Construção de um Servidor WEB com Enfoque Transacional

Eng. Waldemir Cambiucci
waldemir@larc.usp.br
LARC - Laboratório de Arquitetura e Redes de Computadores
Escola Politécnica da Universidade de São Paulo

Prof. Dr. Wilson Vicente Ruggiero
wilson@larc.usp.br
LARC - Laboratório de Arquitetura e Redes de Computadores
Escola Politécnica da Universidade de São Paulo

RESUMO

Não se pode criar aplicativos de missão crítica sem gerenciar os programas (ou processos) que operam sobre os dados. Este é o motivo por que Monitores Transacionais devem acompanhar todo banco de dados de missão crítica. Os monitores transacionais gerenciam processos e orquestram programas através da divisão de aplicativos complexos em peças de código chamadas transações.

A utilização crescente de Servidores WEB com tratamento de programas CGIs pode ser observada atualmente. Neste contexto, foi adotado o problema típico de criação de CGIs - Common Gateway Interface - para exercício de desenvolvimento, onde algumas funções importantes de monitores transacionais permitem ganhos no gerenciamento e criação de CGIs para aplicações típicas presentes na internet.

Este trabalho está dividido em duas partes: na primeira parte, apresentam-se os conceitos relacionados com o processamento transacional, enfocando os monitores transacionais como solução para ambientes cliente/servidor. Na segunda parte, apresenta-se um Servidor WEB com enfoque transacional, onde são implementadas algumas funções típicas de monitores para o tratamento de aplicações CGIs.

Serão observados ainda os benefícios da utilização do enfoque transacional na implementação de um servidor Web e, como subproduto do estudo, serão descritos os benefícios da API - Application Program Interface - criada para a implementação do projeto, concebido e desenvolvido utilizando orientação a objetos.

1. Introdução

O principal objetivo deste trabalho é a descrição do projeto **SERWEJA,** um Servidor WEB projetado e desenvolvido utilizando programação orientada a objetos, onde são implementadas algumas funções típicas de monitores transacionais.

Dentro da proposta apresentada, o trabalho busca também realizar as seguintes contribuições para a área:

- Apresentar de forma clara os diversos conceitos relacionados com monitores transacionais e processamento transacional;
- Apresentar uma discussão sobre as características e benefícios do uso de transações em arquitetura cliente/servidor, buscando discutir suas aplicações diante da nova geração de soluções de mercado;
- Discutir os blocos componentes dos monitores transacionais, apresentando um dos diversos padrões de indústria que estão sendo desenvolvidos na área, visando criar uma infra-estrutura para a construção de monitores transacionais abertos;
- Apresentar um modelo de Servidor WEB com enfoque transacional, que contempla funções características de Monitores Transacionais;
- Apresentar a arquitetura do projeto SERWEJA, bem como uma descrição detalhada de seus módulos componentes, relacionandos suas principais atividades.

Iniciamos apresentando os monitores transacionais e o processamento transacional, visando identificar as principais funções que podem ser aproveitadas em aplicações cliente/servidor diversas. Logo após, descreve-se a arquitetura do projeto SERWEJA identificando os seus módulos internos e operação.

2. Monitores Transacionais e Processamento Transacional

"Monitor Transacional é um sistema operacional para processamento de transações"
- Jery Edwards

Os monitores transacionais são especializados em gerenciar transações de sua origem, tipicamente num cliente, através de um ou mais servidores, e então de volta para a origem com seus resultados. Quando a transação termina, o monitor TP (monitor de Transaction Processing), garante que todos os sistemas envolvidos na transação foram deixados num estado consistente. Além disso, o monitor sabe como executar a transação, roteá-la pelo sistema, executar o "load balance" (balanceamento de carga) e reiniciar quando ocorre falhas no sistema.

Um dos grandes aspectos dos monitores transacionais é poder supervisionar as transações de forma distribuída, gerenciando todos os recursos compartilhados do sistema. Um monitor TP pode gerenciar recursos desde um único servidor até múltiplos servidores e pode cooperar com outros monitores em configurações federadas. Nesta sessão, vamos detalhar os componentes internos do monitor transacional.

As soluções cliente/servidor modernas baseadas em monitores não têm tipo predominância no cenário atual, mas não é exagero presumir que num futuro próximo, cada máquina da rede terá um monitor para representá-la em transações globais.

As transações são mais do que meros agentes de negócios; elas se tornaram uma filosofia de projeto de aplicativos que garante robustez em sistemas distribuídos. Sob controle de um monitor transacional, uma transação pode ser gerenciada desde seu ponto de origem - tipicamente um cliente- através de um ou mais servidores e de volta àquele de origem. Quando uma transação termina, todas as partes envolvidas concordam se ela foi bem-sucessida ou se fracassou.

A transação é o contrato que amarra o cliente a um ou mais servidores. Ela é a unidade fundamental de recuperação, consistência e concorrência em um sistema cliente/servidor. Naturalmente, todos os programas participantes precisam aderir à disciplina transacional; de outro modo, um programa falho pode corromper todo um sistema. Em um mundo ideal, todos os programas cliente/servidor serão escritos como transações.

Os modelos de transações definem quando uma delas começa, quando termina e quais são as unidades apropriadas de recuperação em caso de falha. O modelo de transação plana é, há muito tempo o modelo principal dos monitores da atual geração (e de outros sistemas transacionais).

2.1 Módulos componentes de um monitor transacional

Um monitor transacional deve realizar duas tarefas principais:

- ***Gerenciamento de processos:*** incluindo iniciar os processos servidores, rotear o trabalho para esses processos, monitorar sua execução e balancear a carga de trabalho nesses processos.

- ***Gerenciamento de transações:*** significa que deve garantir as propriedades ACID (Atomicity, Consistency, Isolation, Durability) de todos os programas que estão rodando sob sua ação.

Historicamente, os monitores transacionais foram introduzidos para rodar uma classe de aplicações que poderia servir centenas e algumas vezes milhares de clientes (imagine uma aplicação de reservas de passagens aéreas).

Se a cada um desses clientes fosse dado todos os recursos necessários no servidor - tipicamente uma conexão de comunicação, meio Mbyte de memória, um ou dois processos e uma dúzia de controladores de arquivos ("handles") - até mesmo os mais robustos "mainframes" seriam colapsados. No entanto, quando os serviços são requiridos, os clientes esperam ser atendidos imediatamente. Sabemos que os usuários humanos possuem um tempo de tolerância de espera em torno de 2 segundos ou menos. Logo, os monitores transacionais proveêm um sistema operacional - no topo dos OS existentes - que conectam ao mundo real este milhares de humanos impacientes com um grupo de processos servidores de recursos.

2.2 Operação dos monitores transacionais

O chamado "funneling act" (afunilamento) é parte do que os monitores TP devem fazer para gerenciar o lado dos servidores em aplicações OLTP – "OnLine Transaction Processing". Em ambientes Windows, o lado do servidor de uma aplicação OLTP é tipicamente um módulo DLL – "Dinamic Link Library" - que contém

funções específicas. O monitor TP associa a execução de funções da DLL a uma classe de servidor - estes são grupos de processos ou threads que são iniciados e permanecem esperando por requisições de clientes. Cada processo ou thread numa classe de servidor é capaz de realizar um trabalho. O monitor TP é capaz de balancear a carga de trabalho entre eles.

Quando um cliente envia uma requisição de serviço, o monitor TP trata a requisição através de um processo servidor disponível do conjunto de classes de servidores (veja a Figura 2.2.1 - Monitor transacional e classes de servidores).

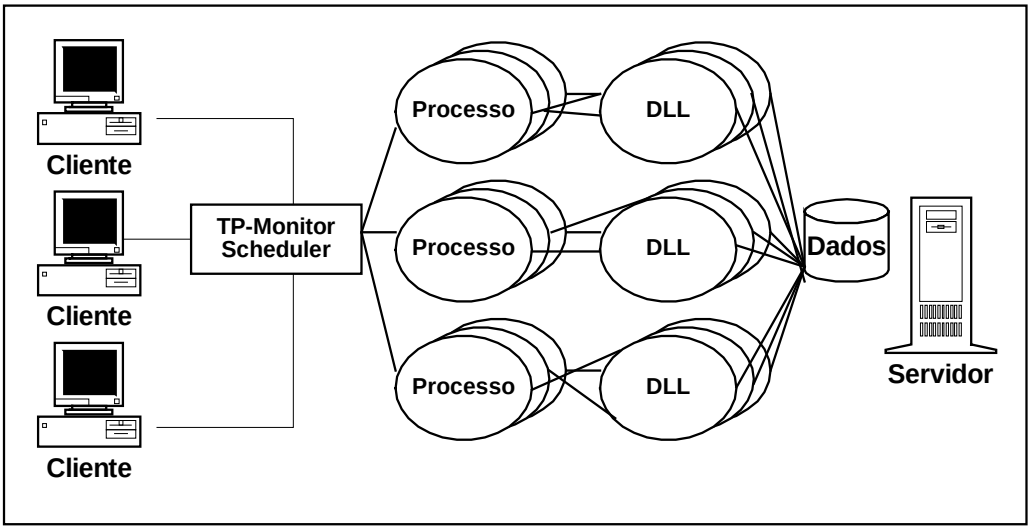

FIGURA 2.2.1 - Monitor transacional e classes de servidores.

O processo servidor amarra dinamicamente a DLL chamada pelo cliente, envoca suas funções e retorna os resultados ao cliente. Depois de completada a operação, o processo servidor pode ser reutilizado por outro cliente. O sistema operacional mantêm as DLL ainda carregadas na memória onde elas podem ser compartilhadas por outros processos.

Em resumo, o monitor transacional remove as requisições processo-por-cliente através do afunilamento das requisições dos clientes para processos servidores compartilhados. Se o número de requisições de clientes demandadas exceder o número de processos na classe de servidores, o monitor TP pode dinamicamente iniciar outros, isto é chamado de balanceamento de carga ("load balancing"). Os monitores mais sofisticados podem distribuir a carga dos processos através de CPUs em ambientes de multiprocessamento simétrico. Parte da ação de balanceamento de carga envolve gerenciamento de prioridades de classes e associação dinâmica de clientes importantes [DEPLED95].

2.3 Monitores transacionais como gerenciadores de transações

A disciplina de transações foi introduzida nos primeiros monitores TP para garantir a robustez de aplicações multiusuários que rodam em servidores. Estas aplicações tinham de ser *"a prova de balas"* e apresentar uma alta disponibilidade se elas fossem servir milhares de usuários em situação de mercado. Os monitores TP foram desenvolvidos a partir de sistemas operacionais para transações. A unidade de gerenciamento, execução e recuperação eram transações invocadas pelos

programas. O trabalho do monitor TP era garantir as propriedades do *ACID* durante uma alta demanda de transações. Para fazer, deveria gerenciar a execução, distribuição e sincronização das transações carregadas.

Com os monitores TP, os programadores de aplicações não precisavam se preocupar com problemas relacionados a emissão, concorrência, falhas, perdas de conexão, balanceamento de cargas e sincronização de recursos através de diversos nós. Tudo isso era feito de forma transparente - muito semelhante aos sistemas operacionais e suas ações diante de programas comuns. Simplesmente, os monitores forneciam um ambiente de execução de transações, rodando no topo do sistema operacional e hardware local.

Sistemas operacionais comuns devem entender a natureza dos jobs e recursos que gerenciam. Isto também é verdade para monitores TP - eles devem fornecer um ambiente otimizado para a execução de transações que rodam sob seu controle.

Existe tipicamente cinco categorias de interações cliente/servidor num ambiente transacional: *Conversacional, RPC (Remote Procedure Call), Queued (Fila de Mensagens), Batch (Lote).* As transações em batch rodam tipicamente em modo de baixa prioridade. Transações *RPC* e *Conversacional* usualmente envolvem usuário humano que requer atenção imediata, rodando em modo de alta prioridade. As transações no tipo *MOM (Message Oriented Middleware*) podem rodar em ambos os modos.

2.4 Padrão X/Open para monitores transacionais

Uma das caracteristicas para se julgar monitores transacionais distribuídos é sua aderência a padrões de sistemas abertos. Especificamente, o padrão X/Open define o modelo de processo e interface de serviços para aplicações TO em ambiente distribuído. De acordo com o padrão X/Open, uma aplicação TP distribuída consiste de quatro componentes e três interfaces. Os componentes são *Resource Manager (RM), Transaction Manager (TM), Communications Resource Manager (CRM), e Application Code*. As interfaces são XA, TX, e XA+ [CARRIE95] .

Em aplicações TP distribuídas, o gerenciador de recursos mantêm e acessa os dados (isto é mais frequente para sistemas de gerenciamento de base de dados, por exemplo o Oracle), enquanto o gerenciador de transações coordena o término ou recuperação do trabalho. O gerenciador de comunicação é um extensão para o presente modelo X/Open e permite uma aplicação passar o contexto de execução, síncrona ou assíncronamente para outros sistemas de transação. O seguinte diagrama apresenta os componentes do X/Open, suas relações e interfaces padrões para uma aplicação TP.

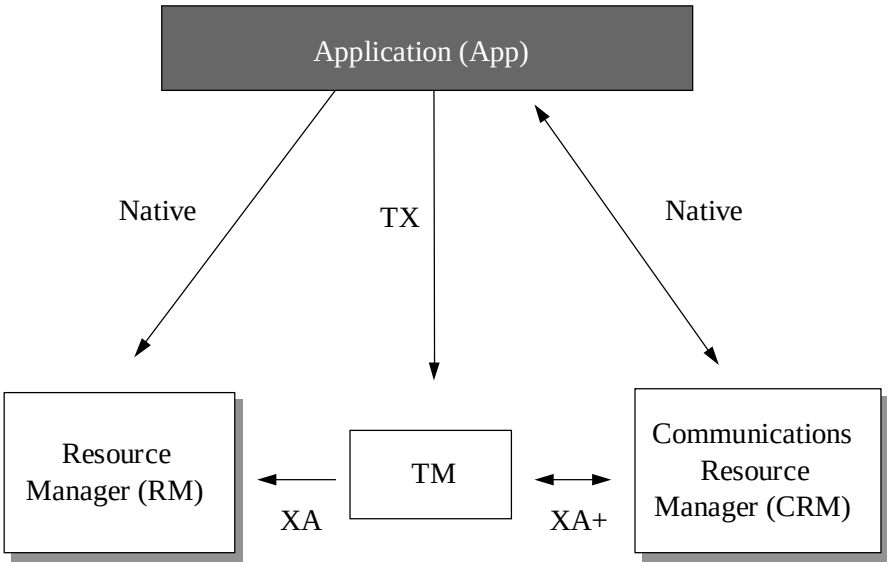

FIGURA 2.4.1 - Padrão X/Open

Uma aplicação usa a interface TX para comunicar com o gerenciador de transação. A interface TX consiste de chamadas de funções as quais definem o escopo da transação (ou demarcam a transação) e o objetivo de resultados da transação. Atualmente, a interface TX tem sido somente disponível em protótipos e portanto, não tem sido reconhecida como um padrão aberto oficialmente.

A interface XA, que é um padrão de sistema aberto, é usada para comunicar informações de transações dos gerenciador de transações para o gerenciador de recursos. Por exemplo, o Encina™ (que é um gerenciador de transações) e Oracle (que é um gerenciador de recursos) comunicam dados e instruções via interface XA. A interface XA+ é utilizada para se comunicar com o CRM.

2.5 Conceitos de transações

Para entendermos completamente o que rege a operação de um monitor transacional, vamos agora definir os conceitos relacionados com as transações, seus diversos modelos e funcionalidades. Primeiro, vamos conhecer as propriedades ACID, já mencionadas anteriormente.

"Transações são mecanismos que garantem operações ACID."
- Gray and Reuter (1993)

Uma transação corresponde a uma coleção de ações que apresentam as características ACID. Neste caso ACID, expressão cunhada por Andreas Reuter em 1983, significa *Atomicity, Consistency, Isolation* e *Durability* [GRAY93].

- ***Atomicidade (Atomicity):*** significa, como citado anteriormente, que a transação é uma unidade de trabalho indivisível: toda as suas ações são executadas com sucesso ou todas falham. Uma proposta de "tudo ou nada". As ações sob uma transação podem incluir filas de mensagens ("message queues"), atualizações em bancos de dados, e apresentações de resultados na tela dos clientes. Atomicidade é definida na perspectiva do consumidor de transações.

- ***Consistência, Integridade (Consistency):*** significa que depois que a transação foi executada, ela deixa o sistema num estado consistente. Se a transação não pode ser executada completamente, ela deve deixar o sistema no estado em que o encontrou, de modo consistente.

- ***Isolação (Isolation):*** significa que o comportamente de uma transação não é afetado pela comportamento de outra que está executando concorrentemente. A transação deve serializar todos os acessos a recursos compartilhados e garantir que programas concorrentes não danifiquem as operações uns dos outros. Um programa multiusuário rodando sob a proteção de uma transação deve comportar-se exatamente como se num ambiente monousuário. As mudanças realizadas nos recursos compartilhados que foram executadas pela transação devem ser visíveis apenas após o término da transação. Novamente, esta é uma perspectiva do consumidor de transações.

- ***Durabilidade, Persistência (Durability):*** significa que os efeitos de uma transação são permanentes após seu término (seu commit). Suas alterações devem sobreviver sobre falhas no sistema. O termo "persistent" também é um sinónimo para Durability.

Uma transação portanto, torna-se a unidade fundamental de recuperação, consistência e concorrência em um sistema cliente/servidor. Qual a sua importância ? Tome como exemplo uma operação bancária de crédito/débito numa conta corrente. A transação só estará completamente executada quando a quantia for debitada de uma conta e creditada na outra destino. Isto significa que em seu código, deve-se garantir a integridade de toda a operação. Esta tarefa é usualmente executada pelo monitor transacional, ou seja, o programador não deveria desenvolver nenhuma linha de código para garantir suas transações.

2.5.1 Delimitando uma transação

Programação usando transações requer um conjunto especial de primitivas que deve ser fornecido pelo sistema operacional ou pelo ambiente de execução ("run time") de alguma linguagem. Alguns exemplos destas primitivas são:

1. BEGIN_TRANSACTION : marca o início de uma transação;
2. END_TRANSACTION: termina a transação e tenta encerrar (operação de commit);
3. ABORT_TRANSACTION: mata a transação, recuperando os valores originais;
4. READ: Lê dados de um arquivo (ou outro objeto);
5. WRITE: Escreve dados em um arquivo (ou outro objeto);

A lista exata de primitivas depende do tipo de objeto que está sendo usado dentro da transação. Num sistema de mail, enviar, receber e consulta um emial. Em sistemas de contabilidade, elas devem ser diferentes. Ler e Escrever (read e write) são exemplos típicos entretanto. Declarações ordinárias, chamadas de procedimentos e outros são também permitidos dentro de uma transação. *BEGIN_TRANSACTION* e *END_TRANSACTION* são usados para delimitar o escopo da transação. As operações entre eles formam o corpo da transação. Todas as operações são executadas ou nenhuma delas. Estas podem ser chamadas de sistema, biblioteca de procedimentos, ou comandos de uma linguagem, dependendo da implementação. A tabela abaixo apresenta alguns exemplos de primitivas usados em monitores:

	Delimitadores de Transações		
Sistema	***Início***	***Término***	***Abortar***
TUXEDO	TPBEGIN	TPCOMMIT	TPABORT
Top End	tx_begin	tx_commit	tx_rollback
Encina RPC	transaction	onCommit	onAbort
X/Open	tx_begin	tx_commit	tx_rollback
OSI TP	C-BEGIN	C-COMMIT	C-ROLLBACK
Tandem RSC	Begin_Transaction	End_Transaction	Abort_Transaction
CICS	SYNCPOINT	SYNCPOINT	ROLLBACK

TABELA 2.5.1.1 - Comparação entre delimitadores de flat transactions.

Ainda, num mundo ideal cliente/servidor, todas as operações deveriam ser feitas baseadas em transações.

3. Servidor WEB - uma aplicação transacional

Até agora apresentamos os conceitos relacionados aos monitores transacionais e o processamento transacional. Vamos agora iniciar a identificação de um tipo de aplicação que apresenta características de um ambiente transacional. Nosso objetivo é apresentar uma solução para essa classe de aplicações, enfocando os ganhos com o uso de funções transacionais.

3.1 Servidor Web Típico

Com o crescimento do uso da internet e de aplicações Cliente/Servidor sobre a rede, diversas tecnologias surgiram com características peculiares associados ao funcionamento sobre um ambiente de rede. Umas dessas tecnologias corresponde ao uso de Servidores WEB, servidores de documentos formatados em HTML.

A linguagem HTML permite que o usuário formate um texto e que este seja transferido entre máquinas diferentes na rede, através do protocolo HTTP - HiperText Markup Language, que se encarrega do gerenciamento das mensagens trocadas entre cliente, o requisitante do documento, e servidor WEB, o fornecedor de documentos HTML.

A FIGURA 3.1.1 apresenta um servidor WEB típico, com seus componentes:

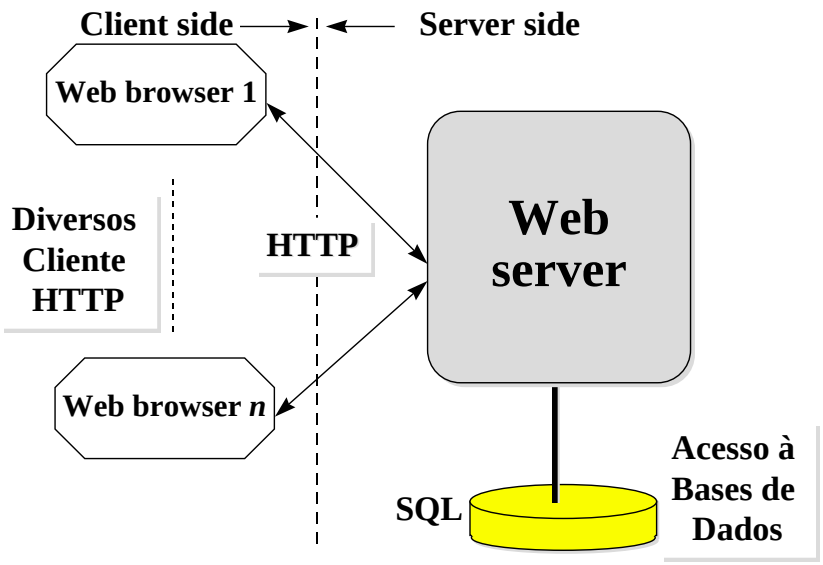

FIGURA 3.1.1 - Servidor WEB típico

Algumas atividades típicas de um servidor WEB são:

- Servidor de arquivos e documentos formatados html
- Requisição de documentos dinâmicos
- Monitoração de dados transferidos
- Acesso à bases de dados no servidor
- Execução de scripts preparados pelo operador do sistema
- Disparo de aplicações no servidor

Assim como tem sido importante o uso de servidores WEB em aplicações internet, outra tecnologia desponta e amplia suas aplicações. Corresponde ao Commum Gateway Interface.

A tecnologia CGI - Common Gateway Interface - inclui duas funções principais como podemos observar abaixo:

- a possibilidade de disparar, do cliente, uma aplicação no servidor;
- a possibilidade de passar alguns valores de variáveis que poderão ser usadas nesse programa, para a geração de resultados dinâmicos para o cliente.

O Common Gateway Interface (CGI) é um padrão de interface para aplicações externas, visando a troca de dados com servidores de informações, como HTTP e servidores WEB.

Um documento HTML comum que é fornecido por servidores WEB é estático, o que significa que está disponível num estado constante, sem alterações a cada requisição. Um programa CGI, por outro lado, é executado em tempo-real, o que permite a emissão de informações de forma dinâmica.

Por exemplo, digamos que você queira compartilhar um banco de dados local com o resto da rede internet, permitindo que pessoas de diferentes partes do mundo executem pesquisas no banco de dados. Basicamente, você necessita criar um programa CGI que o servidor WEB irá executar para transmitir a informação para o núcleo do banco de dados ("database engine"), e receber os resultados de volta, formatando-os e apresentando-os ao cliente. Este é um tipo de exemplo de gateway,

e é exatamente o que o CGI, na corrente versão 1.1, consegue fazer [WEIN96].

O exemplo de banco de dados é uma idéia simples, mas dos que apresentam maiores dificuldades de implementação, tanto por problemas de depuração como manutenção. Porém, não existem limites quanto ao que se pode fornecer através de um WEB server. A única coisa que você precisa saber é que irá necessitar de um programa CGI para executá-lo.

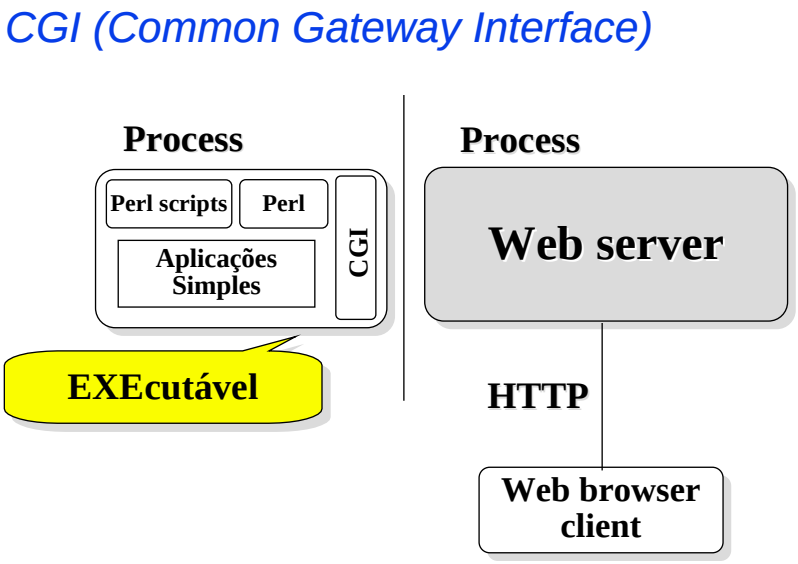

FIGURA 3.1.2 - Servidor WEB típico com CGI.

Os programas CGI são normalmente pequenos, evitando tomar muito recurso do servidor WEB. As próximas seções irão detalhar as características do padrão CGI, descrevendo as variáveis de ambiente que são trocadas entre Cliente e Servidor durante o processo de requisão de documentos dinâmicos.

Desde que um programa CGI é um executável, isto é basicamente o equivalente a deixar que um programa externo execute dentro do seu sistema, o que não é uma coisa muito segura de se fazer. Portanto, existem algumas precauções que precisam ser implementadas quando vamos usar programs CGI. Provavelmente, uma atitude que tipicamente altera nosso WEB é que todos os programas CGIs a serem executados devem permanecer num diretório separado, residente na árvore de diretório do WEB server. Permanece portanto sob o controle do webmaster, proibindo que qualquer usuário da rede insira programas CGI outros que não os autorizados para serem executados. A segurança é obtida garantindo-se que este tipo de programa não possa ser carregado no sistema sem prévia checagem.

3.2 Servidor Web Transacional

Um servidor WEB com Enfoque Transacional permite a inclusão de funções típicas de processamento transacional, tais como:

- ***Afunilamente (Funneling Act)*** - controle de acesso dos diversos usuário ao servidor. Estes usuários estão acessando o servidor através da internet, requisitando serviços típicos de transferências de arquivos e/ou disparo de scripts e programas no servidor, os chamados CGI's;

- ***Balanceamento de Carga (Load Balancing)*** - gerenciamento da carga de sistema, balanceando-a. Este controle pode ser identificado como o número de threads ou processos disparados simultaneamente para o tratamento de cada uma das requisições recebidas de cliente. Cada processo disparado pode estar concentrado em tratar uma transferência de arquivos, um acesso ao banco de dados, ou simplemente a execução de um módulo CGI;

- ***Chamada Remota de Procedimentos Transacionais - TRPC – (Transactional Remote Procedure Call) -*** através de disparo de funções com rótulos de identificação de uma transação. O grande ganho na utilização do enfoque transacional no servidor Web: todos os processos disparados dentro de uma servidor WEB transacional podem ser encarados como transações, aderentes às propriedades ACID já apresentadas. Isto significa que o disparo de CGI pode estar encapsulado dentro de uma transação, o que permite o controle de acesso, o log de eventos durante o processo, a recuperação em caso de falhas e a identificação da transação junto ao cliente originador;

- ***Recuperação de falhas*** - através da recuperação ("rollbacks") em bancos de dados. Como todo processo disparado dentro do sistema é monitorado como uma transação, todas as atividades são registradas, criando-se uma série de relatórios de acompanhamento, que permitem a recuperação de falhas e a auditoria do sistema;

- ***Logging Event*** - criando relatórios de execução de acesso e erros ocorridos durante o tratamento de cada uma das requisições recebidas dos clientes.

Podemos relacionar ainda as seguintes características presentes em um servidor Web que estão presentes em aplicações transacionais:

- O sistema tem um grande número de usuários esperados (100 ou mais);
- O sistema usa múltiplas plataformas de SGBD ou hardware;
- O sistema usa pesadamente sistemas de arquivos;
- O sistema deve apresentar requisitos de performance, balanceando a carga de processamento através de múltiplos processos;
- O sistema deve interoperar como aplicações de mainframes ("legacy systems"), para casos de acesso a bancos de dados.

Desse modo, observamos a aderência de aplicações WEB ao enfoque transacional, iniciando assim a apresentação do projeto **Serweja - Servidor WEB em Java** - desenvolvido como parte prática do estudo sobre monitores transacionais, buscando uma solução para a aplicação dos conceitos até aqui estudados.

4. Projeto SERWEJA - Servidor Web em Java com enfoque transacional

O Projeto **SERWEJA** - Servidor Web em Java - implementa conceitos de processamento transacional. Sendo projetado com este enfoque, apresenta características importante, tais como:

- Carga balanceada de "threads", ou linhas de execução (Load Balancing);
- Gerenciamento de "threads" disparados no sistema;
- Conjunto de "threads" para o tratamento de requisições de clientes HTTP – "HiperText Transfer Protocol";
- Conjunto de "threads" para o tratamento de CGI – "Commum Gateway Interface" - de aplicações do usuário;
- Acesso a bancos de dados SQL – "Structured Query Language" - via JDBC – "Java DataBase Connectivity";
- Conceitos de transações implementados em "threads" CGIs - Propriedades do ACID;
- Modelagem de Transações do tipo "FLAT" em "threads" CGIs ;
- Registro de acessos realizados;
- Registro de erros em conexões de clientes HTTP;
- Manutenção dos estados de uma aplicação CGI de usuário para a mesma conexão.

Outras características presentes no Projeto **SERWEJA** são:

- Tratamento de requisições via métodos GET e POST do protocolo HTTP 1.0
- Tratamento diferenciado de requisições GET para arquivos HTML, GIF, JPEG e TEXT
- Disparo de "threads" CGIs para aplicações monitoradas
- Interface Homem-Máquina para o operador do Servidor orientada para menus
- Acesso aos registros do sistema através da propria interface
- Configuração remota do sistema.

4.1 Arquitetura do projeto SERWEJA

Sendo concebido e desenvolvido com enfoque orientado a objetos, o projeto SERWEJA corresponde a um Servidor WEB com enfoque transacional, que permite o disparo de CGIs e seu tratamento como transações, efetuando assim todo o processo através de funções típicas de monitores transacionais. Sua arquitetura orientada a objetos permite uma maior clareza na identificações e comunicações dos módulos internos do sistema.

Para a concepção do servidor foram adotados os módulos gerais descritos no padrão X/Open, onde os componentes *Resource Manager (RM), Transaction Manager (TM), Communications Resource Manager (CRM) e Application Code* foram transformados em objetos dentro do sistema.

O núcleo do projeto Serweja apresenta os seguintes módulos:

- **Serweja** - módulo básico de inicialização
- **SerwejaConexaoBasica** – "thread" servidor Master de requisições de clientes

HTTP.

- **SerwejaConexaoSecundaria** – "thread" gerenciador de transações **(TM),** responsável pelo tratamento específico de uma requisição de cliente HTTP.
- **SerwejaConf ig** - objeto de controle de recursos **(RM)** que informa ao thread de tratamento os recursos disponíveis para execução.
- **SerwejaCGIxxxx** – conjunto de "threads' de tratamento de requisições de clientes HTTP.

A **classe Serweja** declara o objeto responsável pela interface homem-máquina do servidor, o objeto responsável pelo controle de configurações e gerenciamento de threads do sistema, e o thread que implementa o módulo servidor Master, que aguarda conexões realizadas pelos clientes HTTP.

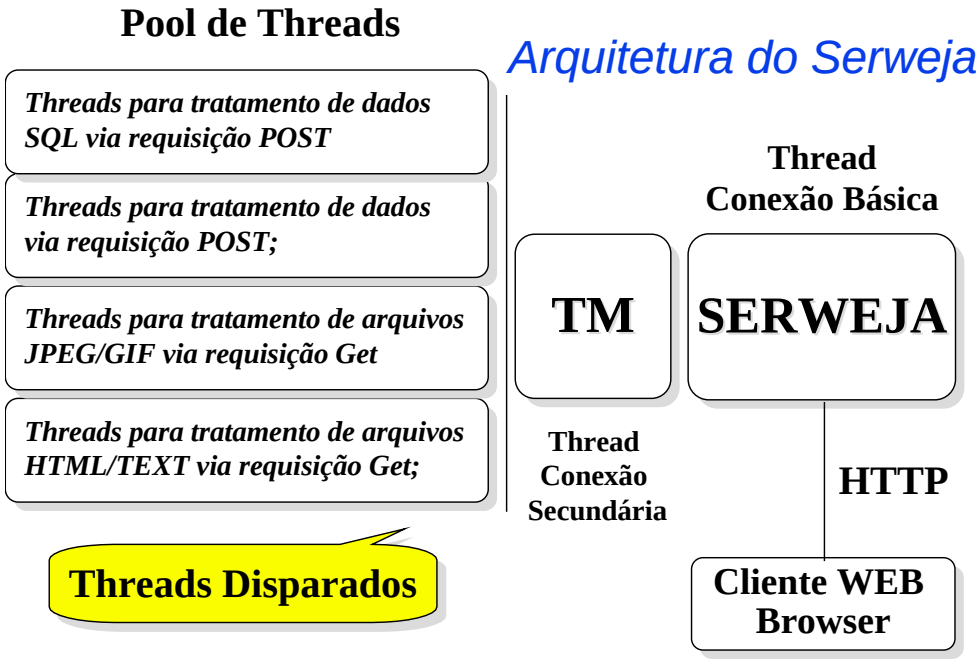

FIGURA 4.1.1 - Arquitetura do Projeto Serweja

Ao atender uma requisição, o "thread" de conexãos básica dispara um objeto dedicado para o tratamento da requisição, através de uma conexão secundária.

A **classe thread SerwejaConexaoBasica** implementa o "thread Master" de atendimento de requisições. Para isso, apresenta a declaração do "socket" de servidor, dedicado para a recepção da requisição.

A **classe SerwejaConexaoSecundaria** implementa o módulo de tratamento de uma dada requisição recebida. Esse objeto, ao ser disparado, cria dinamicamente os **objetos** e **threads** do **conjunto de threads** do servidor.

De acordo com o tipo de requisição tratada, um "thread" determinado é disparado. Para a recepção e envio de dados ao cliente, o "socket" da conexão é passado para a classe por parâmetro na declaração do objeto.

A **classe SerwejaCGIAmbiente** implementa o controle de dados da requisição do cliente. Através do objeto dessa classe, o servidor trata os dados do protocolo HTTP usado na comunicação cliente/servidor.

A classe possui as declarações das variáveis de controle para o protocolo e o cabeçalho, o que permite o tratamento de diversos métodos de transmissão de dados previstos no protocolo HTTP.

No Serweja, dois métodos estão implementados: o métodos GET de requisição de arquivos e o métodos POST de envio de dados entrados pelo usuário.

Como característica principal do Serweja, os threads de tratamento de requisição são monitorados durante sua execução, de acordo com as funções de monitores transacionais implementadas, como *: LOAD BALANCING, FUNNELING ACT, LOG EVENTS*, etc.

4.2 Aspectos de desenvolvimento do projeto SERWEJA

Alguns aspectos importantes de desenvolvimento do projeto **SERWEJA** são**:**

- Uso da Metodologia ***Fusion***, como metodologia de desenvolvimento de software orientado a objetos [COLEMAN94].

  A utilização dessa metodologia fortemente gráfica permitiu uma clara documentação dos módulos internos do projeto, além da concepção orientada para objetos desde as etapas de análise de requisitos. Isso facilitou a fase de implementação, executada quase que imediatamente a partir dos documentos das etapas de análise e arquitetura;

- Utilização da ***Linguagem JAVA***, para criação de aplicações "stand-alone".

  Foi escolhida a linguagem JAVA para a implementação do projeto SERWEJA, devido suas características e aderência aos conceitos de orientação a objetos.

  A linguagem apresenta algumas classes primitivas que são aderentes às classes básicas OBJETO, THREAD e FRAME, permitindo a implementação imediata das estruturas do núcleo do servidor [MONT94].

  O tratamento de sockets e threads apresenta diversos métodos e funções de apoio. O uso da API para acesso a ODBC – Open DataBase Connectivity - permite diversas construções e implementações de transações do tipo Flat automático.A portabilidade do código garante flexibilidade durante o desenvolvimento e execução do projeto.

- Uso de ***API-JDBC*** e ***Bridge JDBC/ODBC*** para acesso a bancos de dados SQL

  Para aplicações de CGI's com acesso a bancos de dados, foram necessários módulos e drivers ODBC que trabalhassem com a linguagem JAVA.

  Para o tratamento de uma requisição que exija acesso a bancos de dados, os módulos envolvidos são apresentados na figura a seguir. Observe que estes módulos fazem parte do sistema, sendo gerenciados e monitorados como componentes de uma transação. Portanto, os módulos internos já apresentados coordenam todo o processo apresentado.

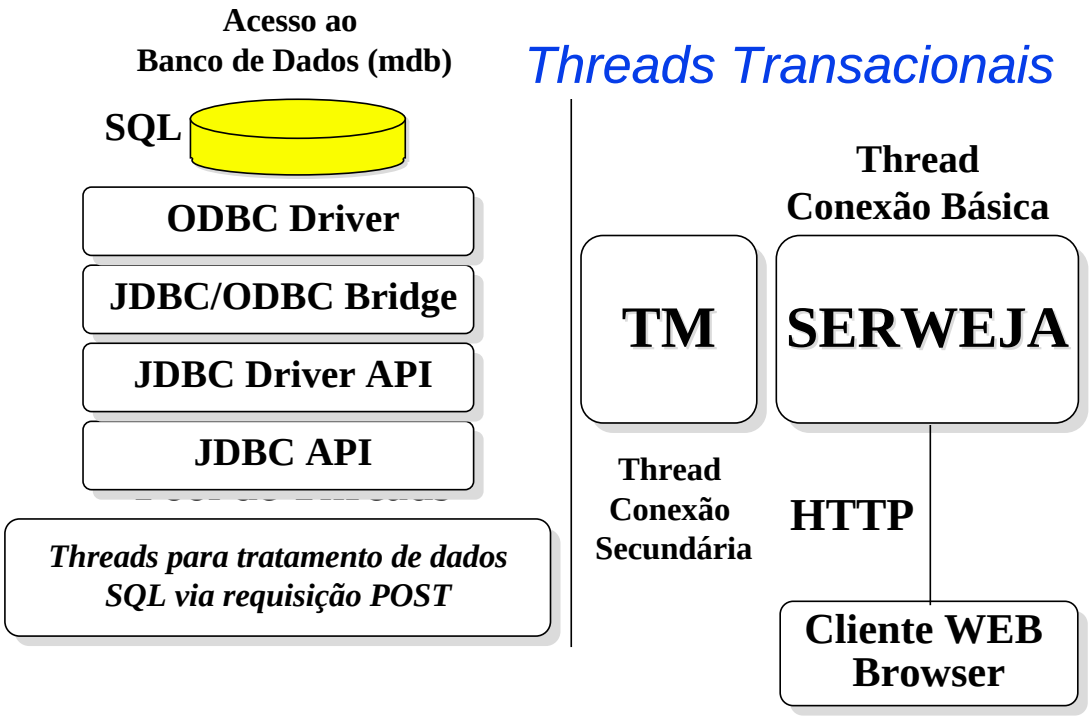

FIGURA 4.2.1 - Threads Transacionais com acesso JDBC

- Utilização de plataforma ***Windows NT 4.0***

  O desenvolvimeto em plataforma Windows NT 4.0 forneceu um ambiente controlado para o trabalho em rede, rodando protocolo TCP/IP através de uma rede de computadores. No Windows NT Server 4.0 foi implementado o servidor **SERWEJA**, acessado a partir de estações **Windows NT WorkStation e Clientes Windows 95**, utilizando browsers Internet Explorer da Microsoft e Netscape Navigator.

- Desenvolvimento em ambiente ***MS-Visual J++***

  A codificação do projeto foi realizada através do ambiente Microsoft Visual J++, que permitiu grandes ganhos na etapa de depuração e teste, facilitando o acompanhamento de testes intensivos.

5. Considerações Finais

Neste trabalho, apresentamos os conceitos relacionados aos Monitores Transacionais, e observamos que estes inserem nas aplicações as seguintes questões:

1. Segurança - autorização e autenticação de usuários que disparam processos no servidor. Os próprios processos estão cadastrados quanto as restrições de recursos disponíveis para sua alocação e execução;

2. Atomicidade - cada unidade de trabalho é um entidade separada e atômica e identificada dentro do sistema durante sua execução;

3. Isolação - completa independência de tarefas, uma vez que cada requisição

disparada possui seu próprio processo. É importante notar que em caso de uma sequência de processos concorrentes pelo mesmo recursos, os processos são serializados para a alocação do recursos;

4. Durabilidade - permanência dos resultados dos processos, mesmo após uma falha no sistema;

5. Recuperação - acompanhamento dos estados durante a operação, permitindo a recuperação em caso de pane em algum dos módulos;

6. Interoperabilidade - tratamento da comunicação entre plataformas diferentes.

Através da construção do projeto SERWEJA - Servidor Web com enfoque transacional - foi possível observar que:

1. O enfoque transacional facilitou o controle de execução de threads para tratamento de requisições de clientes HTTP, permitindo uma monitoração das atividades efetuadas dentro de cada transação, e ainda a geração de registros de eventos associados à conexão geradora da requisição;

2. A implementação de um conjunto de "threads" diferenciados para o tratamento de requisições permitiu uma maior flexibilidade no sistema, uma vez que num determinado momento, o servidor pode optar por disparar threads de diferentes tipos para atender seus clientes, bem como é possível a ativação/desativação em separado de alguns destes threads;

3. Geração de registros de acesso e erros, iden;tificados pela conexão;

4. A Carga Balanceada durante a execução do sistema permitiu uma robustez quanto ao número de clientes acessando o servidor. Através do cadastro dos threads CGI's como transações, é possível identificar o total de recursos necessários para sua execução antes do sistema entrar em colapso;

5. Execução de threads CGIs com enfoque transacional, permitindo a implementação de transações do tipo FLAT. Esse threads permitiram a criação de aplicações exemplo como Bancos Disponíveis via internet, Cadastro de Visitantes Remoto, Configuração Remota do Servidor, etc, fornecendo ainda uma série de APIs que permitem uma rápida implementação de novas aplicações pelos interessados;

Concluímos que a utilização de um enfoque transacional na construção de sistemas "on-line" baseados em servidores Web é factível e de forma prática permite um ganho na robustez do sistema, uma vez que traz conceitos ligados ao gerenciamento de desempenho e segurança na execução dos módulos mais críticos do servidor.

6. Referências Bibliográficas

[BECK89] BECK, K. et. al. **A laboratory for teaching object-oriented thinking.** In ACMOOPSLA ´89 Conference Proceedings, 1989.

[BOOCH94] BOOCH, B. et. al. **Object-Oriented Analysis and Design with Applications.** 2.ed. Benjamin/Cumming Editions, 1994.

[BOONER96] BONNER, P. **Network Programming with Windows Sockets.** 1.ed. Prentice Hall International Editions, 1996.

[CARRIE95] CARRIE, J. I.; BAAFI, R. K.; DRURY, W. B. ACMSxp Open Distributed Transaction Processing. **Digital Technical Journal**, v.07, n.01, p.23-32.

[CHESW94] CHESWICK, W. R.; BELLOVIN, S. M. **Firewall and Internet Secutrity - Repelling the Wily Hacker.** Massachusetts, Addison-Wesley Publishing Company, 1994.

[COLEMAN94] COLEMAN, D. et. al. **Object-Oriented Development - The Fusion Method.** New Jersey, Prentice-Hall Object-Oriented Series, 1994.

[COMER93] COMER, D. E.; STEVENS, D. L. **Internetworking with TCP/IP vol.III: Client-Server Programming and Applications - BSD Socket Version.** Prentice-Hall International Editions, 1993.

[COMER95] COMER, D. E. **Internetworking with TCP/IP vol.I: Principles, Protocols, and Architecture.** Third Edition, Prentice-Hall International Editions, 1995.

[DEPLED95] DEPLEDGE, N. G.; TURNER, W. A.; WOOG, A. An Open Distributable Three-Tier Client-Server Achitecture with Transaction Semantics**. Digital Technical Journal**, v.07, n.01, p.34-42.

[GRAY93] GRAY, J.; REUTER, A. **Transaction Processing: Concepts and Techniques.** San Francisco, Morgan Kaufmann Publishers, 1993.

[JACOBSON92] JACOBSON, I. **System Development**. Reading, MA, Addison-Wesley Publishing Company, 1992.

[KOVACH96a] KOVACH, S. **Arquitetura Cliente/Servidor - Conceitos e Programação.** São Paulo, LARC-Laboratório de Arquitetura e Redes de Computadores, Escola Politécnica da USP, 1996.

[KOVACH96b] KOVACH, S. **Arquitetura TCP/IP.** São Paulo, LARC-Laboratório de Arquitetura e Redes de Computadores, Escola Politécnica da USP, 1996.

[KOVACH96c] KOVACH, S.; CARVALHO, T.C.M.B. **Redes Locais.** São Paulo, LARC-Laboratório de Arquitetura e Redes de Computadores, Escola Politécnica da USP, 1996.

[KOVACH96d] KOVACH, S.; CAMBIUCCI, W. **Interconexão de Redes de Computadores.** São Paulo, LARC-Laboratório de Arquitetura e Redes de Computadores, Escola Politécnica da USP, 1996.

[LEMAY96] LEMAY, L.; PERKINS C.L. **Teach yourself JAVA in 21 days.** 1st edition, Indianapolis, Indiana, SamsNet Publishing, 1996.

[MONT94] MONTGOMERY, S. **Object Oriented Information Engineering.** 1.ed., AP Professional, 1994.

[ORFAL94] ORFALI, R.; HARKEY, D.; EDWARDS, J. **Essential client/server survival guide.** 3.ed. New York, John Wiley & Sons, Inc, 1994.

[ORFAL95] ORFALI, R. et. Al. Computação Cliente/Servidor. **BYTE Brasil**, v.04, n.05, p.58-82.

[PERLMAN92] PERLMAN, R. **Interconnections - Bridges and Routers.** Massachusetts, Addison-Wesley Publishing Company, 1992.

[PRESSMAN95] PRESSMAN, R.S. **Engenharia de Software.** 3.ed. São Paulo, Makron Books, 1995.

[RITCHEY95] RITCHEY, T. **JAVA.** Indianapolis, Indiana, New Riders Publishing, 1995.

[RUMB91] RUMBAUGH, J. et. al. **Object-Oriented Modeling and Design.** New Jersey, Prentice-Hall Inc., 1991.

[SOARES95] SOARES, L. F. G.; LEMOS, G.; COLCHER, S. **Redes de Computadores, Das LANS, MANS e WANS às Redes ATM.** Rio de Janeiro, Editora Campus-Embratel, 1995.

[SOUZA95] SOUZA, E.A. O Modelo OSE do IEEE - Uma forma didática de conceituar Sistemas Abertos. **LAN TIMES Brasil**, v.01, n.02, p.78-85.

[TANEN95] TANENBAUM, A. S. **Distributed Operating Systems.** New York, Prentice-Hall International Editions, 1995.

[TITTEL95] TITTEL, E. et al. **Foundations of WWW programming with HTML & CGI.** 1.ed., IDG Books, 1995.

[WEIN96] WEINMAN, W. E. **The CGI Book.** Indianapolis, Indiana, New Riders Publishing, 1996.

[WEISS94] WEISS, M.; WINGROVE, B. **The Role of the TP Monitor In a Client/Server Platform.** AMSCAT-WP-94-108, AMS Center for Advanced Technologies, Working Paper Series, 1994.

[WUTKA97] WUTKA, M. **Ihacking JAVA - The Java Professional's Resource Kit.**, Indianápolis, Indiana, QUE Corporation, 1997.