Microsoft Transaction Server** (Microsoft, 1997g), da *Microsoft Corporation*, cujos componentes são denominados controles *ActiveX* (Microsoft, 1997 e North, 1996). Neste caso a Plataforma *ActiveX* (Hughes, 1997 e Shoffner, 1997) é o modelo de componente de software utilizado, baseado na especificação DCOM desenvolvida pela própria Microsoft.
- **Oracle WebServer 3.0**, (Oracle, 1996b e ORACLE 1997b), da *Oracle Corporation*, cujos componentes são denominados *Cartridges* (Oracle, 1997 e Oracle, 1996c). Neste caso o *Web Request Broker* (Anand et al, 1997, Adunuthula & Anand, 1996) é o modelo de componente de software utilizado, baseado na especificação CORBA/IIP desenvolvido pelo *Object Management Group* (OMG) (OBJECT, 2000).
- **Beanstalk da Sun Microsystems**, cujos componentes são denominados *Beans* (Sun, 1997e). Neste caso *JavaBeans* (Shoffner, 1997 e Hughes, 1997) é o modelo de componente de software utilizado.

Dois outros *gateways* se enquadram nesta arquitetura, com a variante de os componentes neste caso serem, na realidade, *applets* Java. Estes softwares são o *Internet DE-Light Client* (Transarc, 1996 ; Transarc, 1997b e Transarc, 1997c) desenvolvido pela *Transarc Corporation* e o *Jolt* (Bea, 1997; Bea, 1997b e Bea 1997c) desenvolvido pela *BEA Systems*.

A seguir uma análise mais detalhada da arquitetura Servidor de Transação.

- **Aspectos transacionais:** O objetivo da arquitetura é fazer com que a aplicação no ambiente *Web* suporte a característica *stateful* e transações satisfaçam as propriedades ACID. O Servidor de Transação, juntamente como o SGBD, garante estas características, proporcionando à *Web* uma verdadeira arquitetura de processamento de transações distribuídas, que potencialmente resolve os principais problemas transacionais discutidos ao longo deste trabalho, como gerência de estado e manutenção da integridade transacional.

- **Segurança/Acesso:** O Servidor de Transação proporciona uma camada intermediária de acesso aos componentes da aplicação que, em associação com o SGBD, pode proporcionar um alto nível de segurança aos objetos do Banco de Dados. Toda a segurança de execução da aplicação é garantida pelo Servidor de Transação, que requer uma identificação de sessão de usuário geralmente baseada em mecanismos de autenticação. O Servidor de Transação garante que o par (*login, password*) digitado no *browser* cliente seja validado contra o *login* e a *password* mantidos pelo SGBD. Isto elimina a necessidade do administrador *Web* manter três sistemas de autenticação: um no servidor *Web*, um na camada de aplicação e um no SGBD.
- **Desempenho:** O Servidor de Transação pode garantir um pool de conexões com o SGBD, otimizando em associação com o SGBD, as consultas submetidas durante a sessão do usuário. Como o Servidor de Transação não executa no mesmo espaço de memória do servidor *Web* não há o risco de operações do Servidor de Transação derrubarem o Servidor *Web* nem vice-versa, como na arquitetura Aplicação API. Uma vez ativado, o Servidor de Transação fica residente em memória, podendo realizar processamento distribuído em várias máquinas disponíveis, com balanceamento automático de carga de processamento.
- **Desenvolvimento:** Como a tecnologia de componentes para a *Web* é muito recente e envolve conhecimentos técnicos muito superiores às demais arquiteturas pode-se dizer que o desenvolvimento de aplicações nesta arquitetura é muito mais complexa que nas demais. Nos casos analisados não foi encontrado nenhuma ferramenta para gerar automaticamente os componentes de aplicação específicos para serem usados com o Servidor de Transação.
- **Portabilidade:** O Servidor de Transação é extremamente dependente da plataforma em que foi desenvolvido. No entanto, a idéia de desenvolver

componentes baseados em tecnologias já fortemente estabelecidas como CORBA (*Web Request Broker, JavaBeans*) e DCOM (*ActiveX*) têm por objetivo o desenvolvimento de aplicações que sejam interoperáveis em plataformas distintas que é, em linhas gerais, o objetivo inicial do ambiente *Web*. Isto garante um nível de portabilidade de aplicações *Web* diferente das arquiteturas anteriores, já que um Servidor de Transação pode em tese, trabalhar bem com qualquer outra tecnologia que utilize o mesmo padrão.

5.2 Comparação de arquiteturas

É mostrado na figura 5.11 a seguir um quadro comparativo das oito arquiteturas de transação *Web* Banco de Dados apresentada anteriormente, de acordo com as principais funcionalidades levantadas ao longo desta dissertação.

De forma a ser possível o uso de uma mesma sistemática para comparação entre as diversas arquiteturas segundo as funcionalidades em estudo é utilizado um critério de "cotações" *ad-hoc* baseado na quantidade do símbolo " * " presente. Assim, um " * " significa que a funcionalidade é fraca, difícil ou não é robusta para a arquitetura em estudo. Já dois " * * " significa que a funcionalidade é boa e apresenta melhores características do que aquelas avaliadas com um " * ". Por fim, três " * * * " significa que a funcionalidade é ótima e, comparativamente às demais, apresenta as melhores características.

Tabela 5.11: Quadro comparativo das arquiteturas

	Controle de Transações	Segurança e Acesso	Desempenho	Desenvolvimento	Portabilidade
Programas Executáveis CGI	*	*	*	**	***
Gerenciador de Aplicação CGI	***	***	**	*	***
Aplicação API	***	**	***	*	*
Aplicação SSI	*	*	*	***	*
Acesso Direto Ao BD	***	***	***	*	*
Aplicação Externa	**	***	*	***	*
Cliente Web Estendido	**	**	**	**	**
Servidor de Transações	***	***	***	*	*

O Controle de transações diz respeito à forma como cada arquitetura pode gerenciar o estado da transação submetida pelo usuário (suporte à característica *stateful*) e como cada arquitetura lida com o problema do gerenciamento da execução das transações *Web* Banco de Dados da aplicação (suporte às propriedades ACID). As arquiteturas Programas Executáveis CGI e Aplicação SSI são as mais problemáticas neste sentido, já que a conexão com o Banco de Dados é finalizada após cada pedido do usuário. Pode-se ter inconsistências, uma vez que não é possível controlar o acesso concorrente. As arquiteturas Gerenciador de Aplicação CGI, Aplicação API, Acesso Direto ao Banco de Dados e Servidor de Transação são as que apresentam melhores condições para implementar mecanismos mais eficientes neste sentido. Todas adotam a estratégia de simular uma sessão de usuário contínua sob o Banco de Dados e, portanto, podem fazer com que os mecanismos para controle de concorrência funcionem de forma coerente. A arquitetura Servidor de Transação, em particular, reúne as melhores condições

para, juntamente com o SGBD, trazer para o ambiente *Web Banco de Dados* um modelo de transações segundo uma arquitetura distribuída.

A Segurança e Acesso diz respeito a restrições de acesso aos objetos do Banco de Dados gerenciados pelo SGBD. Para aplicações que exijam um controle total dos dados acessados e manipulados por um SGBD, como em uma *Intranet*, as arquiteturas Acesso Direto ao Banco de Dados e Aplicativo Externo podem ser consideradas satisfatórias, já que nestes casos a segurança e acesso ao Banco de Dados são projetados especificamente de acordo com o SGBD que se está utilizando. As arquiteturas Gerenciador de Aplicação CGI e Servidor de Transação, quando implementadas de forma a cooperar com o SGBD, podem proporcionar níveis de segurança aos objetos do SGBD e à aplicação tão bom quanto estes dois.

O Desempenho de aplicações *Web Banco de Dados* é particularmente influenciado pela arquitetura adotada. De fato, nas arquiteturas Programas Executáveis CGI e Aplicação SSI o desempenho é considerado ruim visto que uma operação não aproveita os mecanismos de otimização dos atuais SGBDs. Em particular, é muito caro abrir e fechar a conexão com o Banco de Dados para cada pedido de um usuário (Hadjefthymiades & Martakos, 1997). A arquitetura Programas Executáveis CGI ainda tem o agravante de, para cada novo usuário, ser necessário disparar um novo processo da aplicação, deteriorando ainda mais o desempenho da aplicação. As arquiteturas Gerenciador de Aplicação CGI, Aplicação API, Acesso Direto ao Banco de Dados e Servidor de Transação, procuram resolver os fatores que limitam a performance neste ambiente, mantendo a aplicação residente em memória, realizando cache de conexões com o Banco de Dados para diversos usuários simultâneos, adotando estratégias de balanceamento distribuído de carga e processamento em várias máquinas disponíveis. Já as arquiteturas Aplicativo Externo e Cliente *Web Estendido* podem sofrer do sério problema de tráfego na rede. No entanto, se a aplicação estiver restrita a uma *Intranet*.

O Desenvolvimento de Aplicações diz respeito ao conhecimento necessário para implementar uma aplicação *Web Banco de Dados*. Aplicações desenvolvidas segundo as arquiteturas Aplicação SSI e Aplicativo Externo são

consideradas mais fáceis de serem implementadas. De fato, as funções SSI para acesso ao Banco de Dados são geralmente disponibilizadas como mais algumas funções que o servidor *Web* pode fornecer ao usuário. A arquitetura Aplicativo Externo geralmente permite transferir para o ambiente *Web* as aplicações de Banco de Dados já disponíveis em outro ambiente. Isto pode ser feito por meio de pequenas configurações no cliente *Web*, semelhantes à estrutura *Plug-In da Netscape*. Aplicações que se enquadram na arquitetura Aplicação API, Gerenciador de aplicação CGI, Acesso Direto ao Banco de Dados ou Servidor de Transação podem ser imediatamente geradas na presença de ferramentas para este fim. No entanto, são difíceis de serem mantidas e exigem amplo conhecimento técnico da arquitetura usada.

Por último, a Portabilidade, que é a característica mais marcante no ambiente *Web* Banco de Dados. Consideramos uma arquitetura portátil se ela só depende de mecanismos implementados pelo ambiente *Web* padrão. Neste sentido, as únicas arquiteturas com característica totalmente portáveis são as que usam a interface CGI padrão para conexão com o Banco de Dados, ou seja, as arquiteturas Programas Executáveis CGI e Gerenciador de Aplicação CGI. A arquitetura Cliente *Web* Estendido entretanto, se destaca dos demais que usam ferramentas proprietárias. Isto se deve às características de portabilidade que podem estar presentes na linguagem que implementa a arquitetura.

Concluindo, a arquitetura de integração mais adequada ao ambiente *Web* Banco de Dados vai depender de vários fatores e principalmente dos requisitos da aplicação que se pretende desenvolver. Por exemplo, se a aplicação exigir flexibilidade, portabilidade e requisitos de gerenciamento do estado segundo o modelo de sessão contínua de usuário, a melhor arquitetura é Gerenciador de Aplicação CGI. Se a aplicação estiver restrita à corporação (*Intranet*) então ela vai exigir bons mecanismos de segurança e performance. Neste caso, as arquiteturas Servidor de Transação e Acesso Direto ao Banco de Dados via servidor *Web* proprietário são as mais indicadas. No entanto, caso se queira aplicações simples (que envolvam somente consultas, por exemplo) e rápido desenvolvimento, a arquitetura Aplicação SSI pode ser uma ótima solução.

5.3 Conclusão

Ao longo deste capítulo apresentou-se oito arquiteturas de transações *Web* Banco de Dados. Foram discutidas vantagens e desvantagens de cada uma delas, levando-se em consideração diversas funcionalidades presentes nos atuais Sistemas de Gerenciamento de Banco de Dados. Um quadro comparativo entre as arquiteturas foi apresentado.

As arquiteturas Gerenciador de Aplicação CGI e Servidor de Transação tendem a ser as mais utilizadas no futuro próximo. A primeira delas apresenta boas características de portabilidade, bom desempenho e permite implementar a característica *stateful* no ambiente de integração. A segunda, além da característica *stateful*, possibilita bom suporte a transações ACID sobre a *Web* e é baseada em arquiteturas para ambientes distribuídos como CORBA e DCOM. Além destas, a arquitetura Cliente *Web* Estendido é bastante promissora pois proporciona uma real extensão à linguagem HTML e tem bom comportamento em todos os itens analisados no quadro comparativo.

Apesar destas previsões, a principal arquitetura atualmente utilizada para o desenvolvimento de transações que integrem as tecnologias ainda é a arquitetura Programas Executáveis CGI. No próximo capítulo será apresentado as alternativas de segurança de transações além de alguns modelos de transações e as tecnologias envolvidas.

6 TECNOLOGIAS ENVOLVIDAS

Neste capítulo serão apresentadas as tecnologias que são utilizadas no desenvolvimento de transações na web, como *Coorba Tansaction Service*, *Object Transaction Service (OTS)*, *Transacton Internet Protocol (TIP)*, *X/open - DTP - Distributed Transaction Processing*.

6.1 *Corba transaction service*

A CORBA - *Common Object Request Broker Architecture* (OMG, 1996 e OMG 1997b) define o *OTS-Object Transaction Service* (OMG 1997c), um componente da arquitetura de gerenciamento de objeto do Grupo de gerenciamento de objeto. O núcleo do CORBA é o *ORB-Object requested broker* cuja especificação descreve a infraestrutura de comunicação para aplicações distribuídas.

No CORBA, tudo é objeto, e todo gerenciador de transação que concordar em cooperar com o gerenciador de transações baseados em CORBA, deverá concordar com interfaces baseadas em objetos CORBA, e deverá confiar em um *ORB-Object Request Broker*. Nós acreditamos que na visão de uma transação na internet, onde o usuário é centrado, não há necessidade de uma estrutura pesada para implementar um serviço de

transação na Internet. A interoperabilidade entre os gerenciadores de transação deve ser reduzida para sua simples natureza.

Além disso, não está pronto para interoperar no nível de transação com outros sistemas. O OTS é desenhado para completar a interoperabilidade de rede, e mais precisamente sobre a forma de "Serviços de Transação Múltiplas, ORBs múltiplos, isto só será possível para um serviço de transação para interoperar com um serviço de transação de cooperação utilizando diferentes ORBs" (OMG, 1997).

Na última revisão do CORBA, definiram muitos protocolos de interoperabilidades, e em particular um protocolo *Internet Inter-ORB* (IIOP), uma especialização do protocolo *General Inter-ORB* (GIOP) criado para conexões TCP/IP. Mas novamente, se a interoperabilidade a nível de ORB for bem definida, a interoperabilidade transacional não é mencionada. Este lida com problemas para interoperação de transação, então eles formam um único domínio OTS sob muitos de múltiplos ORBs. Além disso, OTS é executado, no topo de um ou vários ORBs, e este requisito está lidando com aplicações legadas que devem experimentar um importante esforço para concordar com isto.

6.2 Object transaction service (OTS)

O OTS suporta os conceitos de transações ACID (Little & Shivastava, 1998). O OTS fornece interfaces que permitem múltiplos objetos distribuídos a cooperar em uma transação, mesmo que todos os objetos entraram em *Commit* ou *Abortaram* suas mudanças juntos. Entretanto, o OTS não requer que todos os objetos tenham um comportamento transacional. Os objetos podem escolher não suportar operações transacionais no todo, ou suportá-las para algumas requisições mas não para outras.

A especificação dos serviços de transação distingue objetos recuperáveis e objetos transacionais. Objetos recuperáveis são os que contêm o estado atual que poderá ser mudado por uma transação e deverá ser informado quando a transação termina ou aborta para garantir a consistência das mudanças de estado. Isto se realiza registrando apropriados objetos que suportem a interface *Resource* com a transação corrente. No contraste, um simples objeto transacional precisa não necessariamente ser um objeto recuperável, se o seu estado é atualmente implementado outro objeto recuperável. A maior diferença é que um simples objeto transacional não precisa tomar parte de um protocolo de *commit* usado para determinar a vinda de uma transação, desde que esta não mantenha nenhum estado, tendo delegado a responsabilidade para outro objeto recuperável que tomará parte do processo de *commit*.

É importante entender que o OTS é um simples protocolo que garante um comportamento transacional seja obedecido, mas não diretamente suporta todas as propriedades de transação. Ele requer outros serviços de operação que requerem funcionalidade como:

- **Persistência/Serviço de Recuperação:** Requerido para suportar a atomicidade e propriedades de durabilidade.
- **Serviço de Controle de Concorrência:** Requerido para suportar propriedades de isolar.

Para participar da transação OTS (Litle & Shivastawa, 1998), um programador deve se concentrar em:

- Em criar *Resource* e objetos *SubtransactionAwareResource* para cada objeto que participar da transação ou sub-transação. Estes *Resources* são responsáveis pela persistência, controle de concorrência, e recuperação para o objeto. A OTS invocará estes objetos durante cada fase da sub-transação de *preparo/commit/abort*, e os *Resources* devem executar os trabalhos apropriados.

- Registrar *Resource* e objetos *SubtransactionAwareResource*, no momento certo na transação, e garantir que os objetos só serão registrados uma vez na transação fornecida. Como uma parte do registro um *Resource* deverá receber uma referência para um *RecoveryCoordinator* que deverá ter sido persistente quando uma recuperação ocorrer em função de uma falha.
- Garantir que neste caso de uma transação *nested*, qualquer propagação de recursos como locação de transações pais são corretamente executadas. Propagação de objetos *SubtransactionAwareResource* para os pais também devem ser gerenciadas.
- No evento de falhas, o programador ou o administrador do sistema é responsável por dirigir a recuperação de um *crash* para cada *resource* que estiver participando da transação.

A OTS não fornece nenhuma implementação de *resource*. Este deve ser fornecido pelo programador ou pelo implementador OTS. As interfaces definidas na especificação OTS são também de baixo nível para a maioria de programadores de aplicação. Por esta razão foi projetado o *JTSArjuna* para fazer uso de uma nova interface de Serviços de Objetos Comuns mas fornecida por API de alto nível para a construção de aplicações transacionais e estruturas. Esta API automatiza muito de todas as atividades concentradas que participam de uma transação OTS.

A arquitetura deste interage com o controle de concorrência e serviços de persistência, e automaticamente registra recursos apropriados para objetos transacionais.

Estes recursos também poderão utilizar os serviços de persistência e concorrência.

6.3 *Transacton internet protocol (TIP)*

Algumas pesquisas para que se colocassem serviços transacionais na WWW. A mais importante contribuição foi o *Transactional Internet Protocol* (TIP) (Litle & Shivastawa, 1998), que consiste essencialmente em um protocolo e *two phase-commit* para aplicações na Internet. Entretanto, o protocolo TIP é projetado para facilitar a comunicação e o processo de *commit* para servidores que executam seus próprios gerenciamentos de transação. Além disso, o protocolo TIP requer que todos os servidores conheçam um ao outro, mas em um trabalho colaborativo do que um processamento transacional. Finalmente, o usuário tem uma pequena latitude em escolher o servidor e interagir com ele.

Existe uma visão muito simples para uma transação na Internet, vamos chamá-la de *I-Transação*, que é radicalmente diferente. Os gerenciadores de transação de um Banco de Dados envolvidos em uma simples transação para internet não se comunicarão nunca um com o outro e o protocolo de *commit* atômico é projetado para garantir somente a precisão da ação de um usuário. A precisão de um estado do Banco de Dados é deixada para o gerenciador de transação local trabalhando com o Banco de Dados.

6.4 *X/open - DTP - Distributed transaction processing*

O DTP (X/OPEN, 1993) é o padrão X/OPEN para a distribuição de transações sob muitos nós. Gerenciadores de transação podem interoperar através do componente de processo de transação OSI (OSI TP) que implementa o protocolo de transação da camada OSI, e a comunicação são realizadas agradecendo a transação de *Remote Procedure Call* (TxRPC) (X/OPEN, 1995).

O paradigma RPC é considerado caro quando ele é utilizado na Internet, entretanto DTP parece não apropriado. Se transações são designadas a interoperar facilmente (o que não é o caso do CORBA), sua interoperação com outros sistemas é condicionada a um forte consentimento lidando com aplicações legadas.

6.5 Conclusão

Ao longo deste capítulo foi abordado um estudo das principais tecnologias envolvidas em ambientes transacionais como O CORBA (*Common Object Request Broker Architecture*) (OMG, 1996), que descreve a infraestrutura de comunicação para aplicações distribuídas, o OTS (*Object Transaction Service*), que fornece interfaces permitindo vários objetos distribuídos a cooperar em uma transação, o TIP (*Transaction Internet Protocol*) , que é o Protocolo de *two –phase commit* para *internet* e por fim o X/OPEN – DTP (*Distributed Transactions Processing*), que é o padrão de processos de transação distribuída que vale para distribuição de transações em muitos nós.

No próximo capítulo, com base de que a principal arquitetura atualmente utilizada para integrar as tecnologias Web BD ainda é a arquitetura Programas Executáveis CGI, será apresentado no capítulo seguinte um estudo de caso desta arquitetura por meio de uma aplicação Web Banco de Dados que está sendo usada no departamento de Informática de uma determinada Universidade

7 ESTUDOS DE CASO: TRANSAÇÕES NA WEB

Neste capítulo, será analisada uma aplicação *Web Banco*, desenvolvida por Lifshitz & Lima (1998), que se enquadra na arquitetura de transações que utilizam *Programas Executáveis CGI* cuja variante é SQL dentro da linguagem de programação hospedeira. O foco será dado para as principais funcionalidades de Banco de Dados no ambiente integrado discutidas ao longo dos capítulos anteriores.

Apesar do grande número de *gateways* comerciais já existentes ou que estão sendo lançados, a grande maioria das aplicações *Web Banco de Dados* disponíveis ainda se enquadra na arquitetura de transações que utilizam *Programas Executáveis CGI*. Assim, entende-se que o estudo da aplicação representa o estudo da grande maioria das atuais implementações que visam integrar as tecnologias *Web* e Bancos de Dados.

Esse estudo de caso, pretende discutir detalhes de características já abordadas e outras ainda não citadas de forma mais objetiva e direcionada à arquitetura utilizada escolhida por Lifshitz & Lima (1998), O mesmo foi escolhido pois levanta muitos dos problemas funcionais discutidos ao longo da dissertação.

A seção 7.1 descreve características da implementação da aplicação, e nas seções seguintes apresenta-se diversos tópicos já descritos ao longo dos capítulos anteriores, agora direcionados ao estudo da arquitetura de transações que utilizam *Programas Executáveis CGI*. Estes tópicos são desenvolvimento, segurança, desempenho, gerenciamento do estado da

aplicação, controle de concorrência, consistência das transações, recuperação, restrições de integridade e portabilidade.

7.1 Descrição da aplicação

De acordo com Lifshitz & Lima (1998), responsáveis pelo desenvolvimento deste estudo de caso, o objetivo principal da aplicação foi o de propiciar aos professores do departamento de Informática de uma determinada Universidade e á bibliotecária responsável, um sistema de controle de solicitações e compras de livros para a biblioteca do departamento. Através do sistema é possível encaminhar pedidos de compras de novos livros, controlar os recursos disponíveis para compra dos livros através de cotas dadas aos professores, gerenciar a solicitação, compra e execução da aquisição de livros junto aos fornecedores e distribuidores.

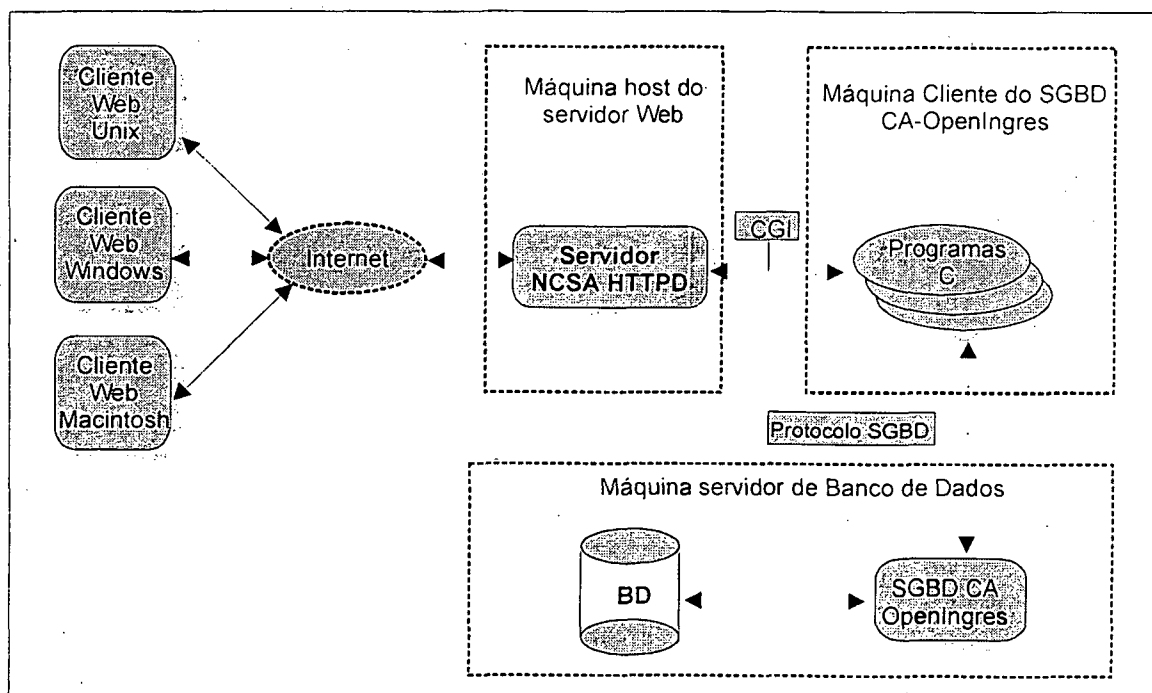
A aplicação se inicia quando um usuário seleciona a URL que contém referência ao programa de autenticação para verificação do par (*login, password*) para uso do sistema. Uma vez autorizado a usar o sistema o usuário pode realizar consultas parametrizadas sobre a base de dados para acompanhar o processo de aquisição de obras já solicitadas ou realizar novas solicitações. Este é essencialmente o módulo do sistema que pode ser usado pelos professores do departamento. A bibliotecária possui um módulo adicional em que pode realizar várias ações sobre o sistema visando o controle de aquisições das obras solicitadas junto aos distribuidores ou fornecedores dos pais ou do exterior. Ela pode, por exemplo, montar um pedido de compra selecionando as obras ainda não adquiridas, pode atualizar dados de um processo de compra em andamento, podem cadastrar novos usuários do sistema, novos distribuidores e novas fontes de recursos para aquisição de obras, e pode realizar diversas consultas. A escolha do ambiente *Web* para o desenvolvimento da aplicação deveu-se ao fato de que todos os professores

atualmente de um departamento de informática têm acesso à *Internet* e, em particular, à *Web*. Além disso, a *Web* era, e ainda é, o ambiente multi-plataforma ideal para que cada professor do departamento utilizasse o sistema na plataforma que melhor lhe conviesse, de qualquer lugar de uma Universidade ou mesmo fora dela.

Foi escolhido o SGBD relacional *CA-OpenIngres* release 1.1 da *Computer Associates* (Computer, 1995), disponível na plataforma SUN Unix. Este SGBD é uma versão comercial do *Ingres* acadêmico (Date, 1987), muito conhecido no meio científico por implementar várias funcionalidades importantes de Banco de Dados. Além dessas características, a versão do SGBD *CA-OpenIngres* utilizada foi uma versão cliente/servidor cujo cliente pode estar em várias máquinas da plataforma SUN ou sob a rede *Windows NT* (Computer, 1995b).

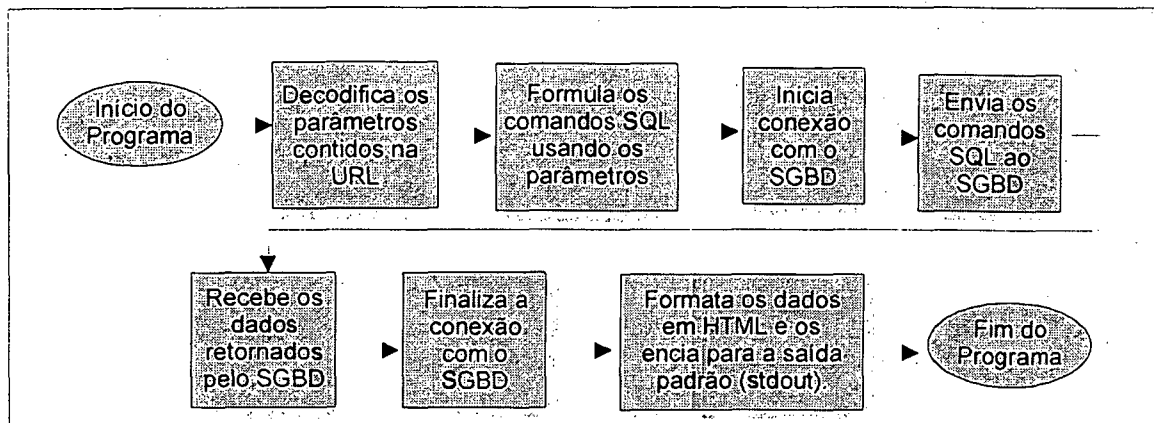
A arquitetura da aplicação e todos os seus componentes envolvidos na versão liberada inicialmente são mostrados na figura 7.1 abaixo.

Figura 7.1 – Arquitetura da Aplicação.



Logo a seguir, na figura 7.2, encontra-se o fluxo de dados dos programas da aplicação.

Figura 7.2 - Fluxo de dados dos programas da aplicação.



A aplicação está disponível na *Internet* e pode ser usada a partir de qualquer *browser* que suporte HTML versão 2.0 ou superior e a linguagem *JavaScript*. Quando o cliente *Web* envia um pedido ao servidor *Web* contendo uma URL, que inclui o nome de um programa CGI da aplicação, o servidor *Web* inicia a execução do programa CGI como um processo separado, enviando os parâmetros inseridos pelo usuário. Os programas que fazem parte da aplicação residem numa outra máquina que é cliente do servidor de Banco de Dados *CA-OpenIngres*. Uma vez disparados pelo servidor *Web*, o programa CGI decodifica os parâmetros enviados pelo servidor *Web*, formula um ou mais comandos SQL e inicia a conexão com o servidor de Banco de Dados.

Após os comandos terem sido efetivados pelo SGBD os dados são retornados ao programa CGI e a conexão é finalizada. O programa formata os dados recebidos do SGBD em HTML e os envia ao servidor *Web* de acordo com a especificação CGI. Este por sua vez retorna a página de dados ao cliente *Web*, que irá exibí-la ao usuário. Todo o processo se repete após o usuário preencher campos de entrada, quando for o caso, e selecionar uma opção apropriada na página que lhe foi mostrada, que irá disparar um novo programa da aplicação.

7.2 Desenvolvimento

As seguintes considerações durante a fase de desenvolvimento merecem destaque:

- O processo de depuração da aplicação foi muito mais complexo do que em aplicações convencionais de Banco de Dados. Isto se deve ao fato de que, além da lógica da aplicação, adicionou-se a lógica da apresentação da interface (HTML) e a lógica de validação de dados (*JavaScript*) no código dos programas.
- A transferência de variáveis de entrada do cliente *Web* (e.g., formulários) para as consultas (e.g., SQL) no servidor de Banco de Dados não é trivial. Este é um ponto particularmente importante, em virtude do fato de que a transferência de dados de um formulário HTML para a aplicação via interface CGI ser bastante complexa.
- Em todos os programas da aplicação que recuperavam grandes volumes de informações no SGBD, foi necessário usar técnicas de alocação dinâmica de memória para que os programas suportassem a carga de dados a ser tratado.
- O tamanho dos executáveis CGI gerados foram consideráveis, da ordem de um *megabyte* cada. Isto foi devido, em parte, às bibliotecas SQL usadas pela linguagem C. Toda a aplicação ocupou cerca de 40 *megabytes* de espaço em disco, o que é considerável, já que não se tem um ambiente de desenvolvimento genérico.

7.3 Segurança

A aplicação só pode, em princípio, ser usada por usuários clientes autorizados. O mecanismo utilizado para garantir este nível de segurança baseou-se na verificação do par *login* e senha, solicitados na primeira página da aplicação, contra uma tabela de autorizações no SGBD.

A execução dos programas junto ao SGBD é feita sob uma conta genérica no ambiente Unix denominada *www*, que possui autorização para realizar operações de consultas, inserções e atualizações de registros no Banco de Dados. Também é importante ressaltar que é sob essa conta que é executado o servidor *Web*. Inicialmente, esta estratégia parecia a mais adequada, devido ao fato de que todos os usuários da aplicação tinham o mesmo nível de segurança (*grants*) relativos aos objetos da aplicação, não sendo, portanto, necessário manter os *grants* para cada usuário da aplicação. Isto também aliviou a carga no gerenciador de acesso do SGBD. Pode-se apontar várias falhas de segurança da aplicação como as descritas a seguir:

- O par *login* e senha solicitado na página inicial da aplicação é enviado sem proteção pela rede até o servidor *Web*, que os repassa ao programa CGI correspondente.
- Todos os demais dados transmitidos na rede podem ser interceptados por usuários não autorizados, já que não foi utilizado nenhum mecanismo seguro para a comunicação cliente *Web* e servidor *Web*.
- Em algumas páginas da aplicação foi necessário passar os parâmetros da aplicação através da URL. Neste caso, o *login* do usuário também foi colocado na URL para que o programa CGI disparado capturasse o usuário da aplicação (ver seção 7.6). Isto, adicionado ao fato de que nenhuma outra verificação do par *login* e senha é realizado pela aplicação uma vez já efetivada na página inicial da aplicação, permite que um usuário do sistema

possa se transformar em outro usuário a partir do ponto da aplicação onde se encontra esta URL.

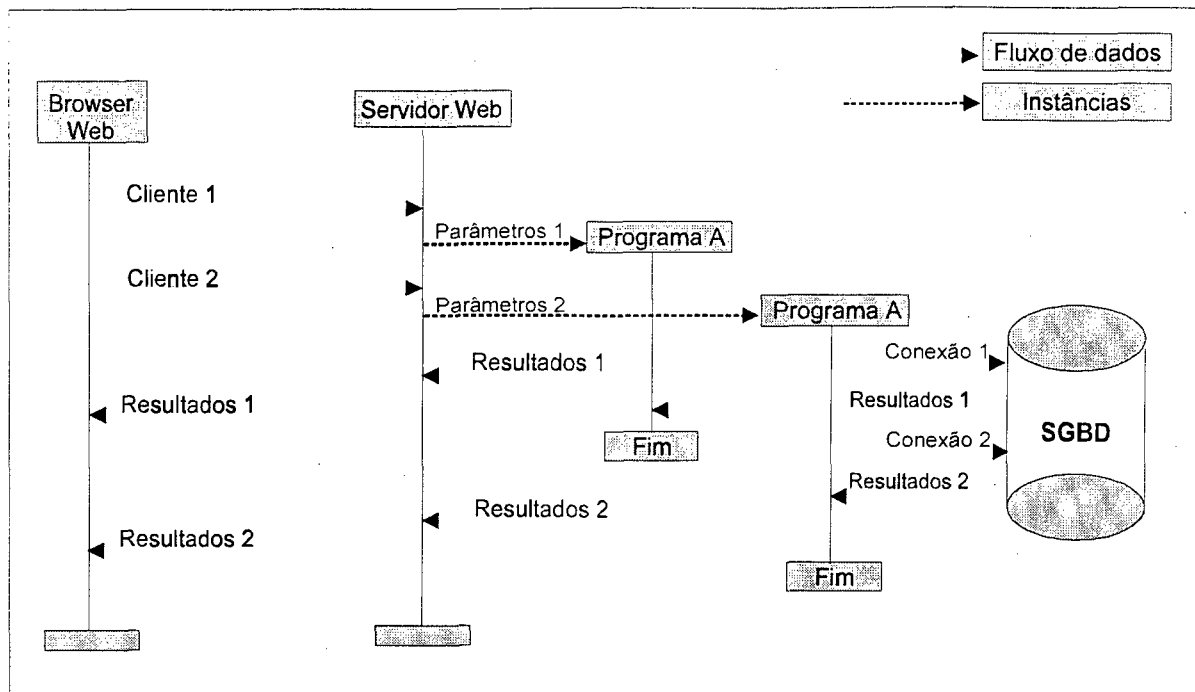
- Baseado no parágrafo anterior, mesmo pessoas que não são usuários da aplicação poderão executá-la se conhecerem a URL com os parâmetros que devem ser passados ao programa que pretendem executar. A partir daí a aplicação torna-se ativa a partir das opções presentes nas páginas da aplicação. Isso, porque o servidor *Web* dispara um programa CGI sob uma conta genérica não verificando se o usuário está ou não autorizado a disparar o programa.

Os problemas de segurança descritos anteriormente não são considerados graves para a aplicação desenvolvida já que os dados armazenados no Banco de Dados da aplicação não são considerados confidenciais. Além disso a URL que dá início à aplicação só foi divulgada aos usuários.

7.4 Desempenho da aplicação

A figura 7.3 abaixo mostra a estratégia seguida pelo servidor *Web* para ativação dos programas CGI que compõem a aplicação. Estão representados dois pedidos idênticos de dois usuários distintos, o que levou o servidor *Web* a disparar duas instâncias do mesmo programa CGI.

Figura 7.3 - Ativação dos programas CGI da aplicação.



Depois de recebido o primeiro pedido, o servidor *Web*, seguindo a especificação CGI, cria a primeira instância do programa solicitado, que por sua vez inicializa todas as variáveis de memória do programa. Enquanto o programa ainda está em execução, interagindo com o SGBD ou tratando os dados, um novo pedido do mesmo programa chega no servidor *Web*. Apesar da existência de uma instância do processo já em andamento, uma nova instância é criada pelo servidor *Web* independentemente da primeira. Novas variáveis de memória do programa são iniciadas e a nova instância segue disputando recursos com a primeira instância do programa. Tão logo os programas são completados, eles vão sendo finalizados pelo servidor *Web*. O mesmo acontece quando vários programas CGI distintos são solicitados ao servidor *Web*.

Notou-se que o desempenho da aplicação é muito dependente da carga a que o servidor *Web* está sendo submetido. Logo, pode ser uma boa estratégia fazer com que os programas CGI sejam executados em outra máquina distinta da máquina onde é executado o servidor *Web*.

7.5 Desempenho do SGBD

Baseado na figura 7.3 anterior fica claro que é o número de conexões estabelecidas com o SGBD durante a execução da aplicação é igual ao número de interações de cada usuário do sistema. Cada conexão implica numa soma substancial de tempo, e é mostrado recentemente em estudos de desempenho da arquitetura de transações que utilizam *Programas Executáveis CGI* (Hadjefthymiades & Martakos, 1997). Além disso, os mecanismos de otimização de consultas do SGBD *CA-OpenIngres* não podem ser utilizados plenamente, uma vez que a conexão não é mantida ao longo da interação do usuário com a aplicação.

Apesar dos problemas de desempenho aqui citados, o desempenho da aplicação como um todo não é ruim. Dois fatores contribuem para isso: os usuários são poucos, portanto, raramente existem duas ou mais conexões simultâneas no SGBD e o tráfego na rede tende a ser pequeno, possuindo características típicas de uma *Intranet*.

7.6 Gerenciamento do estado da aplicação

A questão de manutenção mínima do estado da aplicação foi sem dúvida o maior problema encontrado na implementação da aplicação. Isto se deve ao fato da conexão com o SGBD ter que ser aberta e finalizada para cada pedido do usuário. O objetivo a ser atingido era como solicitar a informação ao usuário somente uma vez e torná-la disponível.

O principal problema deste tipo de solução para a manutenção mínima do estado da aplicação está na modelagem da aplicação, que deve prever quais as informações escondidas devem ser transportadas entre as páginas da aplicação durante sua execução.

7.7 Controle de concorrência

Os aspectos de concorrência aos dados no Banco de Dados também podem ser discutidos com base na figura 7.3 anterior. Quando dois programas distintos solicitam uma sessão com o SGBD, duas conexões distintas são estabelecidas e o SGBD isoladamente garante a concorrência no acesso aos dados durante a execução das operações solicitadas pelos programas.

No entanto, alguns problemas de concorrência, particularmente de isolamento dos dados, podem ocorrer:

- Podem existir situações em que a bibliotecária está atualizando informações dos livros recém adquiridos através de várias páginas de interação com o usuário e, algum professor, consulte informações de alguns destes livros. Como a conexão com o SGBD é finalizada para cada interação, não é possível manter o bloqueio das informações que foram recuperadas para atualização.
- Existe o problema de que o usuário que está utilizando o sistema sempre o faz sob um único *login*. Isto implica que existe somente um usuário, que concorre consigo mesmo o tempo todo no SGBD.

7.8 Consistência

Uma transação deve deixar o SGBD num estado consistente ao final de sua execução. Mecanismos presentes no SGBD *CA-OpenIngres* sempre garantem o estado consistente dos dados ao final da transação. No entanto, devido à possibilidade do usuário voltar a uma página de interação anterior via

navegação *back/forward* dos *browsers*, a aplicação pode apresentar um comportamento não desejado.

O principal problema foi encontrado nas páginas que geravam atualizações no Banco de Dados. Isto porque nas interações que geram consultas a informações no Banco de Dados não há problemas de perda de consistência das informações armazenadas decorrentes da navegação *back/forward*. No máximo, a consulta é re-submetida ao Banco de Dados, o que pode gerar uma carga maior de trabalho no sistema como um todo.

No caso de atualizações o problema pode ser muito sério. Por exemplo, em algumas páginas de inclusão de novas informações os usuários poderiam submeter a página e, logo em seguida, retornar à mesma página de inclusão e re-submetê-la, duplicando a informação no Banco de Dados.

No entanto, houve situações em que não foi possível contornar o problema utilizando-se de consultas no Banco de Dados. Foi feita então uma tentativa de se usar uma opção disponível no *browser* da *Netscape* que permite instanciar um novo processo do *browser* quando se inicia a aplicação, de forma que o *browser* não exiba aos usuários as opções *back* e *forward*, mas só as opções implementadas pela aplicação. Alguns usuários do sistema logo descartaram essa alternativa devido ao fato de que, não somente estas opções seriam retiradas, mas todas as demais funcionalidades dos *browsers*, como a possibilidade de impressão dos dados.

Um problema particularmente importante do ambiente *Web* e não resolvido pela aplicação consiste da característica *Web* em armazenar páginas no *cache* local do cliente *Web*. Uma vez recuperado um documento o cliente *Web* usa o *cache* local para armazená-lo de forma que o usuário posteriormente possa consultá-lo novamente sem que para isso seja necessário uma nova conexão com o servidor *Web*.

O outro problema de inconsistência não resolvido decorrente da navegação dos *browsers* está relacionado à opção *stop* dos *browsers*, que permite cancelar uma solicitação enviada ao servidor *Web* ainda em andamento. Se o usuário seleciona a opção *stop* durante a execução de uma solicitação, ele não vai receber a resposta de que a operação solicitada foi ou

não realizada com sucesso, já que o *browser* sempre ignora toda resposta do servidor Web após a seleção da opção *stop*. Não é possível resolver este problema devido às características da arquitetura de transações que utilizam *Programas Executáveis CGI* em que se enquadra a aplicação.

7.9 Recuperação

Outro problema de difícil solução na aplicação implementada está relacionado com as possíveis falhas de sistema que podem ocorrer e como tratá-las. Mostrou-se anteriormente os pontos de possíveis falhas no ambiente de integração e chegou-se a conclusão de que, para o tratá-las adequadamente, era necessário uma cooperação entre a aplicação e o SGBD segundo uma arquitetura em que o *gateway* implemente uma sessão de usuário contínua junto ao SGBD, o que não foi implementado pela aplicação em estudo.

De fato, se o usuário submete uma página da aplicação e ocorre uma falha no *browser*, na rede, no servidor *Web*, na aplicação ou no SGBD, ele não terá condições de saber se sua operação foi ou não bem sucedida. É retornada ao cliente *Web* uma mensagem genérica do ambiente *Web* que não especifica onde ocorreu a falha e se a falha ocorreu antes ou depois da operação ter sido submetida.

As únicas falhas tratadas pela aplicação se referem às falhas no SGBD causadas por alguma inconsistência da transação sendo submetida (*deadlocks* ou quebra de integridade referencial, por exemplo). Nestes casos a aplicação recebe uma mensagem do SGBD informando a ocorrência da falha e a envia ao cliente *Web*.

Uma questão particularmente importante sob o ponto de vista de falhas no ambiente de integração e ainda não comentada aqui refere-se à arquitetura cliente/servidor do SGBD *CA-OpenIngres*. Esta arquitetura é interessante

porque torna a aplicação mais escalável. No entanto, a comunicação entre a aplicação cliente e o servidor de Banco de Dados era interrompida com frequência, devido a diversos fatores, como por exemplo, quedas de tensão. Isso estava inviabilizando a execução normal da aplicação, já que as falhas tornaram-se muito freqüentes. A solução adotada foi colocar os programas da aplicação na mesma máquina em que estava o servidor de Banco de Dados, o que diminuiu as freqüentes falhas na aplicação devido a falhas na comunicação cliente.

7.10 Restrições de integridade

É desejável, para efeitos de desempenho, que pelo menos algumas das restrições de integridade sejam satisfeitas no *browser* cliente. Na aplicação desenvolvida existem várias situações em que a integridade semântica pode ser primeiramente tratada no cliente *Web* para só depois ser enviado o pedido ao servidor de Banco de Dados. Por exemplo, restrições da forma:

- Tipos de dados, correspondendo à restrição de que um campo só pode ser de um tipo, por exemplo, numérico.
- Valores não nulos, correspondendo à restrição de que um campo é de preenchimento obrigatório.
- Formato de preenchimento, correspondendo à restrição de que um campo só pode estar num determinado formato, por exemplo, campos datas no formato mês/dia/ano.

Para tratar restrições deste tipo no cliente *Web* é necessário usar ferramentas proprietárias, já que o cliente *Web* não foi projetado para lidar com consistência de dados, por mais simples que sejam.

Na aplicação implementada optou-se por utilizar a linguagem de programação interpretada *JavaScript* para tratar as restrições de integridade no cliente *Web*, quando possível.

A implementação mostrou que a opção por um mecanismo proprietário como a linguagem *JavaScript* para tratar o problema da restrição de integridade no cliente *Web* ou para estender a funcionalidade da linguagem HTML pode ter graves conseqüências em termos de portabilidade da aplicação. De fato, mesmo embora a linguagem seja suportada pelos *browsers* citados, houve ocorrências de usuários que não conseguiram usar a aplicação porque o *browser* simplesmente não entendia a linguagem JavaScript implementada. As causas foram diversas, como problemas da versão do *browser* utilizado e configuração inadequada do *browser*, entre outros.

Apesar dos problemas, deve-se destacar os ganhos em termos de desenvolvimento. De fato, como as principais consistências para entradas de dados foram tratadas no cliente, os programas CGI se tornaram mais simples em termos da lógica da aplicação. Por exemplo, não foi necessário fazer prévias verificações de campos de preenchimento obrigatório nos programas CGI.

7.11 Portabilidade

A aplicação está implementada segundo a interface CGI e portanto qualquer servidor *Web* que a implemente pode ser usado. O maior problema em termos de portabilidade, no entanto, foi o fato de se ter usado a linguagem *JavaScript* para tratamento de restrições de integridade, como descrito na seção anterior. Como o ambiente *Web* está em plena evolução corre-se o risco de em poucos anos toda aplicação não ser mais utilizável, bastando para isso que a linguagem *JavaScript* não se firme no mercado.

7.12 Conclusão

Ao longo deste capítulo foram discutidas diversas funcionalidades de uma aplicação *Web* baseada em Banco de Dados desenvolvida por Lima (1998). Como a aplicação se enquadra na arquitetura de transações que utilizam *Programas Executáveis CGI*, que representa atualmente a maneira como é implementada a maior parte das aplicações *Web* Banco de Dados existentes, espera-se que os pontos levantados ao longo deste capítulo tenham esclarecido diversos problemas relacionados às funcionalidades da maioria dos *gateways* implementados segundo esta arquitetura e, em particular, levantado as principais deficiências da arquitetura.

Numa avaliação mais objetiva, pode-se afirmar que a arquitetura não é adequada à grande maioria das aplicações *Web* Banco de Dados. A arquitetura apresenta sérias deficiências no que diz respeito à manutenção das propriedades ACID no ambiente de integração, não permite o gerenciamento eficiente do estado da aplicação e não é segura. Neste sentido, aplicações *Intranet/Internet* que requeiram atualizações no Banco de Dados são desaconselhadas de serem implementadas usando-se a arquitetura de transações que utilizam *Programas Executáveis CGI*.

8 *BOLERO* UMA FERRAMENTA DE MERCADO

Neste capítulo analisaremos uma Ferramenta do Mercado, o *Bolero*, que se encaixa no contexto dessa dissertação, e se enquadra perfeitamente como um Servidor de Transações, cuja característica é dar o suporte ao gerenciamento do estado da aplicação e o suporte às propriedades ACID das aplicações baseadas em Banco de Dados. A ênfase maior será dada às características da Ferramenta, principalmente a parte de controle de longas transações.

A busca por mecanismos que possibilitem gerar um ambiente para execução de transações Web que conservem as propriedades ACID vem merecendo atenção especial por parte de fabricantes e pesquisadores interessados no desenvolvimento de aplicações complexas no ambiente Web. Em particular, o ambiente de integração Web e Banco de Dados deve ser baseado numa arquitetura que solucione os principais problemas. Por isso a escolha de uma ferramenta de mercado desenvolvida pela *Software AG*, uma *Software House* alemã, o *Bolero*, que é totalmente voltada para o desenvolvimento desses tipos de aplicações¹

O *Bolero* é um poderoso ambiente, integrado e produtivo para a construção e implementação de aplicações orientadas a objetos e baseados em componentes. As principais áreas de uso do *Bolero* são: aplicações *e-bussiness*, bem como a integração de dados e aplicações já existentes com

¹ SOFTWARE AG Corporation: Bolero Technical Details . Disponível em: <http://www.softwareag.com/bolero/technical/description.html>) Acesso em 20 agosto 2000

novos sistemas. O *Bolero* integra ao mundo *Web* a cultura de programação caracterizada pelo uso de terminologia, tecnologia e métodos consistentes, presentes no desenvolvimento de grandes sistemas corporativos.

O estudo desta ferramenta pretende discutir detalhes de todos os módulos que compõe o *Bolero*, além de descrever sua principal característica que é ser um Servidor de Transações. O *Bolero* se classifica como Servidor de Transação devido a um de seus módulos, o *Bolero Application Server*, que atua como ambiente central do ambiente *Bolero*. É ele que dará o suporte a persistência de objetos e principalmente o suporte a longas transações.

Na seção 8.1 discutiremos as principais características da ferramenta, já na seção 8.2 vamos identificar as funcionalidades de cada módulo do *Bolero* e enfim na seção 8.3 vamos descrever como o *Bolero* gerencia e administra uma longa transação. E finalmente na seção 8.4 será descrita a conclusão.

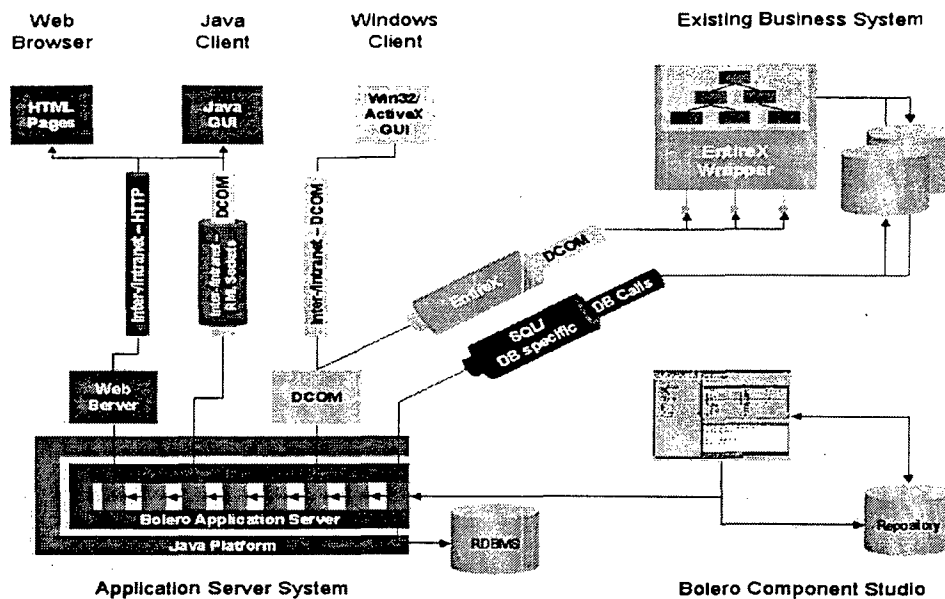
8.1 A fábrica de aplicações para o comércio eletrônico

O *Bolero* é um abrangente ambiente de desenvolvimento voltado para a criação e a manutenção de aplicações baseadas em componentes para a plataforma Java. Os componentes *Bolero* se caracterizam pela facilidade de distribuição e pela integração bidirecional com componentes não *Bolero* através das principais interfaces padronizadas (*DCOM*, *JavaBeans* e *RMI*).

O *Bolero* é capaz de gerar aplicações completas que podem operar de forma distribuída, ou seja, desde componentes cliente (para apresentação) até componentes servidor de dados, passando pelos componentes-servidor de dados, passando pelos componentes-servidor de aplicações

Diferentemente das demais ferramentas de desenvolvimento baseadas em Java e aderentes ao conceito "*thin clients*", o *Bolero* visa a construção de componentes-servidor voltados para o atendimento de aplicações comerciais.

Figura 8.1: Estrutura do Bolero



Fonte: TEIVELIS, Daniel. Business Oriented Language EnviROnmen. Disponível em: <<http://www.consist.com/br/br405000.html>. Acesso em 12 abril 2000

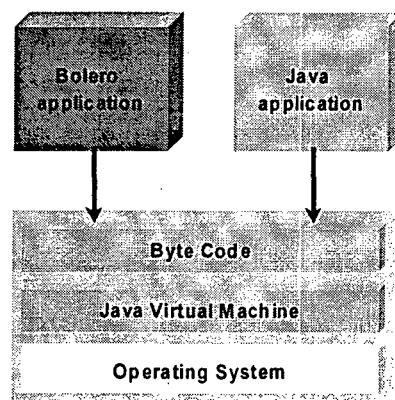
O *Bolero* é composto por uma linguagem de programação orientada a objetos, um compilador, um repositório de dados compartilhado e um servidor de aplicações. O *Bolero* gera *Java Bytecode* diretamente correspondente ao JDK V1.1 (ou acima), ou opcionalmente *Java source code*, que pode ser utilizado em qualquer plataforma que suporte uma *Java Virtual Machine (JVM)* homologada e o servidor de aplicações *Bolero*²

² SOFTWARE AG Corporation: Bolero Enterprise Class. Disponível em: <http://www.softwareag.com/bolero/brochures.htm>. Acesso em 17 agosto 2000

8.1.1 O Bolero e o Java

O *Bolero* ganha destaque pois é uma ferramenta baseada na arquitetura Java. Assim como o Java, o *Bolero* é totalmente orientado a objetos. O compilador *Bolero* gera *Java bytecode* diretamente. Opcionalmente, o *Bolero* também pode gerar *Java Source code*. Esse *bytecode* pode ser migrado para outros sistemas operacionais, executado em qualquer plataforma na qual estiverem presentes uma JVM certificada e o *Bolero Application Server* ("write once, run anywhere"). O código *Bolero* também pode ser integrado com outras aplicações baseadas em componentes, com processamento local ou distribuído. O *Bolero* oferece fácil interoperabilidade com outras aplicações e componentes baseados em Java, pois reconhece as classes Java como classes *Bolero* e vice-versa. As classes *Bolero* e Java podem assim criara instâncias, herdar propriedades e chamar métodos umas as outras³.

Figura 8.2 : O Ambiente *Bolero X Java*



Fonte: TEIVELIS, Daniel. Business Oriented Language EnviRONmen. Disponível em: <http://www.consist.com/br/br405000.html>. Acesso em 12 abril 2000

³ SOFTWARE AG Corporation: Bolero Definition . Disponível em: <http://www.softwareag.com/bolero/features.htm>. Acesso em 18 agosto 2000

Além de utilizar todos os recursos da plataforma Java, o *Bolero* dispensa a intervenção do desenvolvedor para a realização de uma série de tarefas complexas como: gerência automática de memória (*garbage collection*), ausência de ponteiros, monitorações de limites de *arrays* e tratamento facilitado de exceções, entre outros.

O *Bolero* tem construtores que suportam a implementação, disponibilidade e a integração de aplicações comerciais particularmente eficientes.

8.1.2 Integração com a Web

O uso do *Bolero* para o desenvolvimento de aplicações comerciais baseadas em ambientes Internet/Intranet requer uma perfeita integração entre os componentes *Bolero* e as tecnologias *Web*. O *Bolero* oferece uma série de características facilitadoras: os componentes *Bolero* podem ser disponibilizados como *applets* que, por sua vez podem ser descarregados por um *Web browser* a partir de um *Web server* e executar como se fossem Java *applets*. A partir do momento em que a *applet* estiver processando no *browser*, ele poderá estabelecer uma conexão direta com qualquer componente do servidor sem depender do protocolo HTTP (*Hyper Text Transport Protocol*). Como alternativa ao DCOM, poderão ser utilizados os métodos de comunicação suportados pelo Java API, como RMI, conexões *socket* ou RMI via IIOP (*Internet Inter-ORB Protocol*).

Para os *Web browsers* e as plataformas-cliente que ainda não suportam Java, o *Bolero* oferece uma forma alternativa de integração *Web* através da sua capacidade de gerar páginas HTML dinâmicas (*Bolero Server Pages*). Um componente do *Bolero Application Server* comunica-se com um servidor HTTP utilizando a *Java Servlet API*. O *Bolero* gera páginas HTML dinâmicas que são

retornadas ao *browser front-end* através do servidor http⁴. Scripts gerados em linguagens como *JavaScript* ou *Visual Basic Script* também podem acessar objetos *Bolero* a partir de um *Web browser*. O *JavaScript* pode ativar métodos locais do *Bolero* como se fossem métodos locais Java, e o *VBScript* pode utilizar a capacidade dos componentes *Bolero* de oferecer serviços via uma interface de automação baseada no padrão COM.

8.1.3 Aplicações alvo

O *Bolero* é ideal para a geração de aplicações de missão crítica em ambientes comerciais de processamentos de dados. Muito embora o principal uso do *Bolero* seja o desenvolvimento de aplicações *e-commerce*, as facilidades oferecidas pelo *Bolero* são muito mais abrangentes. Além do suporte oferecido pelo *Bolero* para a fácil modelagem da lógica envolvida nos processos comerciais, ele oferece ainda um grande número de funcionalidades especialmente importantes para os ambientes operacionais caracterizados por grandes quantidades de usuários concorrentes e pelo acesso a grandes volumes de dados disponíveis na modalidade 24x7. Além das características da linguagem já descritas, o *Bolero* oferece, por exemplo, persistências de objetos através de mapeamento Objeto-Relacional e um sofisticado conceito de controle de transação ACID (*Atomic, Consistent, Isolated, Durable*), como também suporta Transações Longas que controlam processos comerciais distribuídos, envolvendo normalmente extensos períodos de tempo.

Independente da sua relação muito próxima com a linguagem Java e da sua grande afinidade com a filosofia Java, o *Bolero* não toma partido no debate entre clientes “magros” versus clientes “gordos”. Muito embora a filosofia de

⁴ SOFTWARE AG Corporation: *Bolero Benefits*. Disponível em: <http://www.softwareag.com/bolero/benefits.htm>. Acesso em 18 agosto 2000.

projeto do *Bolero* esteja centrada em um ambiente servidor e portanto, favoreça o uso de clientes “magros”, o *Bolero* não radicaliza esta posição em favor de nenhuma plataforma de *hardware* específica. Graças à sua abertura, O *Bolero* suporta tanto os clássicos PCs, como os “*networks computers*” equipados com uma *Java Virtual machine*. Como ambientes-servidor, o *Bolero* suporta as plataformas *Windows NT*, *Unix* e *mainframes OS/390*. Em função da plataforma escolhida, pode-se utilizar compiladores JIT (*Just-In-time*) ou compiladores de alto desempenho para converter o *Java Bytecode*, ou opcionalmente o *Java source code*, gerado pelo compilador *Bolero*, em código de máquina extremamente eficiente e específico para a plataforma escolhida⁵.

8.1.6 Modelos de Projeto

O conceito de modelos de Projeto do *Bolero* permite a especificação de padrões de programas pré-fabricados que são abstrações de etapas recorrentes de processamento. Um modelo de projeto pode ser “copiado” e então adaptado a um processo de negócio específico. O *Bolero* já é fornecido com um grande número Modelos de Projeto prontos para uso, que podem ser modificados pelos desenvolvedores ou replicados para servir propósitos específicos. Um exemplo de um padrão recorrente normalmente encontrada em uma grande variedade de contextos de aplicações é o relacionamento 1:n existente, por exemplo, entre um cabeçalho e as Linhas Individuais de pedido em um Objeto de “Fatura”, ou entre objetos como “Departamento” e “Funcionário”.

Um outro exemplo típico poderia ser o Modelo de Transição de Estado, pelo qual é possível automatizar parcialmente a definição de transições de

⁵ SOFTWARE AG Corporation: Bolero Architecture . Disponível em: <http://www.softwareag.com/bolero/architecture.htm>. Acesso em 18 agosto 2000

estado para a implementação de Transações Longas. Como construtores reutilizáveis pré-fabricados, os Modelos de Projeto do Bolero ajudam os desenvolvedores de software a padronizar a estrutura de componentes e a implementar rapidamente aplicações consistentes e sem erros.

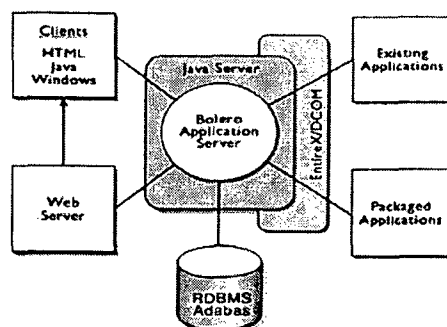
8.2 Módulos do *Bolero*

O Bolero é composto por 5 módulos. Cada um deles será detalhado nas seções que se seguem. São eles: *Application Server*, *Component Studio*, *GUI Builder*, *Report Writer* e o Repositório Compartilhado.

8.2.1 O *Bolero Application Server*

O *Bolero Applications Server* – juntamente com a JVM – fornece a funcionalidade de implementação exigida pelas aplicações *Bolero*.

Figura 8.3: Estrutura do *Bolero Application Server*



Fonte: TEIVELIS, Daniel. Business Oriented Language EnviRONmen. Disponível em: <<http://www.consist.com/br/br405000.html>>. Acesso em 12 abril 2000

As classes para cálculos de números inteiros, decimais e de datas, para o controle de transações, mapeamento objeto-relacional e funcionalidades DCOM (*Distributed Component Object Model*) são parte integrante do *Bolero Application Server*.

Sendo 100% puro Java, o *Bolero Applications Server* pode ser implementado em qualquer ambiente onde exista uma *Java Virtual Machine* certificada.

Em conjunto com outras ferramentas, o conector de componentes baseado no padrão DCOM, os componentes do *Bolero* podem interoperar em aplicações heterogêneas com componentes implementados em outras linguagens de programação como, por exemplo, Natural ou C++.

A primeira impressão visual causada por um programa *Bolero* é a grande facilidade com que o código do Programa pode ser lido. Considerando que a sintaxe Java, semelhante à da linguagem C, está voltada principalmente para programadores técnicos, o *Bolero* posiciona-se como uma ferramenta para programadores comerciais que precisam se comunicar em equipes grandes e também com outros especialistas como analistas e projetistas.

Com o *Bolero* não são mais necessárias longas e confusas formulações para a definição de tarefas rotineiras. A persistência dos objetos, por exemplo, pode ser conseguida através da adição de uma simples palavra à definição da classe ou alternativamente, por meio de um clique do mouse no *Bolero Component Studio*. Assim, o alto grau de abstração oferecido pelo *Bolero* isola o desenvolvedor de *software* dos complexos detalhes inerentes ao uso da programação Java. Além disso, o *Bolero* oferece uma série de extensões e de melhorias vitais para a implementação de componentes comerciais. Graças ao suporte ao *Unicode* o ambiente *Bolero* facilita a internacionalização da codificação: os textos dos componentes das aplicações visíveis pelos usuários finais podem ser armazenados em conjuntos específicos para cada linguagem e carregados e tempo de execução em função do idioma escolhido.

Finalmente, o enfoque orientado a objetos e baseado em componentes do *Bolero* favorece uma metodologia de desenvolvimento que privilegia a

utilização de modelos, classes e métodos reutilizáveis em oposição à codificação individual de módulos que só podem ser aproveitados em um contexto específico.

A reutilização de *software* é uma característica freqüentemente associada ao desenvolvimento de sistemas orientados a objetos, pois parece lógico reutilizar classes que já tenham sido implementadas em outros contextos. Mas, a reutilização pura e simples de *software* pode ser perigosa e muito dispendiosa.

O *Bolero* oferece ferramentas para a implementação consistente de aplicações e de componentes através de conceitos tais como: Programação por Contrato, Modelos de projeto, Transações Longas e Classes Parametrizadas. Com estas facilidades é possível transformar a promessa da reutilização de código em uma realidade produtiva.

8.2.2 O Bolero Component Studio

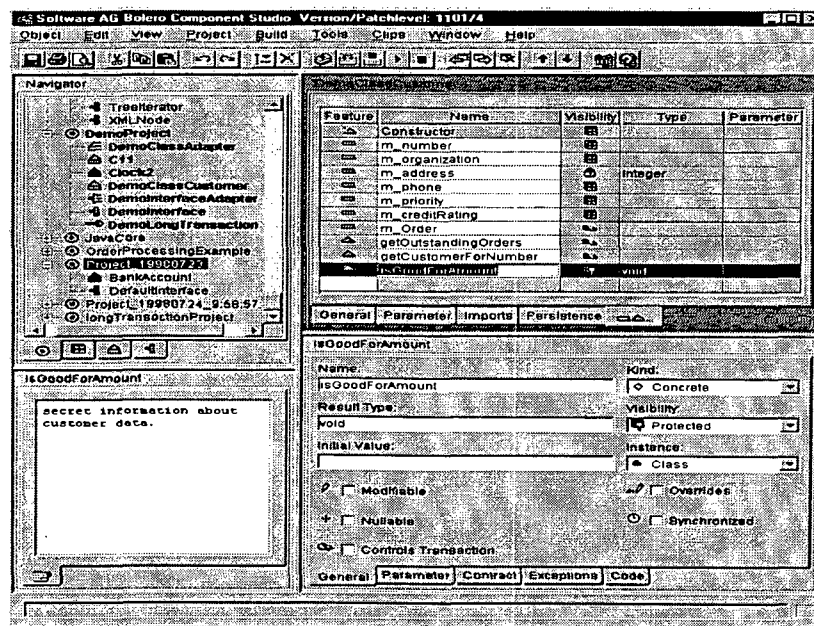
O *Bolero Component Studio* oferece um ambiente gráfico de desenvolvimento baseado em *interface Windows* que permite aos usuários acesso completo a toda informação mantida em seu Repositório orientado a banco de dados. Esta interface é utilizada para a especificação e a modificação de classes e de métodos *Bolero*. Através de uma série de *browsers* o desenvolvedor pode navegar por várias entidades *Bolero*, tais como Projetos, Pacotes e classes. O *Bolero Component Studio* é uma ferramenta visual, grande parte do código do programa não precisa ser escrita mas, ao invés disto, pode ser gerada através de alguns "cliques" do mouse a partir de formulários padronizados. Um editor fácil de usar está disponível para a codificação que for necessária dentro dos métodos

Os componentes externos, como *JavaBeans*, componentes COM, programas escritos em outras linguagens, objetos *activeX*, gráficos e arquivos multimídia são totalmente integráveis através do *Bolero Component Studio*⁶.

O *Bolero Component Studio* organiza os dados do desenvolvimento de acordo como os tipos de elementos. O *Navigator* fornece visões hierárquicas diferentes de todos os componentes da aplicação contidos no Repositório.

O *Bolero Component Studio* consiste de janelas diferentes. Cada janela oferece campos para diferentes visões dos elementos de uma aplicação *Bolero*, conforme mostra a figura abaixo.

Figura 8.4: Interface do *Bolero Component Studio*



Fonte: TEIVELIS, Daniel. Business Oriented Language EnviRONmen. Disponível em: <<http://www.consist.com/br/br405000.html>. Acesso em 12 abril 2000

⁶ SOFTWARE AG Corporation: Bolero Technical Details . Disponível em: <http://www.softwareag.com/bolero/technical/description.html>) Acesso em 20 agosto 2000.

8.2.3 O Bolero GUI Builder

O *Bolero GUI Builder* está integrado ao ambiente de desenvolvimento *Bolero* e suporta a criação de componentes através de uma interface gráfica de usuário. Ele contém um editor para a Geração de componentes gráficos *Bean* tais como: botões, campos de entrada e de saída, listas, barras de menu, etc. As propriedades dos componentes *Bean* podem ser facilmente especificadas utilizando o Editor de Propriedades. O Editor de Eventos ajuda o desenvolvedor a conectar os eventos gráficos à lógica da aplicação. O GUI *Builder* pode ser usado tanto para o modo RPM (*Rapid Prototyping Mode*), ou seja, iniciando o desenvolvimento a partir de um diálogo (desenho da tela), como no modo DDM (*Data Driven Mode*) começando o desenvolvimento pela lógica da aplicação. O *Bolero GUI Builder* é conceitualmente uma “*BeanBox*” que integra a rica funcionalidade de projeto da biblioteca de classes *Java Swing* e que pode ser estendida através da simples adição de *Beans* desenvolvidos pelo cliente, especificados em um arquivo (*pallet file*). Uma vez completados os componentes GUI, o GUI *Builder* pode armazená-lo no repositório.

8.2.4 O Bolero Report Writer

O *Bolero Report Writer* é parte do *Bolero Component Studio*. O *Report Writer* é uma interessante ferramenta visual voltada para a geração de formulários de relatório. O *Report Writer* é 100% *Java* e pode ser executado como um *applet* ou como parte integrante de uma aplicação. Além dos tradicionais elementos de estruturação, normalmente encontrados em ferramentas de geração de relatórios. A aplicação não apenas transfere

estruturas de dados relacionais para o *Bolero Report Writer*, como também transfere estruturas de objetos. Os relatórios podem ser visualizados a partir da tela, impressos ou salvos no formato ASCII (*American Standard Code for Information Interchange*) ou HTML.

8.2.5 Repositório compartilhado

O Repositório orientado banco de dados contém todas as informações relevantes para o desenvolvimento de sistemas, tais como: as definições de Projetos, Pacotes e de classes o *bytecode* gerado pelo compilador *Bolero*, além das referências aos componentes externos.

8.3 Mapeamento objeto-relacional

O Mapeamento Objeto-Relacional é um conceito que permite às aplicações ou aos componentes baseados em *Bolero* interagir com os sistemas de gerenciamento de bancos de dados relacionais para implementar a persistência de objetos. O componente de Mapeamento faz parte do compilador *Bolero* e é responsável pelas seguintes: criação no banco de dados das tabelas necessárias para a representação das classes persistentes, de acordo com o esquema do banco de dados; geração das definições de colunas nas tabelas tomando como base as definições dos atributos correspondentes no ambiente *Bolero*, e mapeamento (conversão) das classes *Bolero* em tipos de dados SQL.

8.4 Linguagem OQL (*Object Query Language*)

Para consultas diretas às bases de dados, o Bolero implementa OQL – *Object Query Language* (Linguagem de Pesquisa de Objeto), um padrão originalmente desenvolvido pelo ODMG (*Object Data Management Group*) para consultas a sistemas de bancos de dados orientados a objetos. Este padrão é amplamente aceito entre os fornecedores deste tipo de sistema. Além de ter sido adotado pelo padrão ISO SQL3, o OQL é parte integrante da linguagem Bolero. Enquanto o JDBC (*Java DataBase Connectivity*) submete consultas SQL a um banco de dados SQL para execução sem verificação, o compilador *Bolero* pode verificar a sintaxe de todas as consultas OQL, eliminando a possibilidade de ocorrência de erros em tempo de execução.

8.5 Transações no Bolero

Como já visto no capítulo 2, o tratamento confiável das transações e a garantia de integridade dos dados resultantes são pré-requisitos básicos para qualquer aplicação. Nos sistemas de aplicação orientados a objetos, o controle de transações pode mostrar-se o elo mais fraco da corrente. O *Bolero* resolve este problema de forma eficiente, efetiva e segura, através de uma série de características inovadoras. O *Bolero* distingue as transações curtas, chamadas “Convencionais”, das grandes transações, denominadas “Transações Longas”, que são utilizadas para modelar todo um processo comercial. Para as transações convencionais. O *Bolero* utiliza os serviços de *resource Managers* como, por exemplo, os sistemas gerenciadores de base de dados compatíveis com o critério ACID (Atomicidade, Consistência, Isolamento e Durabilidade), para o processamento das transações. Em breve, os

componentes *Bolero* também poderão participar de transações distribuídas coordenadas por um gerenciador de transações que suporte o protocolo padronizado *two-phase commit*.

8.5.1 Controle de transações

Uma área crítica do planejamento de desempenho das aplicações em ambiente de produção é o controle de transações. como citado no capítulo 2, entende-se por “transação” (em oposição ao conceito de transação longa) uma unidade lógica de trabalho que atende aos critérios do padrão ACID. É preciso considerar também a otimização dos acessos simultâneos às bases de dados, efetuados por um grande número de usuários concorrentes. Para otimizar os múltiplos acessos paralelos , as transações de acessos a dados devem ser extremamente eficientes e devem evitar, acima de tudo, ser interrompidas por diálogos (interações com o usuário final).

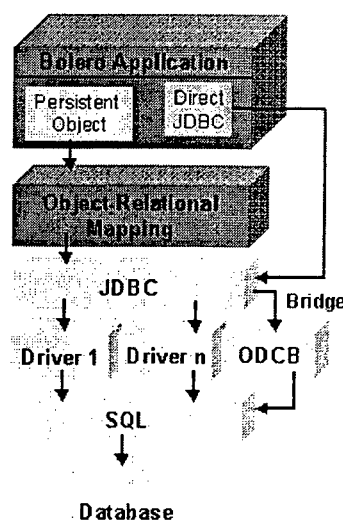
O *Bolero* implementa o seu próprio conceito de transação separando os objetos de controle da transação daqueles objetos que simplesmente fazem parte da transação ⁹. Os objetos de controle das transações são instâncias de classes *Bolero* definidas como responsáveis pelo controle da propriedade e que contém um ou mais métodos com esta funcionalidade. O Uso de um destes métodos indica o início de uma transação (*begin transaction*); um outro método (*end transaction*) sinaliza o fim de uma transação, incluindo um comando *commit* implícito ou um comando *rollback* caso ocorra erro, ou término anormal.

Separar o controle da transação do processamento lógico oferece ainda a vantagem de favorecer a reutilização das classes. As alterações nos objetos

⁹ SOFTWARE AG Corporation: Bolero Application Development for Eletronic Bussiness .
Disponível em: <http://www.softwareag.com/bolero/brochures.htm>. Acesso em 18 agosto 2000.

persistentes só podem ser efetuadas por intermédio de uma transação. Durante a transação, o *Bolero* garante automaticamente a consistência dos objetos envolvidos. Dentro dos limites de uma transação é possível invocar qualquer quantidade de métodos de outros objetos, desde que estes não interfiram no controle de uma transação. As alterações feitas em objetos persistentes durante o curso de uma transação, não são sempre que possível, repassadas ao banco de dados até que a transação tenha sido completada. A associação das execuções de atualizações em um curto período de tempo melhora o desempenho porque reduz o número de chamadas ao banco de dados. A lógica de transação orientada a objetos implementada no *Bolero* suporta sistemas de gerenciamento de banco de dados com estratégias de bloqueio otimistas (*locking*) otimistas ou pessimistas, e em ambos os casos, minimiza o número e a duração dos bloqueios.

Figura 8.5: Objetos persistentes no Bolero



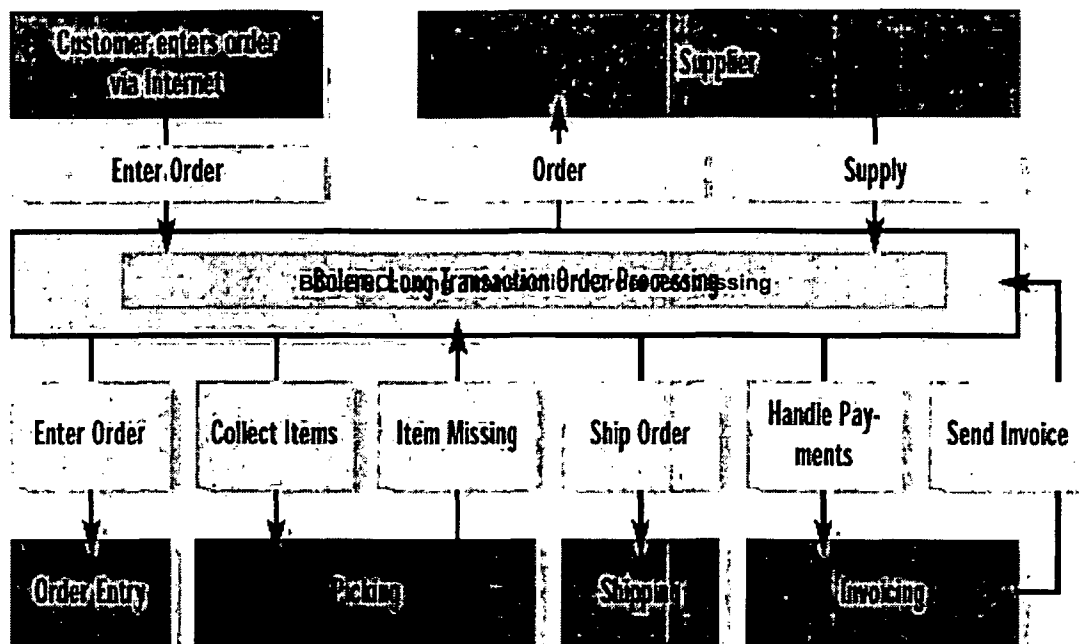
Fonte: TEIVELIS, Daniel. Business Oriented Language EnviRONmen. Disponível em: <<http://www.consist.com/br/br405000.html>. Acesso em 12 abril 2000

8.5.2 Transação longa

Através das transações longas, o *Bolero* oferece um conceito único para a modelagem de processos comerciais complexos de longa duração (*workflows*). Um processo deste tipo pode levar de poucos segundos até vários dias ou semanas para ser concluído. Este tipo de transação pode não estar restrito a aplicações locais, podendo facilmente incorporar processos de outras aplicações. Uma transação longa compreende em conjunto de transações curtas que precisam ser completadas ou desfeitas individualmente. Não há limite de tempo entre a execução de transações individuais, e as transações longas também podem ser aninhadas. As Transações Longas são persistentes, ou seja, elas podem se estender por várias sessões. O *Bolero Application Server* garante seu reinício correto ao início de cada sessão. Diferentemente das transações curtas, as transações longas não podem ser simplesmente canceladas ou desfeitas pelo Gerenciador de Recursos (*Resource Manager*) através do método *rollback*, no entanto, o *Bolero* permite que uma Transação Longa seja reposicionada em qualquer estado desejado. Ao invés dos métodos convencionais de processamento seqüencial, em uma Transação Longa, a transição de um estado bem-definido para o próximo estado é disparado por eventos. A seqüência de processamento é especificada através da definição das mudanças de estado permissíveis.

A execução, por exemplo, do evento "Altera Pedido" não deve ser permitida se a transação longa "Processamento de Pedido" já estiver no estado "Despachado". O mesmo evento ("Altera Pedido"), no entanto, deve ser possível se o estado atual for "Pronto para Enviar".

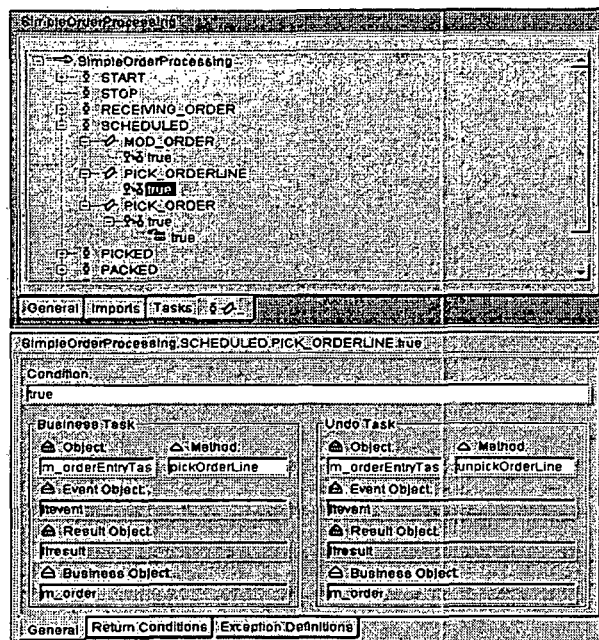
Figura 8.6: Bolero: Transação Longa de Processamento de Pedidos



Fonte: TEIVELIS, Daniel. Business Oriented Language EnviROnmen. Disponível em: <<http://www.consist.com/br/br405000.html>. Acesso em 12 abril 2000

Uma Transação Longa é uma entidade *Bolero* visível através do *Bolero Component Studio*. Ao definirem as transições de estado, os desenvolvedores *Bolero*, auxiliados pelos modelos de projeto, também podem especificar uma ação compensatória para cada estado individual. Esta ação compensatória, se necessário executará uma operação de retorno ao estado anterior (undo), dentro do contexto de Transação Longa. As vantagens das transações longas são: modelagem simples e consistente dos processos comerciais, maior facilidade de navegação entre os componentes da aplicação e reutilização mais fácil de componentes em outros processos.

Figura 8.7 – Editor de Transações Longas



Fonte: TEIVELIS, Daniel. Business Oriented Language EnviROnmen. Disponível em: <<http://www.consist.com/br/br405000.html>. Acesso em 12 abril 2000

As transições de estado de uma Transação Longa podem ser editadas visualmente no editor de Transações Longas como mostra a figura acima. Uma condição de cancelamento “UNDO” pode ser especificada para cada processo individual.

8.6 Conclusão

No decorrer deste último capítulo foram abordadas as principais características de uma ferramenta de mercado, O *Bolero*, que se enquadra na arquitetura de Servidores de Transação, que representa a maneira como é implementada a mais segura das aplicações *Web Banco de Dados* existentes.

Avaliando mais claramente, pode-se afirmar que esta arquitetura é a mais adequada à maioria das aplicações *Web Banco e Dados*. A arquitetura atende e resolve a problemática da manutenção das propriedades ACID.

Logicamente muitas melhorias poderão vir a ser feitas na ferramenta, já a *WWW* está experimentando uma transição em direção a uma nova geração. Muito embora o HTML sobreviva por mais algum tempo como a linguagem de apresentação de páginas na Web, as tarefas mais sofisticadas passarão a ser realizadas com o uso no novo padrão XML (*eXtensible Markup Language*). O XML, por integrar dados e metadados em documentos padronizados, apresenta-se como uma linguagem universal para a transmissão e o armazenamento de dados. Mediante pesquisas pude identificar que novas versões desta ferramenta suportarão XML em ambas as direções: importando documentos XML para aplicações Bolero e criando documentos XML com o *Bolero*.

9 CONCLUSÃO FINAL

O estudo detalhado dos modelos transacionais para Internet visando a integração da *Web* com os Sistemas Gerenciadores de Bancos de Dados, é uma área de pesquisa que ainda está em pleno desenvolvimento. A diversidade de arquiteturas disponíveis para integrar as tecnologias aliada aos diversos problemas transacionais, tem dificultado o desenvolvimento de sistemas para *Web* utilizando Bancos de dados, com funcionalidade capaz de atender aos requisitos essenciais para um perfeito funcionamento e uma transação hoje presente em modernos sistemas de computação de Bancos de dados cliente/servidor.

As pesquisas hoje existentes envolvendo transações que integram a *Web* a Banco de Dados , indicam que ainda não existe uma definição de padrões para os principais aspectos considerados em um ambiente transacional, como as propriedades ACID, controle de recuperação de falhas, controle de atomicidade, controle de concorrência, otimização de acesso à dados, gerenciamento do estado da aplicação, segurança, modelagem, planejamento, portabilidade, devido a complexidade desses itens e a grande rapidez em que aparecem novas tecnologias, os pesquisadores tem apresentado propostas para soluções de itens específicos, uma vez que estes aspectos complementares precisam também ser desenvolvidos e os conflitos resolvidos sob uma visão global.

A contribuição deste trabalho para solucionar estes problemas de modelos transacionais, foi uma abordagem altamente genérica, ou seja, a delimitação dos problemas existentes nos modelos transacionais das

aplicações na *Web*, baseados em requisitos funcionais das atuais transações em Banco de Dados, aliada ao estudo das arquiteturas existentes.

As aplicações *Web* BD devem ser baseados em modelos transacionais que solucionem os dois principais problemas decorrentes da integração de tecnologias tão distintas e independentes como são a *Web* e os Sistemas Gerenciadores de Bancos de Dados: a natureza *stateless* da *Web* e o oposto a natureza *stateful* de sistemas de BD e o suporte transacional dos Bancos de Dados baseados em propriedades ACID em contraste ao suporte não transacional da *Web*. Fundamentalmente, a solução deste problema passa necessariamente pelo *gateway* que integra as tecnologias. O requisito básico do *gateway* é manter o estado da aplicação do usuário através do gerenciamento da conexão com o Banco de Dados, implementando funcionalidades de segurança, concorrência, atomicidade e isolamento baseado em sessão contínua do usuário da aplicação em cooperação com as funcionalidades já existentes no SGBD.

A pesquisa sobre as arquiteturas existentes possibilitou uma extensão das taxonomias das arquiteturas de *gateways* transacionais *Web* Banco de Dados existentes bem como a comparação entre as arquiteturas baseados nos requisitos levantados ao longo do trabalho. Como ressaltado, o núcleo central para o funcionamento das transações na *Web* e banco de Dados, tanto no contexto *Internet* como no contexto *Intranet*, são os *gateways*. Esses *gateways* foram classificados em oito arquiteturas distintas, onde foi enfatizado o comportamento das transações nas aplicações que envolvam Bancos de Dados, descritas ao longo do trabalho para cada arquitetura em particular. Foram identificadas vantagens e inconvenientes de cada arquitetura, visando um melhor aproveitamento transacional das tecnologias para a construção e desenvolvimento de aplicações em que existam *Web* sobre Bancos de Dados.

Por fim o estudo de caso possibilitou identificar elementos que levam a um conflito com a tendência no desenvolvimento de transações para *Internet/Intranet* baseada em BD nos últimos anos: os *Programas executáveis CGI*, apresenta diversos problemas funcionais e não soluciona os dois principais problemas decorrentes da integração das tecnologias: a natureza

stateless da Web e a perda de diversas funcionalidades de Bancos de dados decorrentes da não satisfação das propriedades ACID no ambiente transacional. E além do estudo de caso foi analisado neste trabalho, uma ferramenta de mercado o Bolero , que é baseada na arquitetura de *Servidores de Transações*, que pelo que tudo indica , tende a ser a arquitetura mais utilizada no futuro próximo. Esta arquitetura já apresenta a característica *stateful* no ambiente de integração, e possibilita um bom suporte as transações ACID sobre a Web , e é baseada em arquiteturas para transações distribuídas, utilizando CORBA e DCOM.

9.1 Contribuições da dissertação

As principais contribuições desta dissertação são:

- a) Levantamento dos principais problemas em aberto no ambiente de integração *Web* Banco de Dados e como alguns destes problemas podem sendo resolvidos sem ser necessário uma alteração ou limitação das tecnologias envolvidas.
- b) Apresentação de um conjunto de requisitos funcionais dos *gateways* que integram as tecnologias, principalmente no que diz respeito ao problema *stateless* da Web e da extensão das propriedades ACID de Banco de Dados ao ambiente de integração. Os poucos trabalhos que abordam requisitos para o desenvolvimento de aplicações *Web* Banco de Dados não tratam a questão sob o ponto de vista funcional, como proposto aqui.
- c) Uma extensão das taxonomias existentes para a classificação dos inúmeros *gateways* *Web* Banco de Dados existentes, tendo como base as principais funcionalidades dos SGBDs. As oito arquiteturas de integração

identificadas e descritas neste trabalho possibilitam comparar os diversos gateways existentes segundo a categoria em que se enquadram, facilitando o estudo de requisitos a serem satisfeitos pela aplicação. Neste sentido, foi apresentado o estado da arte da integração das tecnologias sob uma perspectiva da arquitetura dos gateways e suas implicações funcionais. O quadro comparativo das arquiteturas identificadas permite avaliar os gateways segundo as funcionalidades desejadas, o que facilita a definição do melhor gateway que se deve utilizar para uma determinada aplicação.

- d) Uma análise funcional qualitativa da principal arquitetura de transação que envolvem o SGBD e a *Web* existente atualmente. A arquitetura Programas Executáveis CGI foi uma das primeiras arquiteturas implementadas usada para se efetivar a integração das tecnologias, como já previa o projeto *Web* quando de seu surgimento. No entanto, argumentou-se que se a aplicação *Web* Banco de Dados necessitar de requisitos funcionais semelhantes às atuais implementações de aplicações cliente/servidor de Banco de Dados, esta arquitetura é inadequada. A arquitetura que tende a ser a mais utilizada, e que possibilita um bom suporte às transações ACID, é a arquitetura de *Servidor de Transação*.

- e) Descrição dos fundamentos básicos relacionados à *Web* e ao ambiente de integração segundo uma intensa pesquisa de trabalhos relacionados no que diz respeito ao desenvolvimento de aplicações *Intranet/Internet*. O levantamento bibliográfico sobre o assunto é extenso e de boa qualidade técnica, principalmente se considerarmos a novidade e atualidade da pesquisa desenvolvida nesta dissertação. Ainda é muito difícil encontrar artigos técnicos sobre o assunto.

9.2 Trabalhos futuros

Considerando que a pesquisa envolvendo modelos de transações na Web e Bancos de Dados , é relativamente complexa, que há ainda muito a ser explorado e muitas questões ainda devem ser estudadas para que esse ambiente se torne confiável e possa ser utilizado mais intensamente. Em função disso, vários outros trabalhos podem ser desenvolvidos nesta área com o objetivo de melhorar, levando a um nível atual e de mais qualidade as transações na Internet sobre um Banco de Dados.

Neste sentido os trabalhos futuros decorrentes desta dissertação são os seguintes:

- a) Um estudo detalhado sobre segurança de transações na *Internet* e nas *Intranets* através de Sistemas Gerenciadores de Bancos de Dados. Requisitos de segurança especialmente direcionados para aplicações Web Banco de Dados ainda não estão claros. Questões como, o que, por que e como controlar, devem ser melhor direcionadas. Em particular, devem ser direcionados requisitos de segurança no ambiente Web para melhor garantir o acesso, manutenção e distribuição das informações armazenadas em SGBDs.
- b) Uma avaliação e implementação de modelos transacionais utilizando a Linguagem PHP, comparando com uma diversas arquiteturas apresentadas nesta dissertação.
- c) Uma implementação de uma aplicação utilizando o *Bolero*, a ferramenta de Mercado que se encaixa na arquitetura de transações denominada *Servidor de Transações*, avaliando com grande ênfase as características de gerenciamento de longas transações, existente na ferramenta, e realizar uma comparação com as ferramentas de mercado concorrentes ao *Bolero*.

- d) Um estudo comparativo do comportamento de diversos Bancos de Dados de mercado e os modelos transacionais abordados nesta dissertação, parece ser promissor a investigação de um modelo de transações mais adequado especificamente ao ambiente de integração *Web* e Banco de Dados, no que diz respeito ao desenvolvimento de aplicações. Como apontado ao longo do capítulo 4, é possível identificar diversos elementos que sugerem que o modelo de transações planas dos atuais SGBDs pode não ser o mais adequado dentro do contexto *Web*. Particularmente interessante seria escolher uma das arquiteturas de integração existentes, propor um modelo formal de gerenciamento do estado da aplicação e implementá-lo em cooperação com um SGBD que suporte o novo modelo de transação identificado como o mais adequado.

Para finalizar , deve-se ressaltar que embora as duas tecnologias envolvidas neste trabalho, a *Web* e o Banco de Dados, sejam tecnologias independentes e , de certa forma , tenham sido projetadas em momentos diferentes para atuarem em áreas distintas, os frutos decorrentes dos experimentos que as integram já apresentam resultados promissores , o que justifica a continuidade do estudo e da pesquisa no tema desenvolvido nesta dissertação.

REFERÊNCIAS BIBLIOGRÁFICAS

ABITEBOUL, S. Querying Semi-Structured Data, **Proceedings of Intl. Conference on Database Theory (ICDT)**, 1997, p.1-18. Disponível em: <<http://www-db.stanford.edu/pub/papers/icdt97.semistructured.ps>>. Acesso em: 24 novembro 1997.

ABITEBOUL, S. ; VIANU, V. Queries and Computation on the Web, **Proceedings Intl. Conference on Database Theory (ICDT)**, 1997, p. 262-275. <http://www-db.stanford.edu/pub/papers/icdt97.www.ps> .

ADUNUTHULA, S. ; ANAND M. **Oracle Web Request Broker API**. Workshop Program – Fifth Intl. World Wide Web Conference. Paris, 1996. Disponível em: <<http://www.cs.vu.nl/~eliens/WWW5/papers/Broker.html>> . Acesso em: 12 maio 1998

ALLAIRE Corporation. **ColdFusion user's guide**,1997. Disponível em: <<http://www.allaire.com/products/coldfusion/20/userguide/index.htm>>. Acesso em: 15 dezembro 1997.

AMERICA Online Inc. **AOL Server - Administrator's Guide**, 1997. Disponível em: <<http://www.aolserver.com/server/docs/2.1/html/admin.htm>>. Acesso em: 07 janeiro 1998.

ANAND, M. ; CHIEN, E. ; CONDAMOOR, R. ; MATHUR, A. ; ADUNUTHULA, S. ; CHOU, S. ; NAKHODA, S. **The Web Request Broker: A Framework for Distributed Web-Based Applications**. Oracle Corporation, 1997. Disponível em: <http://www.olab.com/www6_1/paper.html>. Acesso em: 02 dezembro 1997.

AROCENA , G. O. ; MENDELZON, A. O. ; MIHAILA ,G. A. **Applications of a Web Query Language**. Sixth Intl. World Wide Web Conference. Santa Clara, California, EUA, 1997. Disponível em : <<http://www6.nttlabs.com/HyperNews/get/PAPER267.html>>. Acesso em: 14 abril 1998.

ATKINSON, M. P. ; DAYNÈS, L. ; JORDAN, M. J. ; PRINTEZIS; T. ; SPENCE, S. **An Orthogonally Persistent Java**. ACM SIGMOD Record, v. 25, n. 4, p. 68-75, 1996.

BEA System. **BEA Jolt**, 1997. Disponível em: <<http://www.beasys.com/products/jolt/index.htm>>. Acesso em: 07 março 1998.

BEA System. **BEA Jolt Whitepaper**, 1997b. Disponível em: <<http://www.beasys.com/products/jolt/joltwhit.htm>>. Acesso em: 07 março 1998.

BEA System. **BEA Jolt User's Guide**, 1997c. Disponível em: <<http://beademo1.beasys.com/jolt/joltdoc/userman/>>. Acesso em 07 março 1998.

BENETT, G. **Intranets: Como implantar com sucesso na sua empresa**. São Paulo: Campus, 1996.

BERGHEL, H. The client's side of the World-Wide Web. **Communications of the ACM**, v. 39, n. 1, p. 30-40. 1996.

BERNSTEIN, P. A. ; HADZILACOS, V. ; GOODMAN , N. **Concurrency Control and Recovery in Dtabase Systems**. Addison–Wesley, 1987.

BINA, E. ; JONES, V. ; MCCOOL, R. ; WINSLETT, M. Secure Acces to Data Over the Internet. **Proceedings of the third ACM/IEEE Intl. Conf. on Parallel and Distributed Information Systems**. Austin, texas, 1994. Disponível em: <<http://bunny.cs.uiuc.edu/CADR/WinslettGroup/pubs/SecureDBAccess.ps>>. Acesso em: 29 setembro 1998.

BJORK, L. A. Recovery Scenario for a DB/DC System. **Proceedings of ACM National Conf.** Atlanta, Ga. Agosto 1973.

BRICKEI, E. ; GEMMELL, P. ; KRAVITZ, D. Trustee-based Tracing Extensions to Anonymous Cash and the Making of Anonymous Change. In **Proceedings of the Sixth ACM-SIAM Symposium on Discrete AlgoriAlgorithms**, p. 457-466, 1995.

CHRISTHOPHIDES, V. ; ABITEBOUL, S. ; CLUEt , S. ; SCHOLL, M. From structured documents to novel query facilities, **Proceedings of ACM SIGMOD'94**, 1994, p. 313-324.

COMPUTER Associates International. **Ingres Database Administrator's Guide**, 1995.

COMER, D. E. ; STTEVENS, D. L. Internetworking with TCP/IP - Volume III. **Client-server programming and applications**. Prentice-Hall, 1994.

CRAIG, J. **Allaire's Cold Fusion**. Allaire Corporation, 1996. Disponível em: <<http://www.innenergy.com/idem/v1n3/idm0896b.html>>. Acesso em 18 janeiro 1999.

CRUZ, R. A. **Data Recovery in IBM DATABASE 2**. IBM Systems. J 23, n.2, 1984.

DATARAMP Corporation: **DataRamp home page**, 1997. Disponível em: <<http://www.dataramp.com/notdoc/contents.htm>> Acesso em 12 julho 1999

DATE, C. J. **A Guide to Ingres**. Addison-Wesley Publishing Company, 1987.

DATE, C. J. **Distributed Database: A Closer Lock**. Addison-Wesley Publishing Company, 1992.

DATE, C. J. **Introdução a Sistemas de Bancos de Dados**. Campus, São Paulo, 2000.

DAVIES, C. T. **Data Processing Spheres of Control**. IBM Systems. J. 17, n. 2, 1978.

DENNY, B. R. **SSI, CGI, API, or SSS, Choosing the Right Tool for the Job**, 1996. Disponível em: <<http://solo.dc3.com/white/extending.html>>. Acesso em: 03 agosto 1999

DOBSON, S. A. ; BURRILL, V. A. **Lightweight Database**. The Third Intl. World Wide Web Conference, Tools and Applications. Darmstadt, Germany, 1995. Disponível em: <<http://www.igd.fhg.de/www95.html>>. Acesso em 13 julho 1999.

DUAN, N. N. **Distributed Database Access in a Corporate Environment Using Java**. Fifth Intl. World Wide Web Conference. Paris, 1996. Disponível em: <http://www5conf.inria.fr/fich_html/paper-sessions.html>. Acesso em: 26 julho 2000.

EICHMANN, D. ; MCGREGOR, T. ; DANLEY, D. **Integrating Structured Databases into the Web: The MORE system**. The First Intl. Conf. on the

World Wide Web, CERN, Geneva, 1994. Disponível em:

<<http://ricis.cl.uh.edu/eichmann/www94/MORE/MORE.html>>. Acesso em : 26 junho 2000

ELMASARI, R. ; NAVATHE, S. B. **Fundamentals of Database Systems**. Second Edition. The Benjamin/Cummings Publishing Company, Inc. 1994.

FRANK, M. Database and the Internet. **DBMS On-Line Magazine**. December, 1995. Disponível em: <<http://www.dbmsmag.com/f19512.html>>. Acesso em: 15 fevereiro 2000.

GLASGOW, University. **The PJava Project - Orthogonal Persistence for Java**. Department of Computing Science, University of Glasgow, 1997. Disponível em: < <http://www.dcs.gla.ac.uk/pjava/>>. Acesso em: 28 dezembro 1999.

GOISLING, J. ; MCGILTON, H. **The Java (tm) Language Environment: A White Paper**. Sun Microsystems, 1996. Disponível em: <http://java.sun.com/doc/language_environment>. Acesso em: 03 outubro 1999.

GRAY, M. **Web Growth Summary**. Student Information Processing Board, 1997. Disponível em: <<http://www.mit.edu/people/mkgray/net/web-growth-summary.html>>. Acesso em: 16 maio 1998.

GRAY, J. ; REUTER, A. **Transaction Processing: Concepts and Techniques**. Morgan Kaufmann Publishers, Inc. 1993.

HADJEFTHYMIADES, S. I. ; MARTAKOS, D. I. **A generic framework for the deployment of structured databases on the World Wide Web**. Fifth Intl. World Wide Web Conference. Paris, 1996. Disponível em:

<http://www5conf.inria.fr/fich_html/paper-sessions.html>. Acesso em: 01 setembro 1998.

HADJEFTHYMIADES, S. P. ; MARTAKOS, D. I. **Improving the performance of CGI compliant database gateways**. Sixth Intl. World Wide Web Conference, Santa Clara, EUA, 1997. Disponível em: <<http://www6.nttlabs.com/papers/PAPER44/PAPER44.html>>. Acesso em : 16 outubro 2000.

HUGHES, M. **JavaBeans vs. ActiveX: A developer weighs competing component frameworks**. **JavaWorld On-line Magazine**, 1997. Disponível em: <<http://www.javaworld.com/jw-03-1997/Old/jw-03-avb-tech-moc.html>>. Acesso em : 12 dezembro 1997.

HUNTER, A. ; FERGUSON, R. I. ; HEDGES, S. **SWOOP: An Application Generator for ORACLE/WWW Systems**. **World Wide Web Journal**. v. 1, n. 1, 1996. Disponível em: <<http://www.w3.org/pub/WWW/Journal/1/hunter.207/paper/207.html>>. Acesso em: 16 maio 1998.

IBM Corporation. **DB2 World Wide Web Connection**, 1997. Disponível em: <<http://www.software.ibm.com/data/db2/db2wgafs.html>> Acesso em: 12 dezembro 1997.

INGHAM, D. ; CAUGHEY, S. ; LITTLE, M. **Fixing the 'Broken-Link' problem: the W3Objects approach**. Fifth Intl. World Wide Web Conference. Paris , 1996. Disponível em: <http://www5conf.inria.fr/fich_html/papers/P32/Overview.html> Acesso em: 03 outubro 1999.

INFORMIX Corporation: **Universal Web Connect**, 1997. Disponível em: <<http://www.informix.com/informix/corpinfo/zines/whitpprs/wpentcom.pdf>>
Acesso em: 12 dezembro 1997.

INTERNET Systems. Internet tools product chart. **DBMS On-line Magazine**, 1996. Disponível em: <<http://www.dbmsmag.com/9610i.html>>. Acesso em 16 maio 1998.

KIM , P. A Taxonomy on the Architecture of Database Gateways for the Web. **Technical Report, Database Laboratory Dept. of Information Communications Engineering**, Chungnam National University, 1996. Disponível em: <<http://grigg.chungnam.ac.kr/projects/UniWeb/documents/taxonomy/text.html>>. Acesso em: 05 janeiro 1999 .

KIM, E. E. **CGIHTML version 1.66**, 1997. Disponível em: <<http://www.eekim.com/software/cgihtml/>>. Acesso em: 28 dezembro 1999.

KONOPNICK, D. ; SHMUELI, O. **W3QS: A query system for the World Wide Web**. In **Proceedings of VLDB**, p. 54-65, 1995.

KOTH, H. F. The Double Life of the Abstraction Concept: Fundamental Principle and Envolving System Concept. **Proceedings of 21st Int. Conf. On Very large Data Bases**. Zurique, Suíça. Setembro , 1995.

KORTH, H. F. ; SILBERCHATZ, A. **Database System Concepts**. Second Edition, McGraw-Hill, 1997.

LIFSCHITZ, S. ; LIMA, I. N. **Arquitetura de Integração Web SGBD: um estudo do Ponto de vista de Sistemas de Banco de Dados**. XXV Software and Hardware Seminar (SEMISH 98) , Belo Horizonte, 1998.

LAGROW, C. Java fires up its database engines. **JavaWorld On-line Magazine**, 1996. Disponível em: <<http://www.javaworld.com/javaworld/jw-05-1996/jw-05-cszone.html>>. Acesso em: 16 outubro 2000.

LAWRENCE, S. ; GILES, C. L. **Searching the World Wide Web**. Science, v. 280, n. 4, p. 98-100, 1998.

LEE, D. L. ; YUWONO, B. **WISE: A World Wide Web Resource Database System**. IEEE Transactions on Knowledge and Data Engineering, v. 8, n. 4, p. 548-554, 1996.

LINTHICUM, D.S. **The JDBC Connection**. Internet Systems, 1996. Disponível em: < <http://www.dbmsmag.com/9610i06.html>>. Acesso em: 01 setembro 1998.

LITTLE, M. C. ; SHIVASTAVA, S. K. Java Transactions for the Internet. **Special Issue Journal**, v. 5, n. 4, p. 156-157, 1998.

LITTLE, M. C. ; SHIVASTAVA, S. K. ; CAUGHEY, S. J. ; INGHAM, D. B. **Constructing Reliable Web Applications Using Atomic Actions**. Sixth Intl. World Wide Web Conference, Santa Clara, EUA, 1997. Disponível em: <<http://www6.nttlabs.com/HyperNews/get/PAPER12.html>>. Acesso em: 12 outubro 1998.

LOMET, D. ; TUTTLE, M. R. Redo Recovery after System Crashes. **Proceedings of 21st Int. Conf. On Very Large Data Bases**. Zurique, Suíça. Setembro, 1995.

LYON, J. ; Evans, K. ; Klein, J. **Transaction Internet Protocol**. Internet Draft, 1997. Disponível em: <<http://www.ietf.org/ids.by.wg/tip.html>>. Acesso em: 26 setembro 2000.

MAIA, A. C. C. ; PINHEIRO, A. S. **DBWeb: Um sistema para conectividade a Banco de Dados via WWW.** Monografia em Ciência da Computação, Departamento de Computação, Universidade Federal Fluminense, Rio de Janeiro, 1996.

MAIER, D. ; ULLMAN, J. D. **Maximal objects and the semantics of universal relation databases.** ACM TODS, v. 8, n. 1, p. 1-14, 1983.

MELLMAN, J. Object Databases on the Web. **Web Techniques Magazine**, v. 1, n. 6, p. 42-45, 1996.

MENDELZON, A. ; MIHAILA, G. ; MILO, T. Querying the World Wide Web. **Proceedings of Intl. Conference on Parallel and Distributed Information Systems (PDIS)**, p. 80-91, 1996. Disponível em:
<<ftp://db.toronto.edu/pub/papers/pdis96.ps.gz>>. Acesso em : 16 outubro 2000.

MICROSOFT Corporation. **ActiveX Tutorial Samples**, 1997. Disponível em:
<<http://www.microsoft.com/intdev/activex/tutorial/>>. Acesso em: 25 agosto 1998.

MICROSOFT Corporation. **Internet Server API Reference**, 1997b. Disponível em: <<http://microsoft.com/win32dev/apiext/isapiref.htm>> Acesso em: 25 agosto 1998.

MICROSOFT Corporation. **Internet Server API Overview**, 1997c. Disponível em: <<http://microsoft.com/win32dev/apiext/isapimrg.htm>>. Acesso em: 25 agosto 1998.

MICROSOFT Corporation. **IIS Installation an planning Guide**. Reference IIS, 1997d. Disponível em: <<http://198.105.232.4/infoserv/docs/program.htm>>. Acesso em 25 agosto 1998.

MICROSOFT Corporation. **VBScript Documentation**, 1997e. Disponível em: <<http://www.microsoft.com/vbscript/us/vbsmain/vbsmaint.htm>>. Acesso em: 25 agosto 1998.

MICROSOFT Corporation. **Internet Explorer**, 1997f. Disponível em: <<http://www.microsoft.com/iis/default.asp>>. Acesso em 25 agosto 1998.

MICROSOFT Corporation. **Microsoft Server Write Paper**, 1997. Disponível em: <<http://www.microsoft.com/transaction/learn/mtswp.htm>>. Acesso em: 25 agosto 1998.

MSA-INFOR Sistemas de Automação Ltda. **CorpWeb Server - Um servidor Web para o ambiente Unisys**, 1997g. Disponível em: <<http://www.msainfor.com.br/icorpweb.htm>>. Acesso em: 23 novembro 1997.

MOLINA, H. G. ; SALEM, K. Sagas. **Proceedings of 1987 ACM SIGMOD Int. Conf. On Management of Data**. San Francisco, California. Maio, 1987.

NATIONAL Center for Supercomputing Applications. **Server Side Includes (SSI) – NCSA HTTPd**. University of Illinois at Urbana-Champaign, 1997. Disponível em: <<http://hoo.hoo.ncsa.uiuc.edu/docs/tutorials/includes.html>>. Acesso em: 13 agosto 1998.

NATIONAL Center for Supercomputing Applications. **Common Gateway Interface Overview**. University of Illinois at Urbana-Champaign, 1997b. Disponível em: < <http://hoo.hoo.ncsa.uiuc.edu/cgi/overview.html>>. Acesso em: 13 agosto 1998.

NATIONAL Center for Supercomputing Applications. **NCSA HTTPd. A WWW Server**. University of Illinois at Urbana-Champaign, 1997. Disponível em: < <http://hoo.hoo.ncsa.uiuc.edu/docs/Overview.html>>. Acesso em 13 agosto 1998.

NETSCAPE Communications Corporation. **Extensions to HTML 2.0**, 1997. Disponível em: <http://www2.netscape.com/assist/net_sites/html_extensions.html>. Acesso em: 23 agosto 1998.

NETSCAPE Communications Corporation. **Netscape LiveWire – Introduction**, 1997b. Disponível em: <<http://developer.netscape.com/library/documentation/livewire/index.html>> Acesso em: 23 agosto 1998.

NEXT Software. **WebObjects: Dynamic Applications for the Web**, 1997. Disponível em: <<http://www.next.com/WebObjects/WebObjectsDynamic.html>> Acesso em 12 dezembro 1997.

NEWMAN, D. ; FOWLER, J. **Web Servers: Digging Into Corporate Data**, Data communications on the Web, 1996. Disponível em: <http://www.data.com/lab_tests/digging.html> Acesso em 12 janeiro 1997.

NGUYEN, T. ; SRINIVASAN, V. Accessing Relational Database from the World Wide Web. **Proceedings of ACM SIGMOD Intl. Conf. on Management of Data**, p. 529-540 Montreal, Canada, 1996.

NORTH, K. Database Programming with OLE and Active X, **DBMS On-line Magazine**, 1996. Disponível em: <<http://www.dbmsmag.com/9611d14.html>>. Acesso em: 22 dezembro 1997

NORTH, K. ODBC Branches Out. **DBMS On-line Magazine**, 1996b. Disponível em: <<http://www.dbmsmag.com/9604d52.html>> Acesso em: 22 dezembro 1997.

NORTH, K. Building Web Databases. **Web Techniques Magazine**. v.1, n. 6, p. 34-41, 1996.

NORTH, K. ODBC Branches Out. **DBMS On-line Magazine**, 1996c. Disponível em: <<http://www.dbmsmag.com/9604d52.html>>. Acesso em: 22 dezembro 1997.

O2 Corporation: **O2Web**, 1997. Disponível em: <<http://www.O2tech.fr/>>. Acesso em: 12 dezembro 1997

OLYMPIA, P. L. Cold Fusion 1.5 and Autobahn. **DBMS On-line Magazine**, 1996. Disponível em: <<http://www.dbmsmag.com/9607d08.html>> Acesso em: 28 dezembro 1997.

OMG Object Management Group. **Object Management Group**, 1997. Disponível em: < <http://www.omg.org/> > Acesso em: 25 novembro 1997.

OMG Object Management Group Inc. **CORBA 2.0/IIOP Specification**, 1997b. Disponível em: <<http://www.omg.org/corba/corbiiop.html>> Acesso em: 25 novembro 1997.

ORACLE Corporation. **Executive Overview of Oracle Web Application Server**. White Paper, 1997. Disponível em: <http://www.olab.com/products/was/html/exec_overview.html>. Acesso em: 23 abril 1997.

ORACLE Corporation. **Oracle WebServer 3.0 Transactions**, White Paper, 1997b. Disponível em: <[http://www.oracle.com/products/websystem/webserver/pdf/ws3_transactions.p](http://www.oracle.com/products/websystem/webserver/pdf/ws3_transactions.pdf)
[df](http://www.oracle.com/products/websystem/webserver/pdf/ws3_transactions.pdf)>. Acesso em 24 abril 1997.

ORACLE Corporation. **The Transactional Web Server for Business-Critical Applications**, White Paper, 1997. Disponível em: <http://www.olab.com/products/was/html/trans_wp.html> Acesso em: 24 abril 1997.

ORACLE Corporation. **Oracle WebServer 3.0**, White Paper, 1997d. Disponível em:

<http://www.oracle.com/products/websystem/webserver/html/ows3_whitepaper.html>. Acesso em: 24 abril 1997.

ÖZSU, M.; VALDURIES, P. **Principles of Distributed Database Systems**, Prentice Hall, 1991.

PEERROCHON, L. **Translation servers: Gateways between stateless and stateful information systems**, Institut für Informations systeme, ETH Zurich, Technical report, PA-nsc94, 1994. Disponível em: <<http://www.inf.ethz.ch/departement/IS/ea/publications/nsc94.html>>. Acesso em: 02 abril 1998.

PERROCHON, L. ; FISCHER, R. IDLE. **Unified W3-Access to interactive servers**. The Third Intl. World-Wide Web Conference (WWW'95), Darmstadt, 1995. Disponível em: <<http://www.inf.ethz.ch/departement/IS/ea/publications/www95.html>>. Acesso em: 02 abril 1998.

PLAIN, S. W. **Web Database Tools**, **PC Magazine**, v.15, n.15, p. 205-256. September, 1996.

POWERSOFT Corporation. **Building Internet Applications with PowerBuilder 5.0**, 1997. Disponível em: <http://www.powersoft.com/products/devtools/pb50/bld_ipb.html>. Acesso em: 12 dezembro 1997.

RAHMEL, D. **Comparing JavaScript and VBScript**. Internet Systems, 1996. Disponível em: < <http://www.dbmsmag.com/9610i07.html>>. Acesso em: 15 julho 1998

RAHMEL, D. **Systems and Methodologies for Identifying and Protecting Weak Spots in Your Web-Enabled Database**. Internet Systems, 1997. Disponível em: <<http://www.dbmsmag.com/9704i03.html>>. Acesso em: 15 julho 1998.

REED, P. Using ODBC to access nontabular data. **DBMS On-line Magazine**. April, 1996. Disponível em: <<http://www.dbmsmag.com/9604d54.html>>. Acesso em: 24 novembro 1997.

RICHARDS, R. Stretching the OLE Net. **LAN On-Line Magazine**, May, 1996. Disponível em: <<http://www.lanmag.com/9605/9605ole.htm>>. Acesso em: 02 dezembro 1997.

ROY, M. ; EWALD, A. Inside DCOM. **DBMS On-line Magazine**, April, 1997. Disponível em: < <http://www.dbmsmag.com/9704d13.html>>. Acesso em: 24 junho 1999.

SHAH, R. Integrating Databases with Java via JDBC. **JavaWorld On-line Magazine**, 1996. Disponível em: <http://www.cs.rutgers.edu/~shklar/web-legacy/www5_tutorial.html>. Acesso em: 15 junho 1997.

SHKLAR, L. **Web Access to Legacy Data, The Fourth Intl. Conf. on the World Wide Web**. Boston, USA, 1995. Disponível em: <http://www.cs.rutgers.edu/~shklar/web-legacy/www5_tutorial.html>. Acesso em: 13 janeiro 1997.

SHOFFNER, M. JavaBeans vs ActiveX: Strategic Analysis. **JavaWorld On-line Magazine**, 1997. Disponível em:< <http://www.javaworld.com/javaworld/jw-02-1997/jw-02-activex-beans.html>>. Acesso em: 12 agosto 1999.

SPIDER Technologies. Inc.: **NetDynamics Programmer's Guide**, 1997. Disponível em: <<http://www.netdynamics.com/support/manuals/nd21/API/toc.htm>>. Acesso em: 03 setembro 1999.

SPIDER Technologies. Inc.: **NetDynamics - White Paper: Delivering Solutions for the Web-centric Enterprise**, 1997. Disponível em: <<http://www.netdynamics.com/about/whitepaper.html>>. Acesso em : 03 setembro 1999.

SUN Microsystems. **JDBC Home Page**, 1997. Disponível em: <<http://splash.javasoft.com/jdbc/>>. Acesso em: 15 junho 1997.

SYBASE Corporation. **Sybase web.sql On-Line Assistance Menu**, 1997. Disponível em: <<http://www.sybase.com/products/internet/websql/>>. Acesso em: 15 junho 1997.

SYBASE Corporation. **jConnect for JDBC Whitepaper**, 1997. Disponível em: <<http://www.sybase.com/products/internet/jconnect/jdbcwpaper.html>>. Acesso em : 15 junho 1997.

TELFORD, J. M. Allaire Cold Fusion 2.0 professional. **DBMS On-line Magazine**, February, 1997. Disponível em: <<http://www.dbmsmag.com/9702d08.html>>. Acesso em: 12 abril 1998.

USENIX Association. Proceedings of the First USENIX Workshop on Electronic Commerce, Julho de 1995

XOPEN Company. **X/OPEN Guide: Distributed Transaction Processing. Reference Model, Version 2**. X/OPEN Company Ltda., 1993

XOPEN Company. **X/OPEN CAE Specification: Distributed Transaction Processing**. The TxRPC Specification, X/OPEN Company Ltda., 1995

WEBSITE Professional. **WebSite API 1.1 SDK Introduction and Overview**, 1997. Disponível em: <<http://solo.dc3.com/wsapi/>>. Acesso em: 12 abril 1998.

WEBSITE Professional. **Windows CGI Interface**, 1997b. Disponível em: <<http://solo.dc3.com/wsdocs/32demo/windows-cgi.html>>. Acesso em: 12 abril 1998.

WEBQUEST Support and Information Center: **SSI+2.0 Reference**, 1997. Disponível em: <<http://webquest.questar.com/reference/ssi/ssi+20ref.sht>>. Acesso em : 12 abril 1998.