UNIVERSIDADE EDUARDO MONDLANE

FACULDADE DE CIÊNCIAS

DEPARTAMENTO DE MATEMÁTICA E INFORMÁTICA

TRABALHO DE LICENCIATURA

SISTEMAS DE PROCESSAMENTO DE TRANSAÇÕES

APLICAÇÃO DO SERVIDOR DE TRANSAÇÕES NUM SISTEMA DE OPERAÇÕES BANCÁRIAS

Obadias Agostinho Langa

UNIVERSIDADE EDUARDO MONDLANE

FACULDADE DE CIÊNCIAS

DEPARTAMENTO DE MATEMÁTICA E INFORMÁTICA

TRABALHO DE LICENCIATURA

SISTEMAS DE PROCESSAMENTO DE TRANSAÇÕES

APLICAÇÃO DO SERVIDOR DE TRANSAÇÕES NUM SISTEMA DE OPERAÇÕES BANCÁRIAS

SUPERVISORES: Eng. Orlando Grosso

Dr. Enrique Rosa

AUTOR: Obadias Agostinho Langa

Maputo, junho de 1998

AGRADECIMENTOS

Sinceros agradecimentos são endereçados aos superviores Eng. Orlando Grosso e dr. Enrique Rosa por não terem poupado esforços para que o presente trabalho se tornasse realidade.

Também a EXI, na pessoa do seu Dir. Geral-Eng. José Murta, por ter permitido que o trabalho se desenvolvesse em paralelo com outras actividades profissionais na Empresa, vão os meus sinceros agradecimentos.

A todos os colegas, amigos e em especial ao Eng. Lázaro, pela atenção e colaboração dispensada, também vão os meus sinceros agradecimentos.

E àquela grande Família, que sempre me orgulharei por nela pertençer, também vão os meus sinceros agradecimentos.

DECLARAÇÃO DE HONRA

Declaro por minha honra, que este trabalho é fruto da minha profunda investigação e não foi submetido para um outro grau que não seja o indicado, **Licenciatura em Informática na Universidade Eduardo Mondlane.**

Maputo, 30 de Junho de 1998

O Autor

(Obadias Agostinho Langa)

ABREVIATURAS

AP- Aplication Program

API- Aplication Programming Interface

COM- Component Object Model

CREDICOOP- Sociedade Cooperativa de Crédito e Investimento

DBMS- Database Management System

DCE- Distributed Computing Environments

DCOM- Distributed Component Object Model

DLL- Dynamic Link Library

DTC- Distributed Transaction Coordinator

DTP- Distributed Transaction Processing

EXI- Engenharia e Comercialização de sistemas Informáticos

IDL- Interface Definition Language

IIS- Internet Information Server

IMS- Information Management System

LPC- Local Procedure Call

MS DTC- Microsoft Distributed Transaction Coordinator

MTS- Microsoft Transaction Server

ODBC- Open Database Connectivity

RM- Resource Manager

RPC- Remote Procedure Call

SNOB- Sistema único de Operações Bancárias

TCP/IP- Transmission Control Protocol / Internet Protocol

TM- Transaction Manager

RESUMO

Devido á crescente complexidade das aplicações actuais há uma tendência para o crescimento da dimensão dos programas. A solução de dividir as aplicações em "cliente" (parte dedicada á ligação com os utilizadores) e "Servidor" (contendo as regras e ligações ás bases de dados) apresenta frenquentemente problemas de optimização de recursos de equipamentos disponíveis.

A tendência actual dos fornecedores de software aponta para o desenho de soluções particionando as aplicações em "cliente", "servidor de aplicações" e "servidor da base de dados", onde o "servidor de aplicações" contém os módulos ou objectos referentes à "lógica da aplicação".

Assim foram estudadas e comparadas soluções cliente/servidor utilizando implementações em dois e três níveis, ensaiando o conceito de servidor de aplicações num sistema de gestão bancária, SNOB.

ÍNDICE

INTRODUÇÃO

As actividades comerciais, governamentais, científicas e culturais, estão-se tornando cada vez mais dependentes de recursos de informação baseados em computador. Como muita e variada informação é capturada e mantida em sistemas computadorizados, as técnicas de exploração, manipulação e protecção desta informação tornam-se muito críticas em sociedades modernas industrializadas.

A tecnologia do processamento de transações é a chave para uma manipulação coerente e exploração de recursos de informação baseados em computador.

O processamento de transações engloba técnicas de manipulação de informação armazenada e programas de aplicação que interpretam e manipulam tal informação. Desde a recuperação da base de dados e controlo da coerência até aos monitores de transações que iniciam e controlam a execução de aplicações, a tecnologia de processamento de transações providencia mecanismos e facilidades necessárias para a protecção e manipulação dos recursos críticos de informação que constituem a base para muitas (se não todas) as actividades comerciais, científicas e culturais.

Em geral, a exploração da informação armazenada envolve o acesso e actualização de itens de dados relacionados que colectivamente descrevem ou modelam o estado e a evolução das actividades e fenómenos do Mundo real. Porque itens múltiplos de dados tem de ser sempre acedidos e modificados juntos para correctamente reflectir o Mundo real, muito cuidado deve ser tomado para manter a consistência mútua entre itens de dados relacionados.

Qualquer interrupção de actualizações a itens relacionados, ou a intercalação de modificações e acessos pode tornar a informação inconsistente.

A chave para manter a consistência de dados é a identificação das sequências de acessos e actualizações que representam a interrogação e modificação de itens de dados relacionados.

Tais sequências são chamadas **transações.** A tecnologia do processamento de transações assegura que cada transação é executada completamente ou não, e aquelas transações executadas concorrentemente, o são isoladamente. O significado desta tecnologia é magnificado pelo facto de que estas garantias são mantidas mesmo em falhas de componentes do computador, da distribuição de dados através de múltiplos computadores, e execução paralela de diferentes transações.

Para manter o controlo sobre tais garantias, monitores de transações foram sempre sendo utilizados. Estes monitores são sistemas de software, geralmente referidos como um intermediário, entre o cliente (interface do utilizador) e o servidor (base de dados), que providencia um ambiente para execução, desenvolvimento e manipulação de Aplicações de processamento de transações. As Aplicações de processamento de transações executam-se sob controlo de monitores de transações e realizam funções computacionais de negócio e acessos à base de dados.

O presente trabalho de licenciatura tem como objectivo, estudar as diferentes formas de processamento de transações em diferentes arquitecturas Cliente/Servidor, fazendo uma comparação entre as formas tradicionais e a mais actual introduzida pela Microsoft, o *Microsoft Transaction Server (MTS)*, baseada no estudo do processamento de transações na Aplicação SNOB, produto da EXI, actualmente a funcionar na CREDICOOP. O trabalho está dividido em quatro(4) capítulos:

O **capítulo 1** dedica-se fundamentalmente à colocação do problema e estudo da sua complexidade através de exemplos típicos de situações nas quais só o uso de ferramentas que controlam eficientemente a execução paralela de sequências de acessos aos contentores de informação, pode garantir a consistência que se deseja que seja permanente em qualquer actividade comercial.

No **capítulo 2** o autor apresenta os monitores de processamento de transações como ferramenta de software que, colocada entre o "Cliente" e o "Servidor da Base de dados", possibilita uma manipulação coerente dos pedidos e respostas, evidenciando como é que o balanceamento da carga, escalabilidade e robustez são garantidas. Uma breve comparação entre os diferentes produtos, também é apresentada como base para a justificação da aplicação do *Microsoft Transaction Server* no SNOB.

Um grande enfoque é dado na definição de conceitos, uma vez que se tratando dum produto novo, a sua funcionalidade e performance seriam infundamentáveis. Assim, a arquitectura, o funcionamento e ligações entre os diferentes elementos que o compõem serão também bem pormenorizados para que as vantagens decorrentes da aplicação deste produto estejam bem assentes nos que ainda possam ter algum receio.

No **capítulo 3** o autor apresenta o modelo proposto para a aquilo que seria a nova arquitectura do SNOB. O modelo baseado em três níveis: Cliente, Componentes e Base de dados, resultante da aplicação do *MTS*, é proposto de forma a solucionar os problemas encontrados em arquitecturas tradicionais de sistemas de processamento de transações.

Os **capítulos 4 e 5** são a última parte, na qual o autor apresenta algumas considerações e recomendações no sentido de mostrar que, nem sempre é necessário deitar abaixo um determinado sistema de base de dados quando se pretende migrar para um outro com outra natureza. Há sempre uma maneira de garantir que se trabalhe com bases de dados de diferentes plataformas (Informix, Oracle, DB2, Sybase), garantido o conceito de transação presente em qualquer tipo de operação de acesso às mesmas.

OBJECTIVOS GERAIS

1. Estudo do processamento de transações em sistemas Cliente/Servidor.
2. Análise dos conceitos do processamento de transações numa Aplicação de gestão bancária, SNOB.

OBJECTIVOS ESPECÍFICOS

1. Análise do problema de transações em sistemas complexos.
2. Implementações Cliente/Servidor de 2 e 3 níveis.
3. Implementações usando bases de dados distribuídas.
4. Utilização de servidores de Aplicações e Monitores de transações.
5. Estudo da implementação do processamento de transações no SNOB (filosofia de trabalho).
6. Elaboração dum modelo usando o *Microsoft Transaction Server* (MTS).
7. Programação dum módulo de teste usando o MTS.

1.0 O PROBLEMA

Em 1687, o Sr. Isaac Newton[1] descobriu a primeira lei do comércio. Ele não a considerou como tal. Ele denominou-a a terceira lei do movimento. Ele disse que qualquer acção dá origem a uma reacção com mesma força, mas com direcção oposta (Sessions,1998).

De facto esta lei descreve bem o comércio, e é a pedra angular do comércio. Para cada compra há uma venda. Para cada levantamento há um depósito. Para cada empréstimo há um débito.

Tomemos a primeira lei do comércio como a seguinte, " todo o movimento do comércio numa direcção é acompanhado por um movimento equivalente em direcção oposta". Tomemos como exemplo a compra de passagens aéreas. Esta pode ser uma boa ilustração da primeira lei do comércio. A companhia aérea dá ao cliente a passagem aérea. O cliente dá o valor monetário correspondente. Dois movimentos que são equivalentes, mas opostos. A companhia estaria a violar a primeira lei se levasse o dinheiro, mas não dando ao cliente a passagem. O cliente estaria a violar a lei se levasse a passagem, mas não dando o dinheiro à companhia.

Há muitas conclusões que podem ser tiradas desta lei, consideremos algumas:

1. Se um destes movimentos iguais, mas opostos, ocorre, o outro tem de ocorrer também. Se o cliente paga o dinheiro, então a companhia tem de dar a passagem.
2. Se um destes movimentos iguais, mas opostos, não ocorre, o outro não ocorre nunca. Se por qualquer razão o cliente não pagar, então a companhia não dá a passagem.
3. Poderá haver mais do que dois movimentos a medida que o sistema vai funcionando e o balanço for mantido. O cliente pode pagar a metade da passagem a agência de viagens e outra metade pelo visto, desde que os dois pagamentos cubram o total da passagem.
4. O futuro não pode anular o passado. Uma vez a passagem atribuída ao cliente, ele tem-na e nada pode mudar isso.

1- Isaac Newton, grande cientista dos sécs. XVII e XVIII

As pessoas do comércio tem um termo especial para descrever a colecção das forças balanceadas descritas na primeira lei. Elas chamam esta colecção de transação. A transferência do dinheiro do cliente requer um levantamento balanceado por um depósito correspondente, assume-se que estas duas actualizações de contas são contidas numa transação. Baseados na discussão da primeira lei, saberemos que todos os elementos da transação terão lugar em um **todo** ou **nada**. E saberemos que uma vez completada a transação, não poderá ser nunca anulada.

Informaticamente, a transação, como visto anteriormente na introdução, muitas das vezes implica actualizações de vários itens de dados, geralmente relacionados. Assim quando a transação terminar é fundamental que e as partes envolvidas, cliente e sistema de base de dados, fiquem satisfeitas como resultado ou da realização com sucesso de seus objectivos ou como resultado do retorno ao seu estado inicial em caso de falhas de ambas partes. No fim o que se pretende é que o cliente tenha a passagem aérea ou continue com o seu dinheiro. Para tal a transação deve ser provida das propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade), que realmente espelham as conclusões tiradas em relação a primeira lei do comércio, onde:

- **Atomicidade,** assegura que todas as actualizações feitas ao longo da operação são realizadas (tornadas duráveis), ou são abortadas e retornadas ao estado anterior. Primeira conclusão.
- **Consistência,** significa que a operação é uma correcta transformação do estado do sistema, preservando os invariantes do estado. Segunda conclusão.
- **Isolamento,** protege as operações concorrentes de terem acesso a resultados parciais das outras (portanto ainda não realizadas), o que a acontecer criaria inconsistências no estado da aplicação. Terceira conclusão.
- **Durabilidade,** significa que as modificações realizadas em recursos manipulados (tais como registos de tabelas), sobrevivem a falhas, incluindo a falhas de comunicações e do sistema. Quarta conclusão.

Tomando como exemplo o caso de levantamento dum valor monetário K, numa determinada conta, contaID, em maior parte das linguagens, estas propriedades são asseguradas usando uma estrutura de programação que não foge da seguinte:

```
SELECT saldo
FROM conta
WHERE cod_conta=ContaID

IF (saldo- K)>0 THEN 'considerando que o sistema não permite saldos negativos'
(1) BEGIN TRANSACTION
    UPDATE conta SET saldo=saldo-K WHERE cod_conta=contaID
(2) IF status=0 THEN 'se não houve problemas
(3)     COMMIT WORK  'Realiza a transação, terminando com sucesso
    ELSE
(4)     ROLLBACK WORK  ' Não realiza, retornando ao estado inicial
    END IF
ELSE
    ERROR "O montante requerido deixa a conta a descoberto"
END IF
```

Onde:

1. *BEGIN TRANSACTION*, considerando que pode haver mais duma instrução, instrui ao sistema que todo o bloco de instruções deve ser executado como uma unidade única, como se duma única instrução se tratasse (atomicidade e isolamento).

2. A pergunta que é feita antes da realização do trabalho (*commit work*) é muito importante, pois é a partir dela que são garantidas as outras duas propriedades (consistência e durabilidade). Quando uma das instruções envolvidas na transação tiver tido alguma falha, então é feito o *Rollback* do trabalho (4), garantindo que o sistema retorna ao seu estado inicial, caso contrário as actualizações são feitas e tornadas duráveis em (3), *Commit Work*.

1.1 COMPLEXIDADE DO PROBLEMA

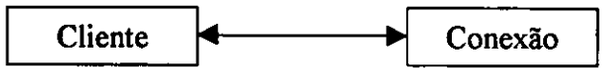
O problema de transações, dentre os vários cenários nos quais pode ser analisado, um deles é a execução paralela de várias actualizações em diferentes recursos da base de dados. Realmente, em sistemas de dois níveis, nível de interface do utilizador e nível de manipulação de dados, como a aplicação de gestão bancária, Snob, foi implementada, há uma ligação um-a-um que se traduz numa situação de "cliente-conexão" cada vez que um cliente acede ao sistema (ver anexo1: Arquitectura de dois níveis do SNOB).

Quando um utilizador entra na aplicação, estabelece-se uma conexão entre este e o sistema de base de dados. **A medida que o utilizador for fazendo pedidos de acesso, portanto através de *queries* (consultas), dentro da conexão estabelecida, processos em quantidade proporcional as *queries* executadas, são abertos**. Porque o sistema não é capaz de identificar o "dono" de cada pedido, este é considerado como sendo dum utilizador diferente e é assim tratado como uma nova conexão.

Assim, os processos para cada conexão vão aumentando de tal forma que, quando superarem a quantidade de processos partilhados suportados pelo sistema, este já sobrecarregado, praticamente vai abaixo, deixando muitas das vezes processos "pendurados", sem "dono".

Tomemos a seguinte situação:

Fase 1 – O cliente entra na aplicação. Temos a seguinte relação: um cliente para uma conexão.



Fase2 – o cliente executa a seguinte transação:

```
{
|

BEGIN TRANSACTION
SELECT cod_conta
FROM conta
WHERE  cod_conta=ContaID
```

```
|
INSERT INTO mov_conta VALUES (..., ContaID,"D",...)
UPDATE conta SET saldo=saldo+valor WHERE cod_conta=ContaID
|
}
```

Neste caso, cada instrução SQL, aqui executada é tratada como uma nova conexão, provocando implicitamente, uma relação de um cliente para várias conexões.

Naturalmente, a medida que a quantidade de utilizadores vai aumentando, várias situações acima referidas acontecem, levando consequentemente à queda do sistema. Devido ao carácter aleatório, presente na forma de acesso á aplicação por parte dos utilizadores, tomemos particularmente uma circunstância na qual estão 1000 clientes conectados.

Se estes clientes todos, forem dados todos os recursos que precisam no servidor, tipicamente uma conexão de comunicação, metadade dum Mbyte de memória, um ou dois processos, e uma dezena de ficheiros abertos, mesmo um largo servidor *mainframe*, poderia ir abaixo (ver figura 1). Felizmente nem todos estes clientes requerem serviços ao mesmo tempo. No entanto, quando eles requerem serviços, querem-nos imediatamente. Tem se dito que o Homem tem uma "tolerância de espera" de dois segundos ou menos.

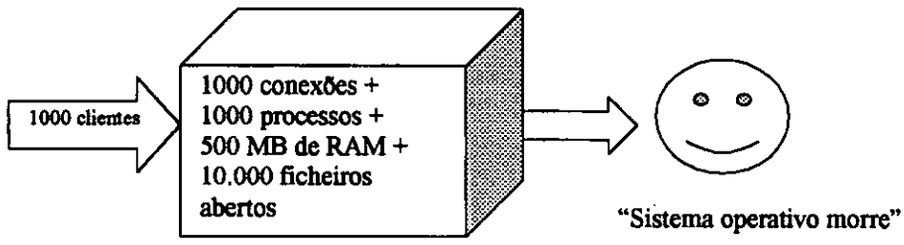


Fig 1

O sistema vai abaixo devido a sobrecarga imposta.

Porque a carga do sistema aumenta consideravelmente, há uma necessidade de o sistema ser capaz de responder a esse crescimento. Uma solução seria iniciar automaticamente outros processos quando as solicitações superam a quantidade de processos partilhados. Mas isto realmente sobrecarrega o sistema podendo assim ir abaixo.

Outra solução seria o uso de processos já anteriormente alocados, para novos pedidos de acesso. O problema encontrado aqui está relacionado com o estado do processo pois, se por exemplo, alguma variável global tenha de ser actualizada em algum processo, o seu reuso pode influenciar negativamente o resultado da nova chamada. O uso de processos pré-alocados só seria efectivo se realmente os processos duma chamada a outra não mantivessem algum estado.

Estes problemas estão relacionados com a necessidade de **balancear a carga** do servidor para dar resposta a um crescente número de solicitações.

O balanceamento da carga, não está ligado somente á capacidade de resposta a um crescente número de solicitações. Aquando da queda do sistema, colocar-se-ia a hipótese de acréscimo de novas componentes (por exemplo, um novo servidor de base de dados). Neste caso, haveria necessidade de reprogramação?, garantir-se-ia a distribuição da carga e a consistência resultante da execução completa das transações?. Portanto, estamos diante da necessidade dum desenvolvimento escalável do sistema, estamos diante da necessidade de ter um controlador, um sinaleiro que coordene o fluxo e encaminhamento dos pedidos.

Para aplicações que controlam milhares de transações por dia e tantos utilizadores, "seria loucura não usar monitores de transações", disse Michael Prince, Director do sistema de informação do armazém principal da fábrica de casacos em Bulrington, N.J.(Radosevich, 1998).

Prince devia saber. Três anos atrás, decidiu mudar o sistema de informação dum "velho" *mainframe* para um ambiente cliente / servidor. Por algum momento, estava duvidoso em relação ao facto de que uma simples base de dados relacional poderia suportar 150Gb de dados necessários para tratar mais de milhares de itens em duzentos (200) armazéns.

Para balancear a carga e assegurar um bom tempo de resposta, ele e a sua equipe dividiram a aplicação em dezassete (17) bases de dados Oracle em quatro (4) servidores. Cada base de dados correspondia a divisão de mercadorias, tais como casacos de senhoras ou roupa de homens.

Apesar do desenho sólido, o sistema foi abaixo aquando do fracasso duma das simples bases de dados Oracle, criadas.

De maneira a manter o sistema em funcionamento, Prince instalou um monitor de transações. O monitor divide uma simples transação de vendas, por exemplo, a compra de duma mala e dum roupão, e envia mensagens diferentes as respectivas bases de dados *back-end*[1]. Se uma delas falha, o monitor guarda a mensagem numa fila e manda-a logo que o servidor se restabelece ou encaminha a mensagem a um servidor disponível. Enquanto isso, a outra base de dados actualiza-se sem demora. "*Assim o sistema é muito resistente e tem alta disponibilidade*", disse Prince.

"Afinal estamos a caminhar para o uso de monitores de transações para a solução dos problemas do balanceamento da carga, segurança, roteamento dinâmico e acessos a múltiplas bases de dados".

2.0 USO DE MONITORES DE TRANSAÇÕES

" A ideia de sistemas distribuídos sem a gestão de transações, é como uma sociedade sem leis. Uma sociedade que não precisa necessariamente de leis, mas que precisa de ter uma maneira de resolver interesses quando disputas ocorrem" (Orfali et al. 1996).

Os monitores de processamento de transações, não são algo novo. Eles são originários dos tempos dos *mainframes*. Muitas bases de dados em *mainframe* incluem os monitores de transações que gerem processos e coordenam acessos a base de dados. Os monitores de transações controlam-nas desde o começo até ao fim, desde o cliente até ao servidor e vice-versa (Jim e Reuter, 1992).

1- back-end, parte referente especificamente ao armazenamento de dados.

As transações são um "tudo ou nada". Elas ou servem ou não, e nunca são incompletas. Assim pode-se contar com os monitores de transações para o controlo da manutenção do sistema num estado estável. Isto dá uma consistência e um modelo de programação fiável e torna os monitores de transações um ambiente natural para aplicações distribuídas que devem trabalhar com muitas bases de dados, filas e interfaces em lotes *(batch)* que correm em plataformas heterogéneas (Jim e Reuter, 1992).

Os monitores de processamento de transações, processam transações do cliente e podem fazer uma distribuição das mesmas em muitos sistemas diversos. É usual ver os monitores ligados a *mainframes*, com um servidor de Windows NT, um servidor de multiprocessamento Unix e um servidor de ficheiros em conjunto. Estes monitores providenciam também um balanceamento da carga, controlo dos processos e a habilidade de recuperação automática em caso de problemas típicos de sistemas (ver anexo2).

Os monitores de transações podem multiplexar e fazer a gestão de transações para reduzir a quantidade de conexões e da carga de processamento que maior parte dos sistemas colocam no servidor da base de dados. Pelo uso dos monitores na estrutura das Aplicações, é possível incrementar o número de clientes sem aumentar o tamanho da base de dados. Como?

Com o monitor de processamento de transações na camada intermédia, remove-se a ligação "um – a - um" que era a típica limitação do "cliente - conexão" no desenvolvimento tradicional de dois níveis. Quando a quantidade de solicitações do cliente supera o número de processos partilhados, o sistema pode suportar outros processos iniciados automaticamente.

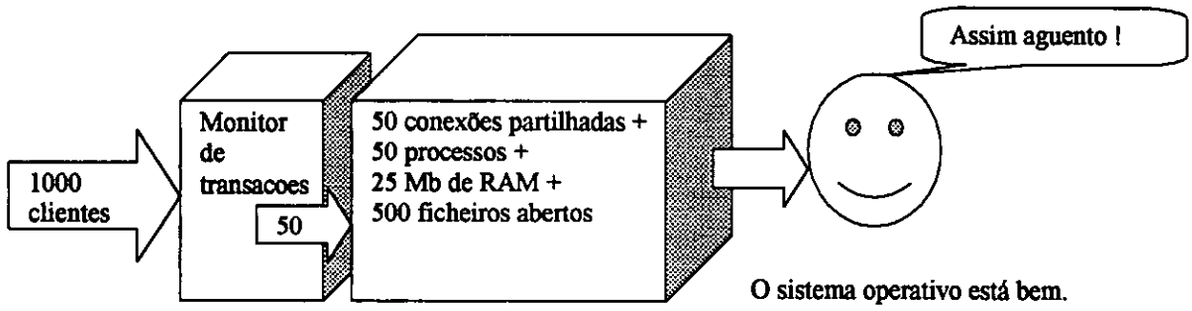


Fig. 2

A maravilha do uso de monitores de transações.

A figura 2, ilustra uma das potencialidades dos monitores de transações, que praticamente reduziu 80% da carga que o sistema tinha. Isto foi graças ao facto de que os monitores de processamento de transações providenciam um "sistema operativo", no topo dos existentes, que conecta em tempo real aqueles milhares de clientes impacientes, com um grupo de processos servidores partilhados (Orfali et al, 1996).

Em ambientes de PCs, o lado servidor da Aplicação de processamento de transações é tipicamente empacotado como uma DLL que contém uma quantidade de funções relacionadas. O monitor de transações atribui a execução das funções DLL a **classes do servidor,** que são grupos de processos anteriortmente iniciados, esperando algum trabalho. Cada processo na classe do servidor é capaz de fazer algum trabalho. (Nota: estas não são classes no sentido do Mundo de *object-oriented.*)

Quando um **cliente** envia um pedido de serviço, o monitor atribui-o a um processo disponível no grupo da classe do servidor. O processo servidor liga-se dinamicamente à função DLL chamada pelo cliente, invoca-a, controla a execução, e retorna os resultados ao cliente. Depois de completado, o processo servidor pode ser reusado por um outro cliente. O sistema operativo mantém na memória as DLLs já invocadas, onde poderão ser partilhadas através dos processos.

Esta potencialidade, aliada à escalabilidade e robustez das Aplicações que usam os monitores, fazem com que os monitores de transações sejam um elemento indispensável no desenvolvimento e disposição de aplicações distribuídas.

Antes foi dito que um dos benefícios dos monitores de transações, é o controlo de transações que afectam sistemas em plataformas heterogéneas, exemplo disso, seria uma transação que afecta uma base de dados Informix e Oracle. Há uma necessidade de estabelecer regras e procedimentos para que haja comunicação entre os sistemas envolvidos e para que os efeitos da transação se tornem duráveis em ambos.

2.1 O TWO - PHASE COMMIT

No Mundo moderno é normal encontrar transações feitas em vários sistemas, que podem ser da mesma plataforma (divisão da base de dados em várias da mesma plataforma) ou de plataformas diferentes (divisão da base de dados em plataformas heterogéneas). Deste modo é necessário que se estabeleça mecanismos para uma comunicação efectiva entre as diferentes bases de dados.

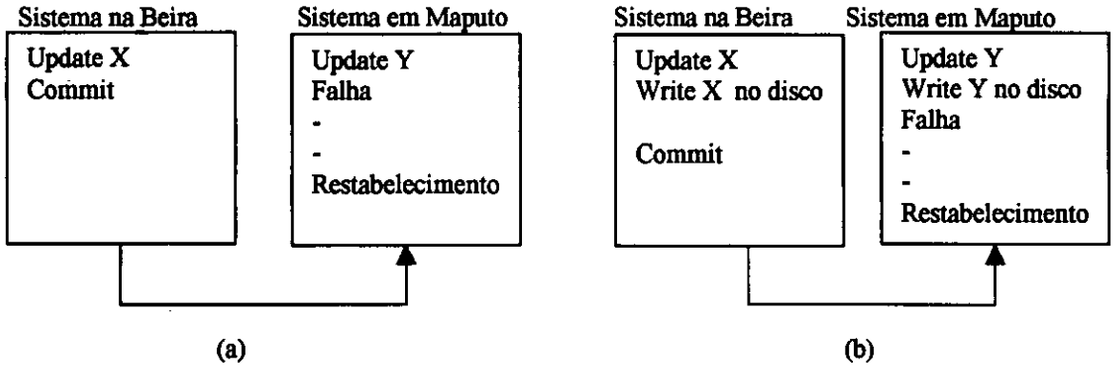
O problema crucial é o facto de que uma transação pode realizar suas actualizações numa base de dados e noutra haver falhas antes da realização *(Commit)*. Quando tal sistema de base de dados se restabelece, tem de ser capaz de realizar a transação.

A realização em duas fases *(two-phase commit)*, uma das características dos monitores de transações, é usada para sincronizar actualizações em várias bases de dados de tal modo que ou todos falhem ou tenham sucesso. Isto é feito pela centralização da decisão de realizar *(commit)*, dando a cada sistema envolvido o direito de vetar. " É como num casamento cristão: aos noivos é dada uma última chance de voltar atrás quando estão no altar. Se nenhuma das partes mostrar objecção, então o casamento tem lugar" (Orfali et al. 1996).

No caso em que um dos sistemas de base dados afectados pela transação falha, quando tal sistema se restabelece deve ser capaz de realizar a transação. Para realizá-la, o sistema em causa tem de ter uma cópia das actualizações da transação. Como o sistema pode perder o conteúdo da memória principal quando vai abaixo, tem de manter uma cópia durável das actualizações, de maneiras que as possa ter depois de restabelecido.

Esta é a essência da realização em duas fases: cada sistema acedido pela transação deve armazenar duravelmente a sua porção de actualizações antes que a transação se realize em algum local. Assim, quando algum sistema falha antes da realização da transação, poderá o fazer depois de restabelecido (Philip e Newcomer, 1997).

Esquematicamente teríamos:



(a) O sistema perde o *Update* a Y quando falha, assim não pode realizar a transação. Este é um caso típico de transações sem *two-phase commit.*

(b) O sistema cria o *Update* no disco antes de falhar, assim pode realizar a transação depois de se restabelecer. Este é o caso *two-phase commit.*

O modelo de processamento de transações distribuídas *X/Open* (Orfali et al, 1996), é uma arquitectura de software que permite aos programas de aplicação a actualização de bases de dados e outros recursos protegidos pela transação . O *X/Open* é originário do "Mundo" Unix.

O modelo *X/Open* descreve como uma aplicação pode usar os monitores de processamento de transações como Tuxedo (Philip e Newcomer, 1996) para actualizar bases de dados em Oracle, sob controlo de transações. O *X/Open DTP* é suportado por todos os monitores de transações da plataforma Unix e por todas as bases de dados Unix, incluindo Oracle, Informix, IBM DB/2 e Sybase.

O modelo original X/Open, é composto por três componentes funcionais e três interfaces funcionais.

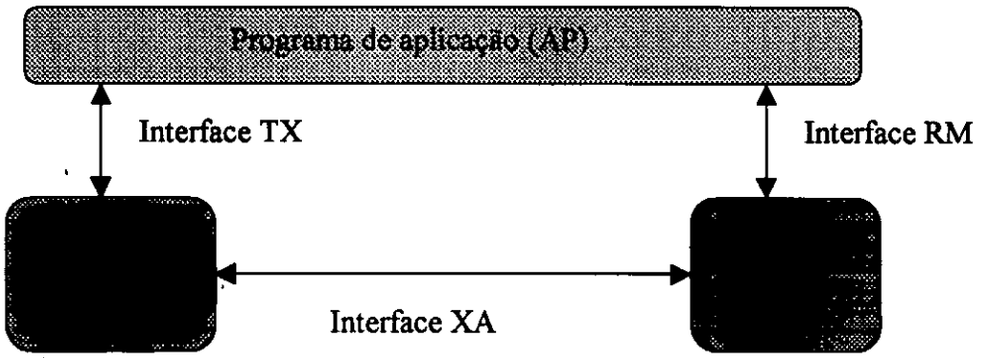


Fig 3

O modelo X/Open

O programa de aplicação contém a lógica da aplicação criada pelo desenhador da aplicação. O programa de aplicação define as fronteiras da transação através das chamadas feitas ao gestor de transações. Controla as operações efectuadas contra os dados da aplicação através das chamadas feitas ao gestor de recursos.

Os gestores de recursos podem ser base de dados, sistemas de ficheiros, sistemas de filas de mensagens e todo outro elemento que providencia o acesso a recursos partilhados. O gestor de recursos típico é uma base de dados relacional.

O gestor de transações cria transações, atribui identificadores de transações às mesmas, monitora o seu progresso e coordena seus resultados. O gestor de transações é responsável pelo orquestramento do processo de *two_phase commit.*

O programa de aplicação comunica com o gestor de recursos usando o interface RM. Tipicamente o providenciador do gestor de recursos define este interface. Por exemplo, uma aplicação pode normalmente usar instruções SQL para comunicar com uma base de dados relacional.

O programa de aplicação comunica com o gestor de transações usando o interface TX. O programa de aplicação usa o interface TX para iniciar e controlar transações. Por exemplo, a aplicação inicia a transação usando *tx.begin.* Realiza a transação chamando o *tx.commit* ou aborta a transação chamando um *tx.rollback.*

O gestor de transações e o gestor de recursos comunicam-se usando o interface XA. As chamadas XA são usadas para incluir o gestor de recursos na transação, para implementar o *two-phase commit,* e efectuar a recuperação depois de quedas do sistema. Por exemplo, *xa_start* inclui um gestor de recursos numa dada transação, *xa_prepare* inicia o processamento da fase 1, *xa_commit* inicia o processamento da fase 2, e *xa_rollback* inicia o *rollback* da transação.

Notar que o interface XA nunca é usado pelo programa de aplicação, é somente usado para a comunicação entre o gestor de transações e o de recursos.

O interface XA é parte do *standard* do *X/Open DTP.* Define o interface que o gestor de recursos e o de transações usam para comunicar. As chamadas XA são usadas para incluir o gestor de recursos na transação, para implementar o *two-phase commit,* e efectuar a recuperação do sistema.

O XA é importante pois é suportado pelos monitores de processamento de transações da plataforma Unix e por todas as bases de dados da mesma plataforma incluindo o Oracle, Informix, DB/2 e Sybase.

2.2 ARQUITECTURA DOS MONITORES DE TRANSAÇÕES

Os monitores de transações implementam o controlo do fluxo de transações, usando um conjunto de componentes de software ilustrados na fig4. Está ilustrada também a estrutura dos processos e das comunicações do controlo do fluxo.

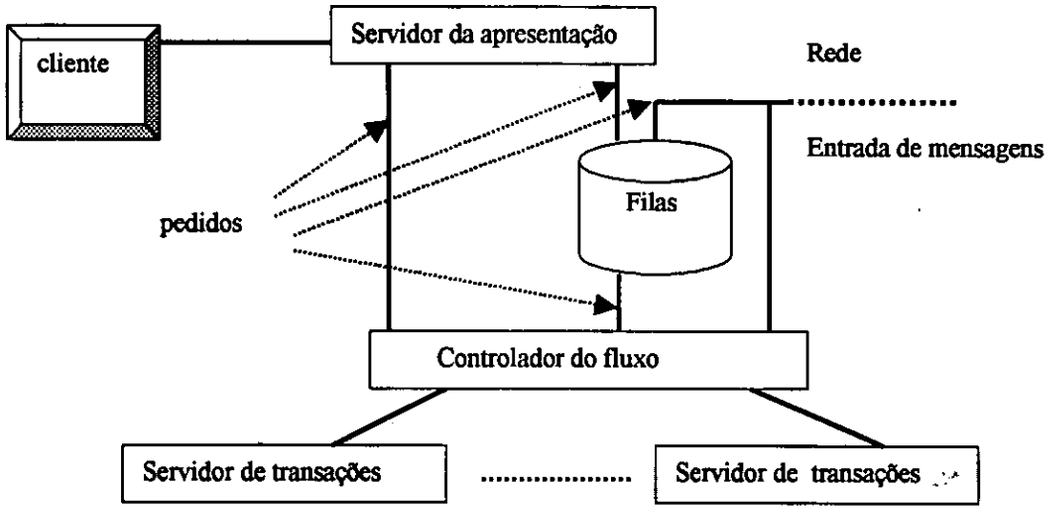


Fig 4

Arquitectura (de três níveis) dos monitores de processamento de transações:
Servidor de apresentação, controlador de fluxo e servidor de transações (Philip e Newcomer, 1997).

O cliente interage com a componente chamada servidor de apresentação, que é responsável pela tradução das janelas (*forms*), selecções de menus e outros inputs num formato *standard* de pedidos. Os serviços de apresentação, estão incrementalmente sendo providenciados por linguagens da quarta geração (4GL), tais como Visual Basic, Powersoft's PowerBuilder e Delphi.

A mensagem de pedido vinda do servidor de apresentação pode ser temporariamente armazenada no disco numa fila, ou pode ser enviada directamente para processamento pela componente chamada controlador do fluxo. O controlador de fluxo encaminha a mensagem de pedido ao próprio servidor de transações e invoca o programa do servidor de transações, que é o programa que faz o actual trabalho do pedido e é onde a transação é realmente executada (Philip e Newcomer, 1997).

2.3 COMUNICAÇÕES

As três componentes dos monitores de transações: servidor de apresentação, controlador do fluxo e o servidor de transações, podem ser organizadas para separar os processos do sistema operativo, e neste caso um dos seguintes mecanismos de segurança é requerido de tal modo a transmitir mensagens duma componente para outra: chamada remota de procedimento (RPC), *peer-to-peer* e filas. Estes paradigmas de comunicação estão especificamente adaptados para uso em ambientes transaccionais, formando um coro da função dos monitores de transações (Philip e Newcomer, 1997).

Uma RPC é um mecanismo de chamada de procedimentos que se executa através dos espaços de endereçamento (dum processo para outro), as vezes localmente no mesmo nó, outras em nós diferentes. A RPC é relativamente fácil de usar porque permite ao programador usar o mecanismo familiar de chamada de procedimento para comunicações remotas, e suporta transparentemente muitos dos aspectos da comunicação.

No entanto, o modelo de RPC introduz algumas complexidades, especialmente no que respeita ao *setup* inicial do sistema, configuração e performance.

Um mecanismo de RPC emprega *proxies*[1] de clientes e *stubs*[2] de servidores. O *proxy* do cliente aceita a chamada local de procedimento do programa chamador, ligando-a ao servidor apropriado, transformando os argumentos do procedimento em mensagem (traduzindo-os numa apresentação certa para a máquina servidora e linguagem de programação), e efectuando a comunicação requerida. O *stub* do servidor transforma a mensagem recebida do cliente numa chamada local de procedimento no procedimento do servidor e envia a mensagem retornada de volta ao cliente.

1- Proxy, objecto que encontrando-se no cliente, estabelece a comunicação com o stub (Philp e Newcomer, 1997).

2- Stub, objecto que encontrando-se no servidor, recebe pedidos do cliente e envia as respostas ao proxy (Philip e Newcomer, 1997).

Alguns sistemas de RPCs, somente suportam *proxy* e *stub* genéricos, sendo assim a disposição *(marshaling*[1]*)* deve ser feita pela aplicação. Outros suportam uma linguagem de definição de interface (IDL) para a definição de interfaces aos procedimentos que podem ser chamados duma forma remota doutros processos. O compilador da IDL gera *proxies* e *stubs* específicos do servidor, que incluem o código da disposição, e os requeridos ficheiros do cabeçalho.

Quando um cliente executa uma transação e chama um servidor, deve passar seu identificador de transação para o servidor. Isto é geralmente escondido ao programador da aplicação para simplificar a programação e para evitar erros.

Para chamar um servidor remoto, um cliente deve estabelecer uma comunicação ligando ao servidor, geralmente pela procura no serviço de directórios. Verificações seguras são também necessárias para autenticar o servidor e assegurar que o cliente está autorizado a chamar o servidor.

As RPCs executam centenas de vezes mais instruções que a chamada local de procedimentos LPC). Elas devem ser cuidadosamente optimizadas para dar uma performance satisfatória.

No modelo *peer-to-peer*, os programas enviam ou recebem mensagens em sequências arbitrárias próximas, não em pares de chamada - resposta. Para comunicar, os programas primeiro alocam uma sessão (conversação), depois da qual podem trocar mensagens.

O modelo peer - to - peer oferece mais flexibilidade em sequências de mensagens e terminações do que RPCs. Esta flexibilidade é resultado do custo da complexidade adicional de programação.

Estes dois modelos constituem um processamento directo de transações, onde um cliente envia um pedido ao servidor e espera, sincronamente, que o servidor execute a transação e retorne a resposta.

1- marshaling, processo de empacotamento e envio dos parâmetros do método do interface através dos limites do *thread* ou processo.

Por exemplo, no modelo de RPC, o cliente envia o pedido ao sistema como uma RPC, que retorna com uma resposta indicando se a transação foi executada ou não. Similarmente, no modelo *peer - to - peer*, o cliente primeiro produz uma operação de envio de mensagem; depois uma operação de recepção da mensagem para esperar que o servidor produza uma mensagem de envio contendo a resposta.

Mesmo que estes modelos sejam largamente usados, na prática há algumas limitações. O primeiro problema está ligado a falha do servidor ou a comunicações cliente / servidor, que impossibilitam a comunicação entre o cliente e servidor. O segundo problema está ligado ao balanceamento da carga, se há um grupo de servidores que podem suportar pedidos de clientes, o mecanismo de ligação do cliente ao servidor deve seleccionar somente um dos servidores no grupo.

Num dado momento, a carga actual pode não estar igualmente balanceada entre os servidores. Isto é, um servidor pode receber muitos pedidos requerendo muito trabalho e assim ficar sobrecarregado. Ao mesmo tempo, outros servidores podem não estar a receber pedidos. Quando a variação da distribuição da carga é alta, esta situação não é muito desejável, criando um tempo de resposta pobre para alguns clientes.

2.4 FILAS COMO SOLUÇÃO

Este conjunto de problemas é resolvido pelo uso de filas como *buffer* para pedidos e respostas entre o cliente e servidor. No lugar de enviar a mensagem directamente ao servidor, o cliente envia-a a uma fila. E o servidor, em vez de receber pedidos directamente do cliente, recebe-as a partir da fila. Similarmente, o servidor envia as respostas para uma fila, e os clientes recebem-nas a partir da fila.

As filas são um recurso transaccional. Assim, as operações na fila são tornadas permanentes ou não feitas, dependendo do sucesso ou não das operações contidas na transação. Geralmente a fila é persistente e é armazenada no disco ou num outro dispositivo de armazenamento não volátil.

Este modelo de processamento de transações em filas, resolve os problemas acima referidos. Sempre que a fila estiver disponível, o cliente pode enviar mensagens mesmo que o servidor esteja ocupado, com falhas ou não conectado. O cliente simplesmente armazena o pedido na fila.

Se o servidor estiver disponível executa-o de maneira correcta. Da mesma maneira o servidor pode enviar a resposta a uma fila de tal modo que logo que o cliente estiver disponível a possa receber.

Muitos servidores podem estar recebendo vários pedidos da mesma fila, assim balanceando a carga em vários servidores. Este balanceamento é totalmente dinâmico. Logo que um servidor termine determinado processamento, pode tomar outro pedido da fila. Nunca em algum momento um servidor está sobrecarregado enquanto outro está livre.

2.5 PRODUTOS DOS MONITORES DE PROCESSAMENTO DE TRANSAÇÕES

No Mundo dos monitores de transações, há cinco maiores "jogadores": Top End da NCR, Encina da Transarc, CICS/6000 da IBM, Tuxedo da BEA e o MTS (Sessions, 1998).

O Encina da Transarc é um monitor de transações muito proximamente acoplado ao DCE (*Open Software Foundation's Distibuted Computing Environments).* O Encina acede bases de dados usando uma arquitectura extendida e funciona em muitas plataformas Unix. O problema deste monitor é a performance. Suas ligações próximas ao limite da RPC em DCE, significa que leva mais alguns ciclos a processar transações. O Tuxedo, o líder do mercado da BEA, providencia a melhor plataforma de suporte. Esta larga plataforma de suporte faz do Tuxedo a aposta mais segura para monitores de transações desde que haja plataformas para corrê-lo.

O Top End da NCR oferece versões para muitas plataformas Unix, incluindo IBM, Sun e NCR. O Top End é somente o segundo em relação ao Tuxedo na partilha do mercado e pode explorar muitos ambientes de multiprocessamento. O NCR ligou-se proximamente ao Top End para os serviços do sistema operativo Unix.

O CICS/6000 da IBM é uma versão do monitor de transações ,CICS, baseado em *mainframe* da IBM. O CICS\6000 trabalha com a linha de servidores RS\6000 da IBM. Pode ser encontrado também no HP/9000s e em poucos outros ambientes Unix. CICS é um monitor antigo que pode correr sistemas existentes CICS que hoje ainda podem ser encontrados.

O MTS é um monitor de transações baseado em componentes (COM e DCOM) que se enquadra perfeitamente na corrente infra-estrutura da Microsoft de desenvolvimento de aplicações da Internet. Seu significado é de que pode fazer dos monitores de transações um produto de comodidade, e deixar qualquer ferramenta COM criar serviços de transação.

2.5.1 MONITORES DE PROCESSAMENTO DE TRANSAÇOES Vs. SERVIDOR DE TRANSAÇÕES DA MICROSOFT (MTS)

Os seguintes monitores de transações interagem com o servidor de transações da Microsoft:

CICS DA IBM

As aplicações MTS podem invocar aplicações CICS correndo em MVS via integrador de transação COM para CICS e IMS (COMTI). O COMTI suporta o segundo nível do protocolo LU 6.2 da IBM para transações distribuídas. Isto permite que o sistema CICS participe numa transação distribuída com o MTS. O CICS da IBM não pode invocar componentes MTS.

IMS DA IBM

As aplicações MTS podem invocar aplicações IMS correndo em MVS via integrador de transação COM para CICS e IMS . O IMS ainda não suporta o segundo nível do protocolo LU 6.2 para transações distribuídas. Como resultado, os programas IMS não podem participar em transações distribuídas com o MTS. O IMS da IBM não pode invocar componentes MTS.

O XA não é um *standard* para a interacção entre monitores de processamento de transações. Este não é o objectivo de *standard*, assim não pode ser usado para fazer com que o MTS possa interagir com outros monitores do processamento de transações.

O XA é um *standard* que define como um gestor de transações se comunica com gestores de recursos. Por exemplo, ele define como o Tuxedo (gestor de transações) se comunica com o Oracle (gestor de recursos). Mais especificamente, descreve como um gestor de transações e gestores de recursos efectuam o *two-phase commit* e a recuperação transaccional.

O XA não define como o gestor de transações coordena transações distribuídas com outros gestores de transações. Por exemplo, o XA não define como o Tuxedo coordenaria transações distribuídas com o MTS, ou como o TopEnd coordenaria transações com Encina.

O XA não define como programas de aplicação se comunicam com outros programas de aplicação. Por exemplo, o XA não define como uma aplicação Tuxedo se comunicaria com uma aplicação MTS, ou como uma aplicação TopEnd se comunicaria com uma aplicação Encina.

Geralmente, as pessoas que não estão extremamente familiarizadas com o *standard X/OPEN DTP*, estão muito confusas em relação a isto. Elas acreditam que o XA define um *standard* para a interacção entre um monitor de processamento de transações e outros. O XA simplesmente é um *standard* que permite que um gestor de transações efectue o *two-phase commit* com um gestor de recursos. Ele permite que o Tuxedo, TopEnd, e Encina coordenem transações com Oracle, Informix, DB2, e Sybase.

2.5.2 O SERVIDOR DE TRANSAÇÕES DA MICROSOFT

O crescimento da Internet e da computação distribuída aumenta a necessidade de se dispor soluções em servidores. A Internet tem crescido num ambiente largamente usado para a publicação e partilha de informação *online*. No lugar de simplificar a publicação de relatórios de saldos e catálogos de produtos, pode-se operar sistemas de contabilidade e sistemas de facturação em servidores, com os utilizadores a acederem as funções partilhadas de negócio desde os *browsers* até aos sistemas em *Desktop*. Este novo ambiente de soluções *online* requerer que as aplicações partilhadas estejam correndo em servidores.

Historicamente, tem havido um problema com esta visão. O desenvolvimento e disposição de aplicações partilhadas em servidores é muito mais difícil do que de sistemas em *Desktop*. Aplicações em servidor precisam de ser mais confiáveis que aplicações em *Desktop*. Aplicações em servidor requerem uma infra-estrutura sofisticada que custa muito a desenvolver e é difícil de manter. Aplicações em servidor precisam de ser mais confiáveis do que aplicações em *Desktop*, porque o impacto de falhas do sistema ou corrupção de dados no servidor pode afectar o negócio em geral, não somente um simples utilizador. Aplicações em servidor precisam de ser fácies de desenvolver e manter como aplicações em *Desktop*, sem requer um treinamento especial ou uma infra-estrutura dispendiosa.

Para resolver estes problemas a Microsoft desenvolveu o servidor de Transações, um produto que combina a flexibilidade e baixo custo de aplicações em *Desktop* com características de processamento de transações em missão crítica normalmente encontradas em sistemas *mainframe*. O servidor de transações da Microsoft, é um sistema de processamento de transações baseado em componentes para o desenvolvimento, disposição e manipulação de aplicações em servidor de alta performance, escaláveis e robustas. Também providencia uma infra-estrutura de *run-time* para a manipulação e distribuição de tais aplicações.

O MTS providencia a maneira mais fácil de executar aplicações robustas e escaláveis no sistema operativo Windows NT.

As aplicações são construídas a partir de componentes *ActiveX* (Sessions, 1998), possibilitando um desenvolvimento facílimo e grande reuso em posteriores desenvolvimentos. As transações são automaticamente incluídas nas aplicações, providenciando uma confiabilidade de missão crítica em aplicações distribuídas.

Pela integração do software componente com uma infra-estrutura de processamento de transações, o MTS elimina os problemas inerentes a construção e distribuição de soluções em servidor.

2.5.2.1 COMPLEXIDADE

O desenvolvimento e distribuição duma solução de servidor em rede, não é tarefa fácil. A implementação da função actual do negócio (ou por exemplo, o tratamento de reservas *online* para uma biblioteca) é actualmente uma fracção do trabalho envolvido. Uma solução escalável também requer uma canalização da infra-estrutura da aplicação sofisticada que possibilita a função do negócio ser partilhada por um largo número de utilizadores. A canalização inclui (Microsoft, 1998):

- **A gestão de recursos do sistema operativo de baixo nível-** por exemplo, processos e *threads* do sistema- assim múltiplos utilizadores podem aceder e executar uma aplicação ao mesmo tempo.
- **Sincronização do acesso a dados partilhados**, de maneiras que a performance não sofra quando múltiplos utilizadores acedem as mesmas partes da base de dados.
- **A gestão de informação sobre utilizadores-** quem eles são e que operações estão efectuando- de tal modo que o trabalho dum utilizador não corrompa ou interfira no trabalho doutro.
- **Implementação da segurança**, de modo a que utilizadores não autorizados não acedam ao sistema.
- **Implementação da gestão e configuração**, de modo que a aplicação possa ser distribuída, gerida e modificada dentro dum custo efectivo.

2.5.2.2 ARQUITECTURA DO MICROSOFT TRANSACTION SERVER

Uma aplicação MTS, consiste no código cliente, objectos do servidor de transações(objectos MTx, que são instâncias de componentes), dispensadores de recursos e gestores de recursos.

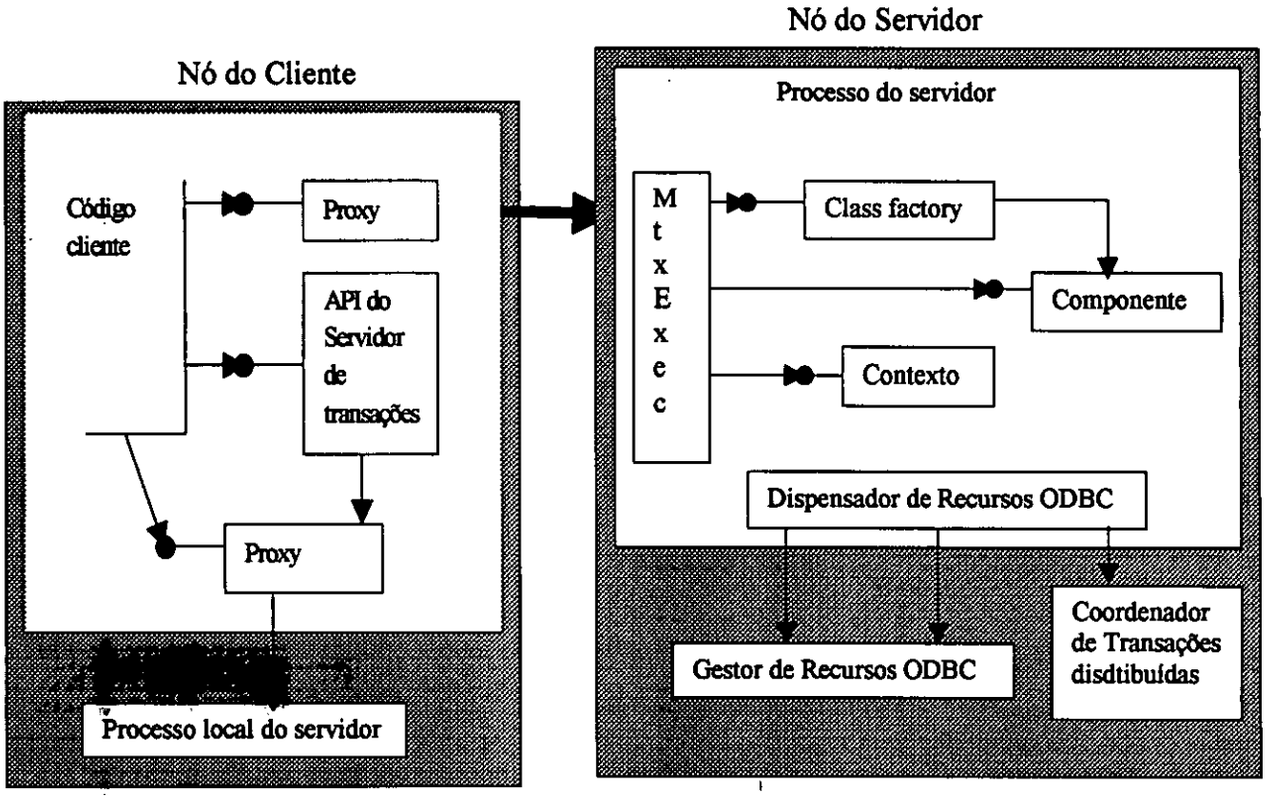


Fig 5.

Arquitectura do MTS. O servidor de aplicações e o API do MTS são componentes por si. O MtxExec implementa muitas das semânticas do MTS; faz a gestão de *threads* e contextos, começa e realiza transações automaticamente (Philip e Newcomer, 1997).

O código cliente é qualquer programa que chama objectos MTx, tais como o servidor de apresentação. Um objecto Mtx pode ser um controlador de *workflow* ou servidor de transações.

Um dispensador de recursos suporta um recurso partilhado não durável. Os dispensadores de recursos são usualmente fornecidos por vendedores de sistemas de software, mas pode ser desenvolvido por um utilizador, se for apropriado. Sabido que muitos monitores de processamento de transações oferecem alguns tipos de dispensadores de recursos, o MTS é o único que faz deles um mecanismo extensível, de tal modo que terceiras partes possam se adicionar por si.

Para dispor e manipular uma componente, o MTS requer uma especificação da componente consistindo numa DLL, definições de interfaces ás componentes, e alguma informação adicional de configuração, tal como quando é que uma componente se executa dentro duma transação. Esta última informação diz se as transações **não são suportadas** (não correm o objecto numa transação), **requeridas** (se o criador do objecto não está correndo a transação, cria uma para o objecto quando este é iniciado), ou **requer novas** (quando o objecto é criado, iniciar uma nova transação mesmo que seu criador esteja correndo uma transação). Estas propriedades declarativas de componentes são usadas para automaticamente criar e manipular transações por parte de quem desenvolve a aplicação. Pelo uso deste mecanismo, em muitas aplicações não se espera que usem mecanismos explícitos de controlo de transações (*Start, Commit, Abort*).

No modelo de programação do cliente, um cliente cria um objecto que precisa e depois chama o respectivo método. O cliente pode começar uma transação antes de criar o objecto, o objecto pode começar a transação no momento em que é criado, ou pode não haver nenhuma transação dependendo nos desejos do cliente e na especificação da transação da componente do objecto.

O MTS realça a programação do COM, pelo suporte do contexto das componentes. O contexto é herdado dum criador do objecto e pode ser influenciado pelos atributos definidos para a classe do objecto. Exemplo da informação do contexto inclui a transação para o objecto e a identidade da segurança.

Uma componente é uma aplicação que nada sabe da sua configuração *run-time*. Isto permite a junção de componentes compradas em companhias independentes e sua disposição numa larga variedade de ambientes de sistemas, sem a modificação da componente no seu todo.

Uma componente pode correr no mesmo processo que o objecto ou código cliente que a cria. A mesma componente pode ser disposta para correr num processo diferente do seu criador sem recompilação. Neste caso, o processo pode estar no mesmo nó (local) ou diferente (remoto) do seu criador.

A escolha para correr num processo com outro objecto é um *trade-off* entre a performance (chamadas entre objectos são mais rápidas dentro dum processo, mas o objecto remoto pode ser colocado junto ao dados que acede) e a protecção contra falhas em outros objectos (objectos estão melhor protegidos quando estão em diferentes processos).

Os objectos MTx, geralmente correm nos processos servidores, que são manipulados pelo MTS. O DCOM é responsável pela ligação das chamadas de métodos a partir do cliente para objectos MTx correndo no processo servidor. O MtxExec propaga o contexto da transação e manipula a atribuição de ordens de execução (*threads*) no processo servidor.

O servidor de transações, enquadra-se no nível intermédio na arquitectura de três níveis, concretamente no nível denominado, nível de componentes. O MTS é o ambiente de componentes. É possível correr componentes em ambientes diferentes do MTS, mas somente este é que providencia a escalabildade, segurança e o suporte transaccional requeridos por aplicações de comércio. Há muitas características importantes do MTS na perspectiva de componentes do negócio.

Retomando o problema do balanceamento da carga, constatado no "cliente – conexão", presente em arquitecturas de dois níveis, caso do Snob, vejamos como o MTS soluciona este problema usando o *object pooling*, que é um requisito absoluto para o suporte eficiente das arquitecturas multi - nível.

2.5.2.3 ARQUITECTURAS DE TRÊS NÍVEIS

Sistemas comerciais baseados em componentes correm em arquitecturas de três níveis, onde se pode separar os computadores de acordo com o objectivo. Torna-se importante referir que esta é uma arquitectura lógica pois, a divisão em três níveis pode acontecer num único computador, em dois computadores ou mais.

Um computador ou grupo lida directamente com o Homem. Estes computadores correm os programas do interface do utilizador, provavelmente escritos em Visual Basic, que é particularmente destinado a este tipo de programas. Estes computadores constituem o **nível de cliente.**

O segundo grupo de computadores trabalha com as componentes do negócio. Estes correm a lógica do negócio escrita em algo como Java ou Visual Basic, que são particularmente destinadas a este tipo de programação. A lógica do negócio é constituída por um conjunto de regras ou procedimentos internos duma determinada organização, por exemplo no caso do banco, o montante mínimo requerido para a manutenção duma conta. Estes computadores constituem a **camada de componentes**. As componentes nesta camada, recebem instruções dos programas do interface do utilizador que correm na camada cliente.

O terceiro grupo de computadores trabalha na manipulação de dados. Estes computadores correm bases de dados largas e complexas, a mais proeminente que a MicroSoft oferece é o *SQL Server*. Estes computadores constituem a **camada de dados**. Este nível interage com o nível de componentes.

Juntos, estes três níveis constituem a arquitectura de três níveis na qual os sistemas que se pretendem que sejam robustos, escaláveis e com suporte do balanceamento da carga devem ser desenvolvidos.

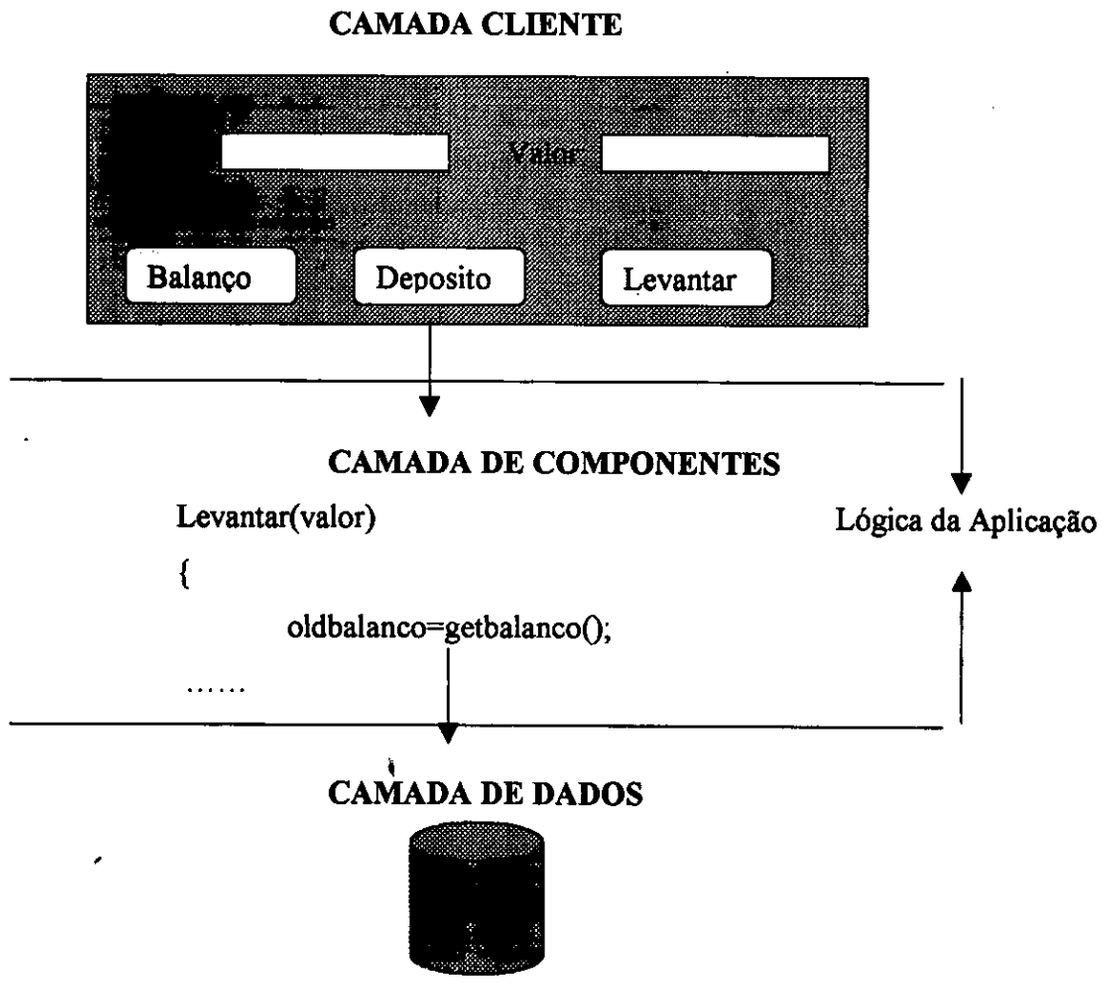


Fig. 6.
Arquitectura de três níveis

Actualmente existem produtos como o Visual Modeler, que possibilitam a disposição das componentes nos diferentes níveis. Por exemplo um projecto desenvolvido em Visual Basic, constituído por *forms*, módulos, classes e objectos pode ser distribuído num ambiente visual oferecido pelo Visual Modeler, indicando-se exactamente o local onde cada componente estará.

2.5.2.4 O MTS e o Object Pooling

Para muitos, a principal característica do MTS é o *object pooling*. Contudo o *object pooling* é fundamental para toda a estrutura da Microsoft e tem importantes ramificações na implementação de componentes.

O *object pooling* é baseado numa simples ideia. Em arquitecturas típicas de objectos distribuídos (sem *pooling*), os objectos esperam pelos clientes. O tempo "morto" típico é enorme para os objectos. Se houver uma maneira de deixar que os objectos sirvam outros clientes durante o tempo que seria morto, podemos aumentar significativamente a quantidade de clientes que o objecto pode suportar.

Com o *object pooling*, os *proxies* não são ligados aos objectos remotos. Assim, são ligados a grupos (*pools*) de objectos. Quando um cliente invoca um método num objecto, um objecto aleatório é escolhido do grupo para servir tal pedido. Quando o pedido é satisfeito (terminado), o objecto é retornado ao grupo para ser usado por outros clientes. Os objectos são mantidos ocupados, e um número largo de clientes pode ser servido por um pequeno grupo de objectos.

O *object pooling* tem duas implicações. Primeiro, um dado objecto pode ser usado por mais dum cliente. Segundo, o cliente não pode contar com um objecto particular servindo qualquer invocação particular de métodos. Desta perspectiva do cliente, um objecto aleatório será escolhido do grupo para servir a seguinte invocação do método.

Os objectos devem ser escritos especificamente para o *pooling*. Isto significa que os objectos devem ser escritos de tal modo que as invocações sejam independentes umas das outras. Em geral os objectos não devem manter dados internos ou o estado entre invocações do método. Tais objectos são chamados "sem estado".

Deve ser esta "falta de estado" que garante que objectos pré-alocados possam servir outras solicitações sem a necessidade de criar novos objectos, resolvendo deste modo os problemas do balanceamento da carga e da escalabilidade abordados em 1.1.

O *object pooling* incrementa a eficiência da camada de componentes em arquitecturas de três níveis. O problema que o *object pooling* resolve é muito simples: se se pretende que o sistema, no seu todo, seja de custo efectivo, **o número de máquinas não caras na camada de clientes deve de longe exceder o número de máquinas caras na camada de componentes** (Microsoft, 1998).

Hoje, a camada de componentes em sistemas de comércio largos, podia correr num *mainframe*. Se a Microsoft for a tomar conta desta camada, deve oferecer uma solução competitiva em termos de custos. Em outras palavras, a Microsoft deve mostrar que pode-se servir mais máquinas clientes numa camada constituída por *Workstations* do Windows NT do que numa camada constituída por *mainframes*, e pode-se fazer assim com baixo custo. O *object pooling* é o centro para este plano.

Em muitos sistemas distribuídos, um programa do cliente correndo na camada cliente, invoca métodos num objecto remoto correndo na camada de componente. Ele faz isto através do objecto *proxy*, que é um objecto que vive no processo do programa cliente mas trabalha enviando métodos para o objecto remoto. DCOM é um exemplo deste tipo de arquitectura.

Se olharmos para a eficiência da camada de componentes dentro do contexto de aplicação de objectos distribuídos, podemos basicamente colocar a seguinte questão: quantos *proxies* podem ser suportados por uma dada configuração de componentes? A resposta é que isto será determinado por dois factores: a quantidade de componentes que podem ser executadas na camada de componentes, e a quantidade de *proxies* que cada objecto pode suportar.

Em geral, é pouco o que se pode fazer para incrementar a quantidade de objectos que podem ser executados numa dada máquina, senão comprando uma maior. No entanto, podemos dramaticamente aumentar o número de *proxies* que cada um destes objectos pode suportar. Aqui é onde o MTS e *object pooling* jogam seu papel.

2.5.2.5 DISPENSADORES E GESTORES DE RECURSOS

O dispensador de recursos do MTS tem duas funções. Faz a gestão dum grupo de conexões ao gestor de recursos e automaticamente alista o gestor de recursos na corrente transação da componente.

O *ODBC Driver Manager* é um dispensador de recursos típico. Faz a gestão dum grupo de conexões à base de dados e alista as conexões na corrente transação da componente. As componentes da aplicação usam chamadas do *standard ODBC API* para usar o dispensador de recursos ODBC.

Gestão do grupo de conexões

O agrupamento de conexões torna barata a conexão e desconexão de componentes *short-lived* aos gestores de recursos.

As componentes da aplicação chamam o dispensador de recursos quando precisam duma conexão a um gestor de recursos. O dispensador de recursos toma seu grupo de conexões ao gestor de recursos já abertas, procurando uma conexão que vá de acordo com as necessidades da componente. Se encontrar uma conexão satisfatória, o dispensador de recursos atribui-a à componente. Se o dispensador de recursos não encontra uma conexão satisfatória, abre uma nova. Quando a componente liberta a conexão, o dispensador de recursos coloca a conexão no grupo para que seja usada por outras componentes. Se a conexão não é usada durante um período de tempo, o dispensador de recursos fecha-a e remove-a do grupo. O administrador do sistema pode configurar o período *timeout.*

Os dispensadores de recursos, geralmente providenciam interfaces administrativos para o controlo do grupo de conexões. Por exemplo, o dispensador de recursos pode permitir ao administrador controlar o tamanho mínimo e máximo do grupo de conexões e o *timeout* para conexões não usadas.

Alistamento automático de transações

O dispensador de recursos alista automaticamente a conexão do gestor de recursos na corrente transação da componente. O dispensador de recursos assim o faz obtendo a corrente transação a partir do contexto do objecto da componente e enviando-a ao gestor de recursos. Fazendo isto, o gestor de recursos faz com que o alistamento de transações seja automático para a aplicação.

Um gestor de recursos é uma base de dados, sistema de ficheiros, sistemas de filas de mensagens ou algum outro elemento que providencie acesso a recursos partilhados. Um gestor de recursos transaccional permite que os dados que gere sejam actualizados sob controlo de transações. **Um dispensador de recursos gere conexões a um gestor de recursos**.

3.0 APLICAÇÃO DO SERVIDOR DE TRANSAÇÕES NO SISTEMA BANCÁRIO

Um sistema desenvolvido dentro do modelo do *Microsoft Transaction Server*, deve necessariamente ser desenhado dentro da arquitectura de três níveis de tal modo que as funções da aplicação, no seu todo ,estejam particionadas garantindo que a carga de trabalho esteja equitativamente subdividida. **Esta subdivisão da carga fará com que os pedidos do cliente sejam executados (processados) no servidor sempre que for possível e no cliente quando necessário (Sessions, 1998).**

Como analisado anteriormente, a presente arquitectura do SNOB, em estudo, apresenta dois níveis: o nível do cliente (interface do utilizador) e o nível do servidor (lógica da aplicação e ligação a base de dados). Pelo redesenho, decorrente da aplicação do MTS, o sistema teria três níveis: o nível do cliente (interface do utilizador), o nível de componentes (lógica da aplicação e ligações a base de dados), e o nível da base de dados (servidor de base de dados). Assim a nova arquitectura seria:

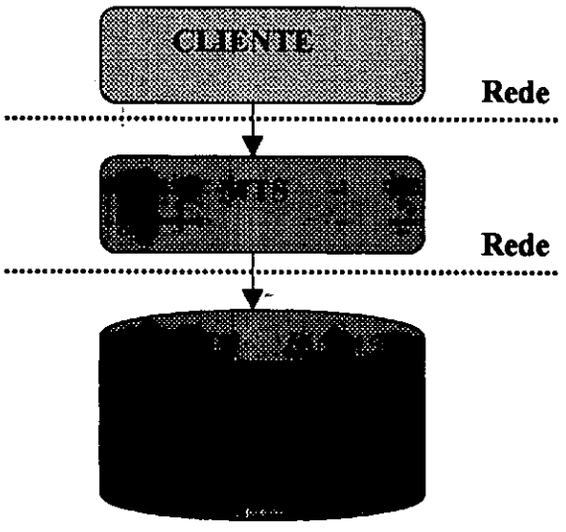


Fig 7.
Arquitectura simplificada do modelo proposto para o SNOB.

3.1 FUNCIONAMENTO DO SERVIDOR DE TRANSAÇÕES DA MICROSOFT

Para uma melhor aplicabilidade do MTS, nada mais é importante do que o conhecimento detalhado do modo do seu funcionamento. Neste ponto veremos como o MTS funciona e como os seus elementos providenciam uma infra-estrutura e um modelo de programação para o desenvolvimento e disposição das suas componentes.

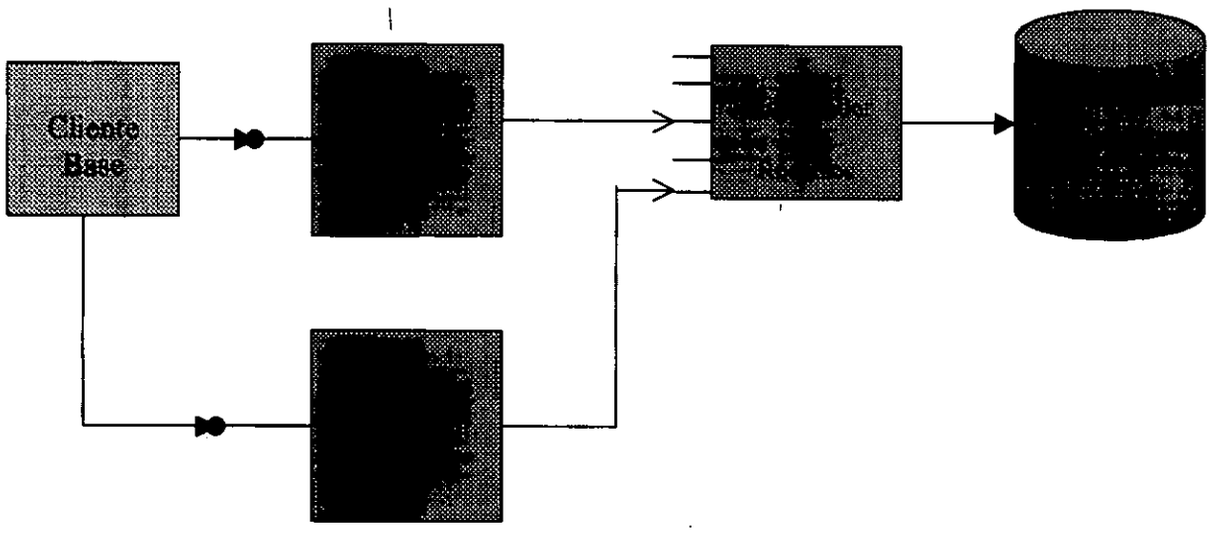


Fig 8.

Interligação dos elementos do MTS

Componente da Aplicação

As componentes da aplicação modelam a actividade do negócio. Estas componentes implementam as *business rules* providenciando visões e transformações do estado da aplicação. Consideremos, por exemplo, o caso do banco. Os registos na base de dados representam o estado durável do negócio, tal como o montante de dinheiro numa conta. As componentes da aplicação actualizam tal estado para reflectir as mudanças resultantes dos débitos e créditos.

Porque as componentes que correm no ambiente MTS podem se beneficiar da vantagem das transações, as aplicações são desenvolvidas de tal modo que corram como se estivessem isoladas. O MTS suporta a concorrência, o agrupamento de recursos, segurança, manipulação do contexto e outras complexidades ao nível do sistema.

O sistema de transações, trabalhando em cooperação com os servidores de base de dados e outros tipos de gestores de recursos, assegura que as transações concorrentes sejam atómicas, consistentes, tenham isolamento próprio, e que, uma vez cometidas, as mudanças sejam duráveis.

O MTS também facilita a construção de aplicações distribuídas pela providência da transparência da localização. Uma componente do MTS pode ser carregada para um processo da aplicação do cliente (componente *in-process*[1]), ou para um ambiente dum processo separado do servidor, ou para o computador cliente (componente local) ou noutro computador (componente remota).

Processo do servidor

Um processo do servidor é um processo do sistema que suporta a execução da componente da aplicação. Um processo do servidor pode suportar múltiplas componentes e podem ser de dezenas, centenas, ou potencialmente milhares de clientes. Pode se configurar múltiplos processos servidores para correrem num simples computador. Cada processo servidor reflecte uma fronteira confiável separada e domínio do isolamento de falhas.

Outros ambientes de processos podem suportar componentes da aplicação. Como resultado, pode-se dispor aplicações que suportam várias distribuições, performance e requisitos de isolamento de falhas. Por exemplo, pode-se configurar componentes do MTS para serem carregadas directamente para o Servidor de Informações da Internet (IIS).

1-in-process, uma componente que corre no espaço do processo do cliente, é tipicamente uma DLL.

O Executivo do MTS

O executivo do MTS é uma DLL que providencia serviços em *run-time* para componentes MTS, incluindo a gestão de *threads* e contextos. Esta DLL carrega-se para o processo que suporta as componentes da aplicação e corre em *background*.

O MTS também providencia um conjunto de dispensadores de recursos que simplificam acessos a recursos partilhados num processo servidor.

Coordenador de transações distribuídas da Microsoft (MS DTC)

O MS DTC é um serviço do sistema que coordena transações. O trabalho pode ser cometido (*commited*) como uma transação atómica mesmo que afecte múltiplos gestores de recursos, potencialmente em computadores separados.

O MS DTC foi primeiro publicado como parte do *SQL Server* 6.5 e está incluído no MTS, providenciando uma infra-estrutura de baixo nível para as transações. O MS DTC implementa o *two-phase commit* para assegurar que o resultado da transação seja consistente em todos os gestores de recursos envolvidos na transação.

Explorador do MTS

A gestão das componentes é feita usando o *MTS Explorer*. Antes que a componente possa correr com o contexto no ambiente do *run-time* do MTS, é preciso definir a componente no catálogo do MTS. Em adição à manutenção dos atributos básicos das componentes, tais como o nome da DLL implementada, o catálogo do MTS mantém um conjunto de atributos específicos do MTS. O MTS usa estes atributos para providenciar capacidades em adição às providenciadas pelo COM. Por exemplo, o atributo da transação controla as características transaccionais duma componente.

O explorador do MTS atribui componentes a um pacote que controla a distribuição das mesmas pelos processos servidores e controla o acesso do cliente às componentes.

3.2 O MODELO PROPOSTO

O modelo proposto consiste no particionamento da arquitectura de dois níveis presente no SNOB, em : cliente (nível 1), servidor de Aplicação (nível 2) e servidor de base de dados (nível 3), pela integração do MTS no nível intermédio.

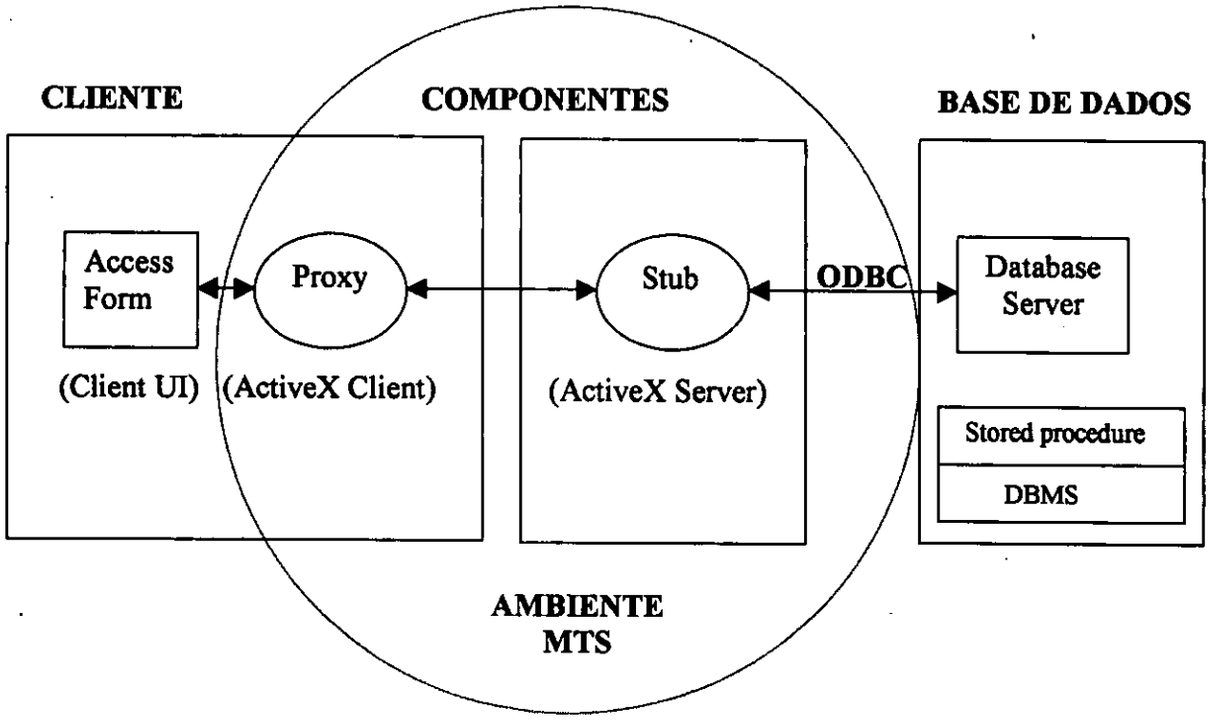


Fig. 9
A nova arquitectura do modelo de três níveis proposto para o SNOB.

3.2.1 NÍVEL DE INTERFACE DO UTILIZADOR

A parte respeitante ao Cliente (Interface do Utilizador), seria basicamente as *forms* de acesso, desenhadas em Visual Basic, para possibilitar que sejam clientes *ActiveX*[1]. A existência dum *proxy* na parte cliente é fundamental para garantir a comunicação, pois o *proxy* cria um interface através da transformação dos parâmetros passados a uma componente do servidor, num formato "compreensível" para o servidor (processo de *marshaling*).

1- Activex, tecnologia que permite que produtos de software de diferentes linguagens, possam ser interligados sem considerar, a linguagem na qual foram programados.

Consideremos, por exemplo, o caso em que o cliente pede o saldo dum conta, pressionando o botão "Balanço". O *proxy*, por exemplo, acrescentaria á mensagem do pedido, o método indicando a linguagem usada para o desenho do cliente (seja VB, para Visual Basic, VJ para Visual Java, VC para Visual C++).

Teríamos o seguinte código:

```
{
-
-
Dim ProgID As String
   If  Balanco= True Then
      ProgID = "Balanco.Conta"
   Else
    If  Levantar=True Then
      ProgID = "Levantar.Conta"
   Else
      ProgID = "Depositar.Conta"
   End If

   ' Decidir  que linguagem usar
   If  VC = True Then
      ProgID = ProgID + ".VC"
   Else
    If  VJ = True Then
      ProgID = ProgID + ".VJ"
   End If
-
-
}
```

3.2.2 NÍVEL DE COMPONENTES

As componentes implementam a lógica do negócio, portanto seriam módulos compilados como DLLs (em Visual Basic, implementados obrigatoriamente como *class modules*), de modo que sejam servidores *ActiveX*. Assim, os procedimentos e regras para levantar, depositar valores são programados e guardados, depois de compilados, em pacotes de software. Isto é feito através do Explorador do MTS (*MTS Explorer*), ver anexo3 para exemplos de algumas componentes .

O Explorador do MTS, permite a definição de componentes e seus pacotes e também a definição de suas propriedades, por exemplo, marcando uma componente como transaccional elimina a necessidade de incluir no seu código os comandos *start, commit ou rollback*. Também permite o controlo da execução das transações, oferecendo um ambiente visual que mostra o estado e as estatísticas (quantidade de transações abortadas, cometidas e o tempo que duram tais transações).

A tecnologia de componentes da Microsoft, COM, seria especialmente a mais adequada para as necessidades desta camada de componentes, pois tem as seguintes importantes características (Sessions, 1998):

- Para os familiarizados com a programação orientada a objectos, as componentes são naturais (fácies).
- As componentes são bem encapsuladas, com interfaces bem definidos.
- Estes interfaces podem ser descobertos (manipulados) no *run-time*.
- As componentes são independentes da linguagem, assim o código correndo na camada do interface do utilizador pode fazer uso da camada de componentes sem ter em conta a sua linguagem de programação (aqui está mais uma vez vincada a necessidade das componentes serem compiladas como DLLs).

O DCOM (Sessions, 1998), estende estas características todas, para providenciar um modelo simples de distribuição. Uma vez, as componentes entendidas no geral, a extensão de ter o código na camada do interface do utilizador fazendo chamadas remotas a componentes na camada de componentes é simples.

O modelo DCOM para a distribuição é mais simples de compreender do que, por exemplo, *TCP/IP Sockects*, ou mesmo chamadas remotas de procedimentos (RPCs)- outras duas tecnologias usadas geralmente para a comunicação entre a camada do cliente e a de componentes.

3.2.3 NÍVEL DA BASE DE DADOS

Finalmente teríamos a camada de gestão de dados, onde os dados são armazenados e recuperados. Neste ponto não há muito a dizer, pois através da configuração do ODBC, os gestores de recursos podem muito bem ser conectados ao MTS, o que se requer unicamente é que tais bases de dados suportem o protocolo XA.

A base de dados do "SNOB", está implementada em Informix-Online e este suporta este protocolo, precisando unicamente dum *Driver ODBC* compatível com o MTS. A seguir é apresentada uma tabela de diferentes bases de dados que funcionam com o MTS e outras em que os seus representantes estão trabalhando na produção de *Drivers ODBC*, e que num futuro muito próximo integrar-se-ão facilmente com o MTS.

BASE DE DADOS	**COMENTÁRIO**
SQL Server	O SQL Server 6.5, suporta completamente o MTS
Oracle	O Oracle 7.3.3 e 8 são bem suportados completamente na versão 2.0 do MTS
DB2	O MTS trabalhará com a versão 5.0 do DB2 no Windows NT. Uma aplicação MTS poderá conectar uma base de dados DB2 em Windows NT, via TCP/IP, SNA, NETBIOS ou SPX/IPX.
Informix	O MTS trabalhará com Informix em Windows NT ou UNIX.
Sybase	O Sybase está planeando o suporte do MTS.

Tab1.

Tabela de bases de dados que já suportam o MTS e outras que suportarão (Microsoft Faq, 1998)

4.0 CONCLUSÕES

1. Arquitecturas de dois níveis tem limitações no balanceamento da carga aquando do crescimento da quantidade de solicitações. Não são flexíveis quando for necessária a integração de novas componentes.

2. Em sistemas Cliente/Servidor de três níveis, a lógica da aplicação (ou processo) vive na camada intermédia separada dos dados e do interface do utilizador. Os processos tornam-se "cidadãos de primeira classe", podem ser manipulados e dispostos separadamente do interface do utilizador e dados. Arquitecturas de três níveis são mais escaláveis, flexíveis e robustas.

3. Os monitores de processamento de transações são desenhados para trabalhar em volta de todos os tipos de falhas. A penetração dos princípios ACID em todas as componentes ajuda a criar sistemas auto-recuperáveis. Os monitores estão sempre por cima do estado dos recursos cliente/servidor sob seu controlo. Com os princípios ACID é possível detectar o local exacto onde ocorreu a falha.

4. O MTS é um monitor de transações que providencia a maneira mais fácil para correr Aplicações escaláveis e robustas no sistema operativo Windows NT. Pela integração com o servidor Windows NT, a família de produtos *BackOffice*, com os sistemas existentes, reduz-se os custos de disposição e gestão das Aplicações.

5. No modelo proposto, pela integração do software baseado em componentes com a infra-estrutura de processamento de transações, o MTS elimina os problemas de construção e disposição de soluções em servidor.

6. O COM é uma tecnologia usada para definir componentes e não uma tecnologia para implementar componentes, para isto usa-se Java, Visual Basic ou Visual C++. O DCOM é uma tecnologia que estende o COM para possibilitar que objectos componentes vivam em máquinas remotas.

5.0 RECOMENDAÇÕES

1. Os monitores do processamento de transações, tornam-se pouco importantes em arquitecturas de dois níveis, onde a disparidade dos recursos computacionais é limitada, onde os sistemas são baseados num único recurso, numa única plataforma ou limitada, onde as capacidades de comunicação são limitadas, onde os *standards* não são tomados em conta. Portanto, os monitores são bem aplicáveis em:

   - Múltiplas bases de dados.
   - Múltiplas redes Informáticas .
   - Crescimento do número de utilizadores.
   - Ambientes múltiplos de computação.
   - Necessidade de alta disponibilidade e performance.

2. Implementação do modelo proposto, tendo em conta que para o caso do SNOB, a base de dados foi implementada em Informix-OnLine, sendo para tal necessário o *Driver ODBC* compatível com o MTS 2.0, que até ao fim deste trabalho ainda não tinha sido produzido pela Informix.

3. Testar a nova versão do SNOB, já com o MTS, contra a versão actual em termos de performance. Para tal é necessário utilizar os padrões do TPC (*Transaction Proccesing Performance Council*), que definem medidas de quantidade de transações por unidade de tempo e o respectivo custo.

4. Testar o *two-phase commit*, pela integração dum novo servidor de base de dados diferente do Informix-OnLine, por exemplo o SQL Server (o mais recomendado pela Microsoft).

6.0 BIBLIOGRAFIA

6.1 REFERÊNCIAS BIBLIOGRÁFICAS

[Jim e Reuter, 1992] Jim G., e A. Reuter (1992). **Transaction Processing**. 1070 pp, San Mateo, Morgan Kaufmman.

[Orfali at al, 1996] Orfali, R., D. Harkey e J. Edwards (1996).**The Essential Client\Server Survival Guide.** 2ª Edição, 644 pp. U.S.A, John Wiley & Sons.

[Philip e Newcomer, 1996] Philip, A. B. e E. Newcomer (1997). **Principles of Transaction Processing**. 364 pp, San Mateo, Morgan Kaufmman.

[Sessions, 1998] Sessions, R. (1998). **COM and DCOM: Microsoft's Vision for Distributed Objects**. 472 pp, U.S.A, John Wiley & Sons.

[Microsoft, 1998] Microsoft, (1998). http://www.microsoft/transaction.com

[Microsoft Faq, 1998] Microsoft Faq, (1998). http://microsoft.com/support/transaction/content/faq/

[Randosevich, 1998] Radosevich, (1998). **What is Middleware?** http://tuxedo.novel.com/products/mid/articles/newwart4.htm

6.2 BIBLIOGRAFIA CONSULTADA

1. Claybrook, B (1992). **OLTP (Online Transaction Processing Systems)**. 334 pp, U.S.A., John Wiley & Sons.

2. Lynch N., M. Merritt, W. Weil, e A. Fekete (1994). **Atomic Transactions**. 485 pp, San Mateo, Morgan Kaufmman.

3. Rob, P., e C. Coronel (1995). **Database Systems**. 679 pp, 2ª edição, U.S.A., boyd & fraser (bf).

4. Smith, P. (1992). **Client \ Server Computing**. 291 pp, U.S.A., Sams.

5. Yourdon, E. (1994). **Object-Oriented Systems Design (An Integrated Approach)**. 1ª Edição, 400 pp. New York, Prentice-Hall, Prentice-Hall, inc.

ANEXO1 : ARQUITECTURA DE DOIS NÍVEIS DO SNOB

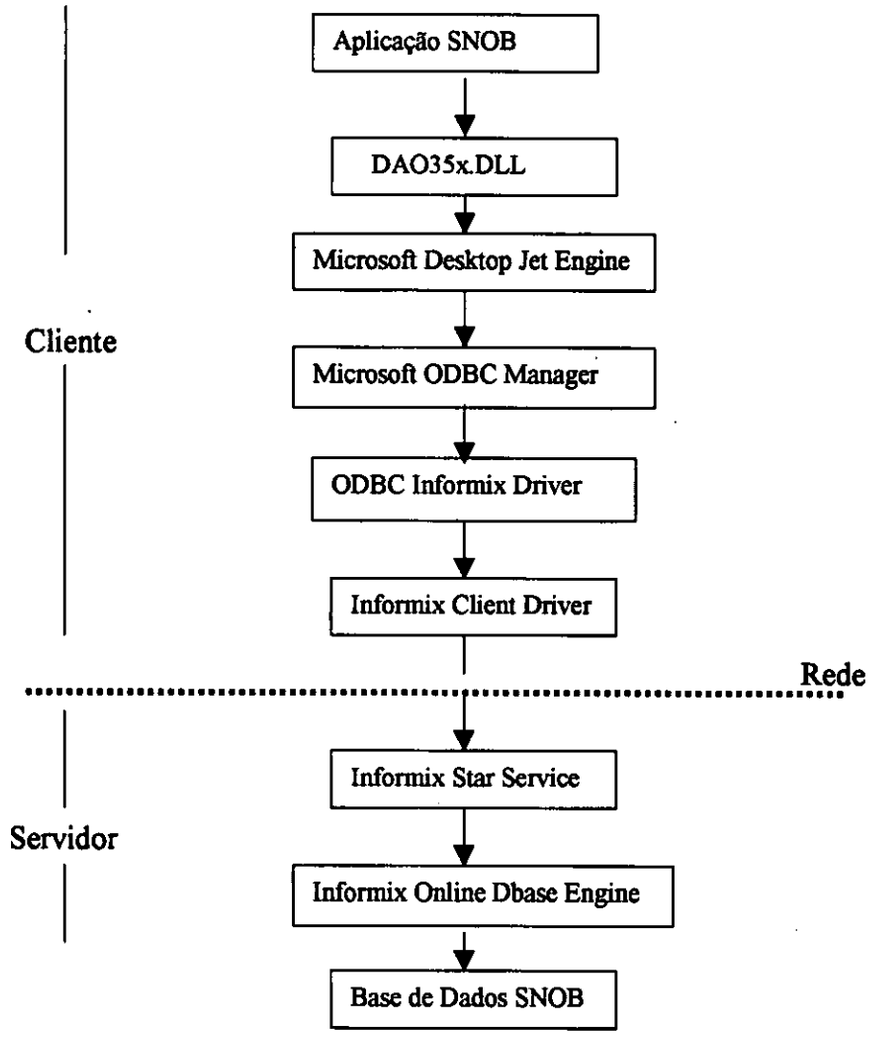
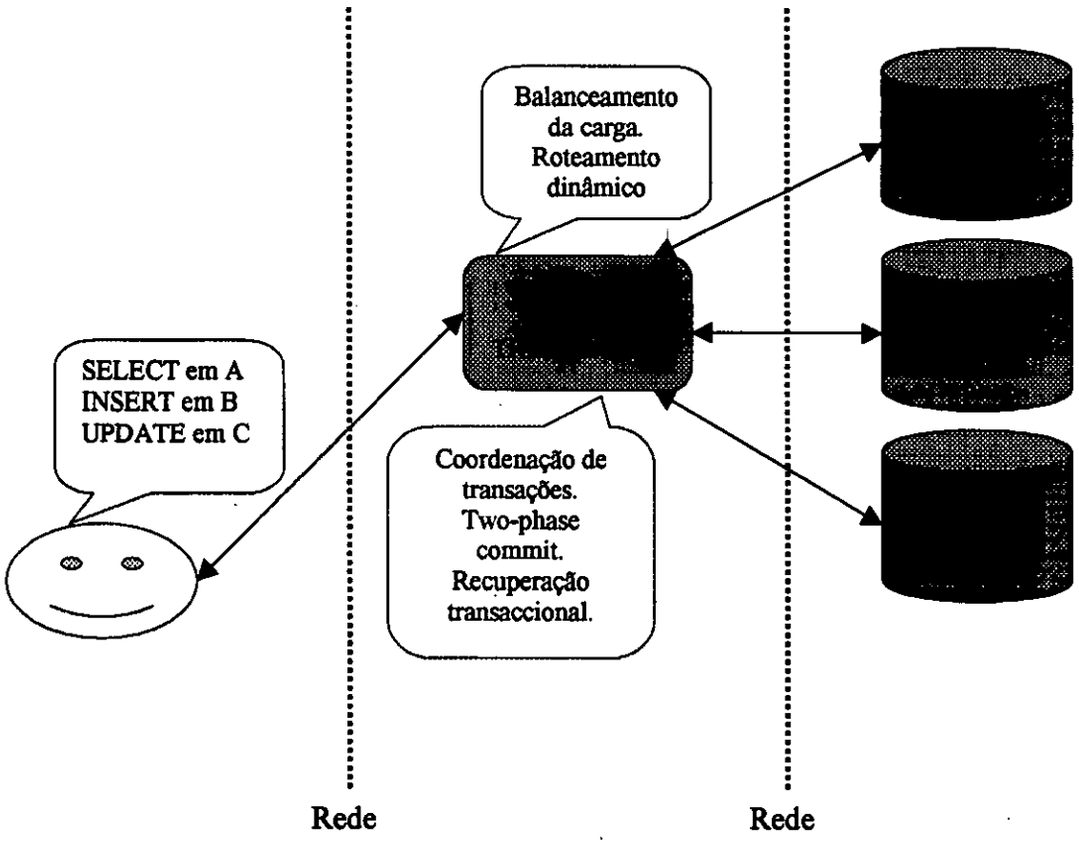


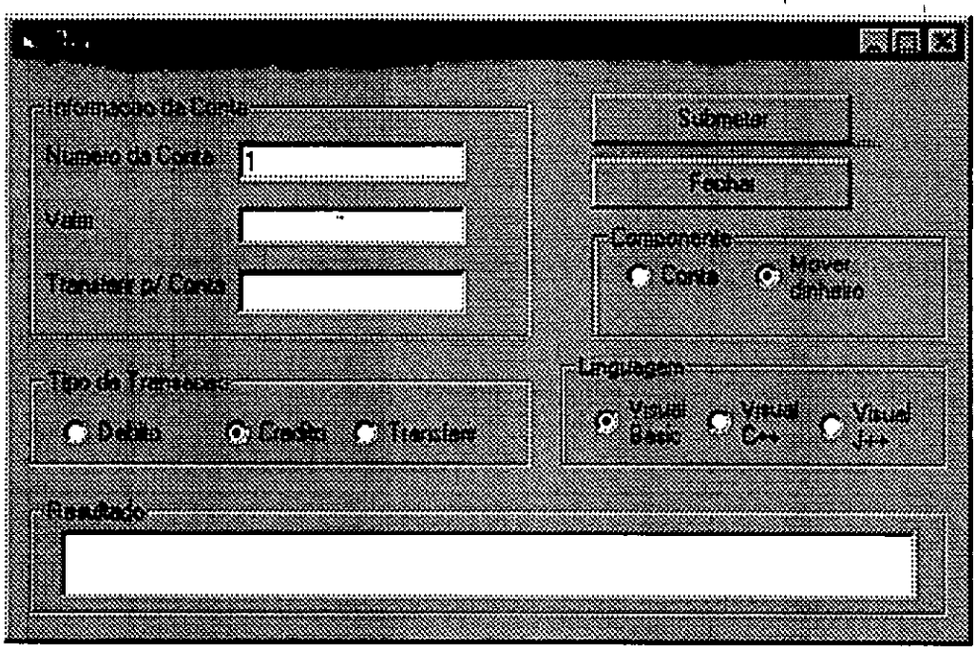
Fig 1. Arquitectura de dois níveis do Snob

Na fig anterior, está ilustrada a arquitectura na qual foi implementada a Aplicação SNOB, estando evidenciados os dois níveis da aplicação: o nível do Cliente e o nível do Servidor, conectados via rede.

ANEXO 2: PAPEL DO MONITOR DE PROCESSAMENTO DE TRANSAÇÕES



ANEXO 3: EXEMPLO DE COMPONENTES



Número da conta- indicação do número da conta.

Valor – montante movimentado na operação.

Transferir p/conta – em caso de transferência, indicar para que conta transferir.

Conta- escolher conta em caso de débito ou crédito no número da conta indicada.

Mover dinheiro- escolher em caso de transferência de contas.

Débito- escolher em caso de depósito.

Crédito- escolher em caso de levantamento.

Transferir- escolher em caso de transferência dum valor duma conta para outra.

Visual Basic, Visual C++, Visual J++- escolher para indicar que linguagem é usada. Importante para o processo de formatação dos parâmetros passados às componentes.

Submeter- pressionar este botão para confirmar a operação pretendida.

Fechar- fechar a janela.

Resultado- campo destinado a apresentação do resultado da operação, por isso não se preenche.

```
' Código referente a form apresentada anteriormente em Visual Basic
' Ao pressionar o botão submeter

Private Sub cmdSubmeter_Click()
  ' Decidir que componente usar
  If Conta = True Then
    ProgID = "Banco.conta"
  Else
    ProgID = "Banco.Movedinheiro"
  End If
  ' Decidir o tipo de linguagem
  If VC = True Then
    ProgID = ProgID + ".VC"
  ElseIf VJ = True Then
    ProgID = ProgID + ".VJ"
  End If
  ' Decidir o tipo de transacao
  Resultado = ""
  lSegundaConta = 0
  If debito = True Then
    tipotrans = 1
    Mult = -1
  ElseIf credito = True Then
    tipotrans = 2
    Mult = 1
  Else
    If ProgID = "Banco.conta" Then
      MsgBox "Erro. Usar a componente Mover Dinheiro para transferir."
      Movedinheiro.SetFocus
      Exit Sub
    End If
```

```
    If conta2 = "" Then
      MsgBox "Erro. A conta de transferencia deve se preenchida."
      conta2.SetFocus
      Exit Sub
    Else
      tipotrans = 3
       lSegundaConta = CLng(conta2)
    End If
  End If
  ' Criacao do objecto proprio Movedinheiro
  On Error GoTo objError
  Dim obj As Object
  Set obj = CreateObject(ProgID)
  On Error GoTo ErrorHandler
  If obj Is Nothing Then
      MsgBox "A criacao do Objecto " + ProgID + "falhou."
    Exit Sub
  End If
' Chamar o objecto
 Res = ""
 If Conta = True Then
      ' chamar a componente post, neste caso implementada como uma class module
      Res = obj.Post(CLng(conta1), CLng(valor * Mult))
 ElseIf Movedinheiro = True Then
      ' chamar a componente perform, neste caso implementada como uma class module
      Res = obj.Perform(CLng(conta1), lSegundaConta, CLng(valor), tipotrans)
 End If
 Resultado = Res
objError:
  MsgBox "Erro " & Err.Number & ": Certificar se o pacote do Banco foi correctamente instalado
no MTS."
```

```
    Err.Clear
    Exit Sub
ErrorHandler:
    MsgBox Err.Number & "(" & Err.Source & ") :" & Err.Description
    Err.Clear
End Sub
```

```
Public Function Post(ByVal contaID As String, ByVal valor As Long) As String
' conta.cls: class module que faz a componente conta

' obtenção do ambiente ADO (Activex Data Object) e a Conexao
    Dim adoConn As New ADODB.Connection
    adoConn.Open strConnect
    On Error GoTo TryAgain
' actualizar o saldo
    Dim strSQL As String
    strSQL = "UPDATE conta SET saldo = saldo + " + Str$(valor) + "
    WHERE cod_conta = " contaID
TryAgain:
    adoConn.Execute strSQL, varRows
    ' se alguma outra coisa acontecer
    On Error GoTo ErrorHandler
    ' tomar o saldo
    strSQL = "SELECT saldo FROM conta WHERE cod_conta = " + contaID
    Dim adoRS As ADODB.Recordset
    Set adoRS = adoConn.Execute(strSQL)
    If adoRS.EOF Then
        Err.Raise Number:=APP_ERROR, Description:="Erro. Conta " + contaID + " nao existe."
    End If
    Dim lngSaldo As Long
    lngSaldo = adoRS.Fields("saldo").Value
```

```
  If (lngSaldo) < 0 Then      ' ver se a conta ficou a descoberto
     Err.Raise Number:=APP_ERROR, Description:="Erro. conta " + contaID + " sera descoberta
" + Str$(lngSaldo) + ". o saldo ainda e " + Str$(lngSaldo - valor) + "."
   Else
     If lngAmount < 0 Then
        strResult = strResult & "Debito da conta " & contaID & ", "
     Else
        strResult = strResult & "Credito a conta " & contaID & ", "
     End If
     strResult = strResult + "o saldo e $" & Str$(lngSaldo) &". (VB)"
   End If
   ' libertar o objecto e a conexão
   Set adoRS = Nothing
   Set adoConn = Nothing
   GetObjectContext.SetComplete        ' fim e sucesso (commit)
   Post = strResult
   Exit Function
ErrorHandler:
  If  Not adoRS Is Nothing Then
     Set adoRS = Nothing
  End If
  If  Not adoConn Is Nothing Then
     Set adoConn = Nothing
  End If
   GetObjectContext.SetAbort          ' fim e fracasso (Rollback)
   Post = ""                  ' indicaçcão de que algum erro ocorreu
   Err.Raise Err.Number, "Banco.conta.Post", Err.Description
End Function
```

```
Public Function Perform (ByVal conta1ID As String, Conta2ID As String, ByVal Valor As Long,
ByVal tipotrans As Long) As String
' movedinheiro.cls: class module que faz a componente mover dinheiro

' criacao do objecto conta usando o contexto
  Dim objConta As Banco.Conta
  Set objConta = GetObjectContext.CreateInstance("Banco.conta")
  If objAccount Is Nothing Then
    Err.Raise ERROR_NUMBER, Description:="Nao pode criar o objecto Conta"
  End If
' chamar a funcao post baseada no tipo da transacao
  Select Case tipotrans
    Case 1
      strResult = objAccount.Post(conta1ID, 0 - Valor)
      If strResult = "" Then
        Err.Raise ERROR_NUMBER, Description:=strResult
      End If
    Case 2
      strResult = objAccount.Post(Conta1ID, valor)
        If strResult = "" Then
          Err.Raise ERROR_NUMBER, Description:=strResult
      End If
    Case 3
      Dim strResult1 As String, strResult2 As.String
      ' faz o credito
      strResult1 = objAccount.Post(Conta2ID, valor)
      If strResult1 = "" Then
        Err.Raise ERROR_NUMBER, Description:=strResult1
      Else
      ' faz o debito
```

```
    strResult2 = objAccount.Post(Conta1ID, 0 - valor)
      If strResult2 = "" Then
        Err.Raise ERROR_NUMBER, Description:=strResult2
      Else
        strResult = strResult1 + "  " + strResult2
      End If
    End If
  Case Else
    Err.Raise ERROR_NUMBER, Description:="Tipo de Transação Invalido"
 End Select
 GetObjectContext.SetComplete          ' fim e sucesso (commit)
 Perform = strResult
 Exit Function
ErrorHandler:
 GetObjectContext.SetAbort             ' fim e fracasso (rollback)
 Perform = ""                  ' indicação de que algum erro ocorreu
 Err.Raise Err.Number, "Banco.Movedinheiro.Perform", Err.Description
End Function
```

Id	Numero d	AssNumEleitor	Codig	PresCodigo	Codig	CartaoInvalido
35	1	45K125252		009		Nao
36	2	4548K2152		009		Nao
37	3	08K1000110		002		Nao
38	4	08K1000110		002		Nao
39	1	08K188032		009		Nao
40	2	08K10001710		003		Nao
41	5	08K1000110		003		Nao
43	10	152252155		009		Nao
44	11	45K125252		002		Nao
45	5	08K1003110		003		Nao
46	5	08K1006110		003		Nao
47	5	08K1006110		003		Nao

AssDuplicaML	AssDuplicaDLista	AssValida
Nao	Sim	Nao
Nao	Nao	Sim
Nao	Sim	Nao
Sim	Sim	Nao
Nao	Nao	Sim
Nao	Nao	Sim
Nao	Sim	Nao
Nao	Nao	Sim
Nao	Sim	Nao
Nao	Nao	Sim
Sim	Sim	Nao
Sim	Sim	Nao

UNIVERSIDADE EDUARDO MONDLANE
FACULDADE DE CIENCIAS
DEPARTAMENTO DE MATEMETICA E INFORMATICA

TRABALHO DE ANALISE DE SISTEMAS_I

Regente

Docente :Esselina Macome

Docente:Carlos Cumbane

Assistente:Miriam

Discente:Alcindo A.Sulinde

TEMA:

SERVICOS ONLINE