Fundamentos da Arquitetura Cliente/Servidor

SUMÁRIO

Fundamentos da Arquitetura Cliente/Servidor

1 - Introdução

Este relatório tem por objetivo mostrar as características técnicas que envolvem a Arquitetura Cliente/Servidor, desde a sua concepção teórica até a implantação da comunicação entre Cliente e Servidor por meio de uma biblioteca de Sockets.

No item 2 será definido a parte teórica da Arquitetura Cliente/Servidor bem como alguns aspectos de definição desta arquitetura, além de mostrar os modelos existentes para implantação desta arquitetura. Em 3 serão mostrados alguns tipos de processamentos existentes, porém enfocando o processamento distribuído. Em 4 serão vistos os conceitos básicos sobre Rede de Comunicação, enfatizando o protocolo TCP/IP e a biblioteca de Sockets. Em 5 são apresentadas as características técnicas de alguns SGBDOOs numa arquitetura Cliente/Servidor. Finalmente, em 6, é feita uma breve conclusão com respeito ao modelo de arquitetura apresentado.

2 - Arquitetura Cliente/Servidor

A arquitetura Cliente/Servidor vem sendo desenvolvida há vários anos, porém em pequenos passos. Primeiro, a realocação de aplicações em Mainframe para as chamadas plataformas abertas rodando, Sistema Operacional UNIX. Depois, com relação a abordagem dos dados, saindo de Sistemas de Arquivos ou Banco de Dados Hierárquicos locados em Mainframes para Sistemas de Banco de Dados Relacional, e posteriormente, a importância da capacidade gráfica dos pacotes de “front-end” existentes, facilitando a interação com o usuário (MCKIE,1997).

Vários aspectos sobre uma definição da arquitetura Cliente/Servidor podem ser descritos.

- ⇒ O termo Cliente/Servidor refere-se ao método de distribuição de aplicações computacionais através de muitas plataformas. Tipicamente essas aplicações estão divididas entre um provedor de acesso e uma central de dados e numerosos clientes contendo uma interface gráfica para usuários para acessar e manipular dados.
- ⇒ Cliente/Servidor geralmente refere-se a um modelo onde dois ou mais computadores interagem de modo que um oferece os serviços aos outros. Este modelo permite aos usuários acessarem informações e serviços de qualquer lugar.
- ⇒ Cliente/Servidor é uma arquitetura computacional que envolve requisições de serviços de clientes para servidores. Uma rede Cliente/Servidor é uma extensão lógica da programação modular.

Portanto, uma definição para a arquitetura Cliente/Servidor seria a existência de uma plataforma base para que as aplicações, onde um ou mais Clientes e um ou mais Servidores, juntamente com o Sistema Operacional e o Sistema Operacional de Rede, executem um processamento distribuído.

Um sistema Cliente/Servidor poderia ser, então, entendido como a interação entre Software e Hardware em diferentes níveis, implicando na composição de diferentes computadores e aplicações.

Para melhor se entender o paradigma Cliente/Servidor é necessário observar que o conceito chave está na ligação lógica e não física. O Cliente e o Servidor podem coexistir ou não na mesma máquina (RENAUD,1994). Porém um ponto importante para uma real abordagem Cliente/Servidor é a necessidade de que a arquitetura definida represente uma computação distribuída (MCKIE,1997).

Algumas das características do Cliente e do Servidor são descritas a seguir: (SALEMI,1993) (HULQUIST,1997)

Cliente

- Cliente, também denominado de "*front-end*" e "*WorkStation*", é um processo que interage com o usuário através de uma interface gráfica ou não, permitindo consultas ou comandos para recuperação de dados e análise e representando o meio pela qual os resultados são apresentados. Além disso, apresenta algumas características distintas:
- É o processo ativo na relação Cliente/Servidor.
- Inicia e termina as conversações com os Servidores, solicitando serviços distribuídos.
- Não se comunica com outros Clientes.
- Torna a rede transparente ao usuário.

Servidor

- Também denominado Servidor ou "*back-end*", fornece um determinado serviço que fica disponível para todo Cliente que o necessita. A natureza e escopo do serviço são definidos pelo objetivo da aplicação Cliente/Servidor. Além disso, ele apresenta ainda algumas propriedades distintas:
- É o processo reativo na relação Cliente/Servidor.
- Possui uma execução contínua.
- Recebe e responde às solicitações dos Clientes.
- Não se comunica com outros Servidores enquanto estiver fazendo o papel de Servidor.
- Presta serviços distribuídos.
- Atende a diversos Clientes simultaneamente.

Outras características dos processos Cliente e Servidor podem ser encontradas em (KALAKOTA,1997).

Alguns tipos de serviços que um Servidor pode proporcionar são:

- Servidor de Arquivos
- Servidor de Impressora
- Servidor de Banco de Dados
- Servidor de Redes
- Servidor de Telex

- Servidor de Fax
- Servidor X-Windows
- Servidor de Processamento e Imagens
- Servidor de Comunicação e etc.

O estilo de interação entre o usuário e o Cliente não precisa, necessariamente, ser feita por poderosas interfaces gráficas. Porém, já que o poder de processamento local do Cliente está disponível, pode-se retirar todo seu proveito, através de interfaces gráficas - GUI (Graphical User Interface), para melhor rendimento do usuário no seu trabalho.

Dentre as muitas vantagens da arquitetura Cliente/Servidor, pode-se citar: (SALEMI,1993)

2.1 - Vantagens

- **Confiabilidade**
  Se uma máquina apresenta algum problema, ainda que seja um dos Servidores, parte do Sistema continua ativo.

- **Matriz de Computadores agregando capacidade de processamento**
  A arquitetura Cliente / Servidor provê meios para que as tarefas sejam feitas sem a monopolização dos recursos. Usuários finais podem trabalhar localmente.

- **O Sistema cresce facilmente**
  Torna-se fácil modernizar o Sistema quando necessário.

- **O Cliente e o Servidor possuem ambientes operacionais individuais / Sistemas Abertos**
  Pode-se misturar várias plataformas para melhor atender às necessidades individuais de diversos setores e usuários.

Além destas vantagens, pode-se encontrar dentro de uma arquitetura Cliente/Servidor a interoperabilidade das estações Clientes e Servidoras entre as redes de computadores, a escalabilidade da arquitetura visando o crescimento e a redução dos elementos constituintes, a adaptabilidade de novas tecnologias desenvolvidas, a performance do hardware envolvido na arquitetura, a portabilidade entre as diversas estações que compõem a arquitetura e a segurança dos dados e processos (MCKIE,1997).

Embora o avanço da arquitetura Cliente/Servidor tenha trazido uma variada gama de facilidades para o desenvolvimento de aplicações distribuídas, também possui algumas desvantagens:

2.2 - Desvantagens

- **Manutenção**
  As diversas partes envolvidas nem sempre funcionam bem juntas. Quando algum erro ocorre, existe uma extensa lista de itens a serem investigados.

- **Ferramentas**
  A escassez de ferramentas de suporte, não raras vezes obriga o desenvolvimento de ferramentas próprias. Em função do grande poderio das novas linguagens de programação, esta dificuldade está se tornando cada vez menor.

- **Treinamento**
  A diferença entre a filosofia de desenvolvimento de software para o microcomputador de um fabricante para o outro, não é como a de uma linguagem de programação para outra. Um treinamento mais efetivo torna-se necessário.

- **Gerenciamento**
  Aumento da complexidade do ambiente e a escassez de ferramentas de auxílio tornam difícil o gerenciamento da rede.

2.3 - Modelos da Arquitetura Cliente / Servidor

Existem cinco tipos de modelos para a implantação da arquitetura Cliente/Servidor em processamentos distribuídos: (SALEMI,1993)

2.3.1 - Arquitetura C/S Simples

A primeira abordagem para um sistema distribuído é a arquitetura Cliente/Servidor Simples. Nesta arquitetura, o Servidor não pode iniciar nada. O Servidor somente executa as requisições do Cliente. Existe uma clara função de diferenciação: Pode-se estabelecer que o Cliente é o mestre e o Servidor é o escravo, como mostra a figura 1.

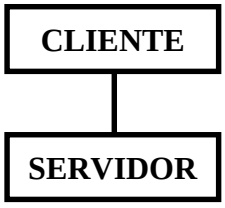


Figura 1 - Arquitetura Cliente/Servidor Simples

2.3.2 - Arquitetura C/S em Dois Níveis

A configuração usual Cliente/Servidor encontrada na maioria das empresas, é aquela em que existem vários Clientes requisitando serviços a um único Servidor. Esta arquitetura se caracteriza como sendo c*entrada no Servidor* (figura 2a). Porém na visão do usuário, ele imagina que existem vários Servidores conectados a somente um Cliente, ou seja, c*entrado no Cliente* (figura 2b). Entretanto, com as várias ligações de comunicação possíveis, existe na realidade uma mistura de Clientes e Servidores (figura 2c).

Figura 2 - (a) Arquitetura C/S em Dois Níveis - Centrado no Servidor

Figura 2 - (b) Arquitetura C/S em Dois Níveis - Centrado no Cliente

Figura 2 - (c) Arquitetura C/S em Dois Níveis - Comunicação Mista

2.3.3 - Arquitetura C/S Multinível

Nesta arquitetura (figura 3), permite-se que uma aplicação possa assumir tanto o perfil do Cliente como o do Servidor, em vários graus. Em outras palavras, uma aplicação em alguma plataforma será um Servidor para alguns Clientes e, concorrentemente, um Cliente para alguns Servidores.

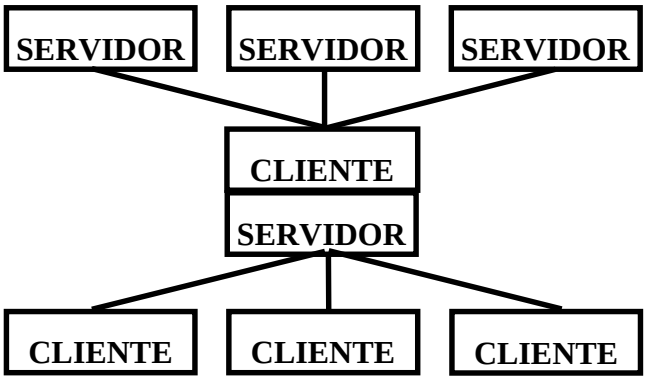


Figura 3 - Arquitetura C/S Multinível

2.3.4 - Arquitetura C/S Par-Par

Esta arquitetura pode ser vista como o caso mais geral da arquitetura Cliente/Servidor, ilustrado na figura 4. Cada um dos nodos desta arquitetura assume tanto o papel de Cliente quanto de Servidor. Na verdade, torna-se pouco funcional lidar com quem é o Cliente ou o Servidor. É o caso onde o processo interage com outros processos em uma base pareada, não existindo nenhum Mestre ou Escravo: qualquer estação de trabalho pode iniciar um processamento, caso possua uma interface de comunicação entre o usuário e o processo Cliente.

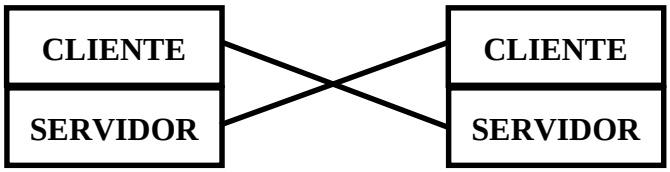


Figura 4 - Arquitetura C/S Par-Par

3 - Alguns Tipos de Processos numa Arquitetura Cliente/Servidor

A arquitetura Cliente/Servidor divide uma aplicação em processos, que são executados em diferentes máquinas conectadas à uma Rede de Computadores, formando um único sistema. O paradigma da tecnologia Cliente/Servidor, serve como um modelo, entre outros (ANDREWS,1991), para interação entre processos de softwares em execução concorrente.

Os processos ou tarefas, a serem executadas são divididas entre o Servidor e o Cliente, dependendo da aplicação envolvida e das restrições impostas pelo Sistema Operacional de Rede (SOR). Quanto mais avançado for o Sistema Operacional de Rede, menor será a aplicação em si, uma vez que a implementação do código para acessar a rede já se encontra definido no SOR.

Atualmente dois tipos de processamentos estão sendo divulgados, Processamento Distribuído e Processamento Cooperativo. A característica de cada um destes é descrita a seguir (KALOKATA,1997), e neste trabalho o enfoque será destinado ao processamento distribuído.

Processamento Distribuído
A distribuição de aplicações e tarefas se faz através de múltiplas plataformas de processamento. O processamento distribuído implica que essas aplicações/tarefas irão ocorrer em mais de um processo, na ordem de uma transação a ser concluída. Em outras palavras, o processamento é distribuído através de duas ou mais máquinas e os processos, na maioria, não rodam ao mesmo tempo, por exemplo, cada processador realiza parte de uma aplicação em uma seqüência. Geralmente, o dado usado em um ambiente de processamento distribuído também é distribuído através de plataformas.

Processamento Cooperativo
A cooperação requer dois ou mais processadores distintos para completar uma simples transação. O processamento cooperativo é relatado para ambos os processos cliente-servidor. É uma forma de computação distribuída onde dois

ou mais processadores distintos são requeridos para completar uma simples transação de negócios. Normalmente esses programas interagem e executam concorrentemente como diferentes processos. Os processos cooperativos também são considerados como um estilo de Cliente/Servidor através da arquitetura de mensagens, que devem obedecer a um determinado padrão.

No contexto do presente trabalho pretende-se utilizar as características do processamento distribuído. Este tipo de processamento apresenta duas configurações para uma arquitetura Cliente/Servidor. A primeira, que é representada por três camadas, é responsável pela visualização da interação entre os aplicativos e o hardware, como pode ser visto na figura 5.

A segunda representação, também fornecida em três camadas, mostra como é tratada a divisão da funcionalidade de uma aplicação, segundo as configurações do *Gartner Group*, como pode ser visto na figura 6.

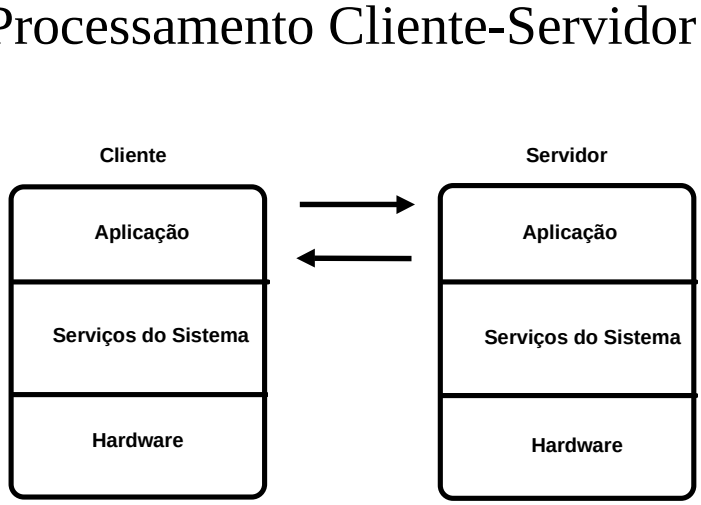
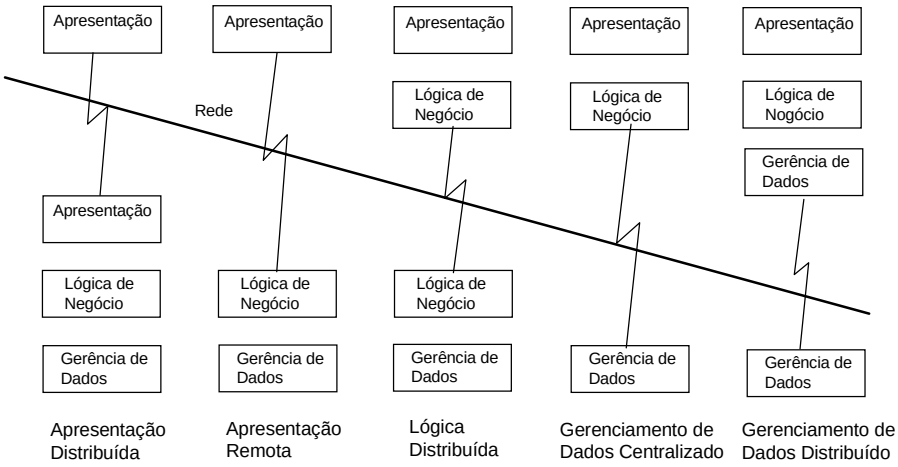


Figura 5 - Sistema Cliente/Servidor

3.1 - Processamento Distribuído

O processamento distribuído, também denominado de processamento concorrente utiliza-se do mecanismo de passagem de mensagens para a comunicação entre processos, que podem ser de quatro tipos básicos; Filtro, Peer (não hierárquico), Cliente e Servidor (RENAUD,1994), como mostrado na figura 7 a seguir.

Modelo de Distribuição de Processos



Fonte: Gartner Group

Figura 6 - Arquitetura da Aplicação Cliente/Servidor

Processamento Distribuído

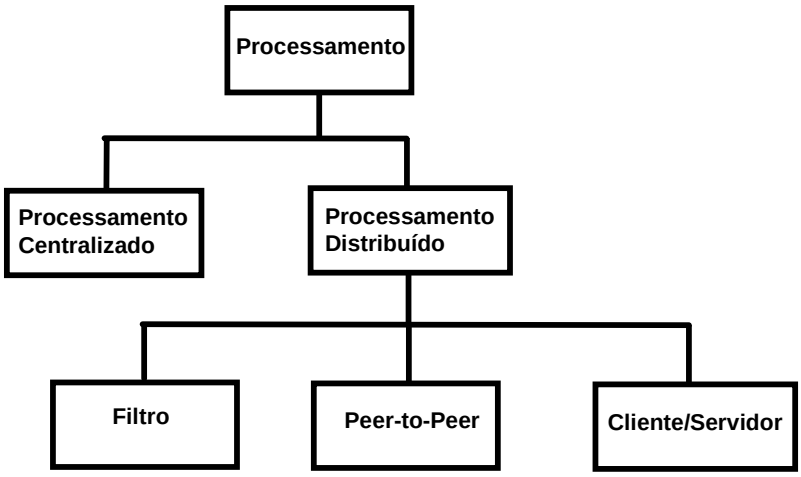
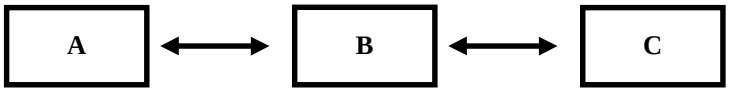


Figura 7 - Processamento Distribuído

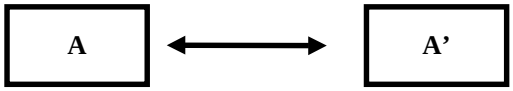
Características do **Processo de Filtro**:

- Determina uma conversão na mensagem de comunicação entre o usuário e o *host*. Ex.: Ligação de um desktop com um mainframe através de um emulador de terminal.



Características do **Processo Peer-to-Peer** (não hierárquicos):

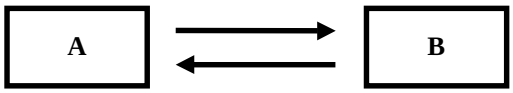
- São processos “clones” rodando em todas as máquinas e prestando serviços uns para os outros.
- Não existem processos servidores, estabelecendo um Servidor dedicado.
- Cada processo pode ser Cliente e Servidor para outros processos.



Durante a execução das consultas, os processos Cliente e Servidor são confundidos com os processos Peer-to-Peer, porque este processo foi desenvolvido com base na *LU6.2* (Logical Unit versão 6.2) do *SNA* (*Systems Network Architecture*) da *IBM* (RENAUD,1994).

Características do **Processo Cliente / Servidor**:

- Existem processos distintos: o processo cliente é diferente do processo servidor.
- Os processos servidores tornam a estação Servidora dedicada ao seu trabalho.
- Processos clientes são sempre clientes.
- Processos servidores podem se tornar processos clientes de outros Servidores.



O modelo que utiliza processos Cliente/Servidor depende do cliente para inicializar a comunicação. Caso o Servidor comece a comunicação, o processo instalado fica sendo o Peer-to-Peer

A característica básica da arquitetura Cliente/Servidor é a que processos Clientes enviam pedidos a um processo Servidor, que retorna o resultado para o Cliente. O processo Cliente fica então liberado da ação do processamento da transação podendo realizar outros trabalhos.

3.2 - Camadas da Arquitetura Cliente / Servidor

A arquitetura Cliente/Servidor é dividida em três camadas básicas, como ilustra a figura 8. A camada de Aplicação consiste dos processos da aplicação, entre eles, os processos Cliente e Servidor. A camada de Serviços de Sistemas compreende o Sistema Operacional (SO) e o Sistema Operacional de Rede (SOR), destinando-se ao controle do hardware. Por último a camada de hardware, onde estão localizados os periféricos ligados aos Clientes e Servidores.

Figura 8 - Três Camadas da Arquitetura Cliente/Servidor

A tecnologia Cliente/Servidor pode existir tanto no nível da camada de Aplicação, quanto no da camada de Serviços do Sistema. A coexistência do paradigma nestas camadas surge em função da hierarquia das atuações no sistema. Caso o "usuário" seja externo ao sistema, então os processos Cliente e Servidor compõem a camada da Aplicação, enquanto que, se o "usuário" for um programa de aplicação o Cliente é um processo redirecionador, e o Servidor será um processo respondedor da camada de Serviços do Sistemas.

A utilização de sistemas Cliente/Servidor pela camada de aplicação utiliza Serviços do Sistema não Cliente/Servidor (figura 8), entretanto, sistemas não Cliente/Servidor, ao nível da aplicação utilizam Serviços do Sistema Cliente/Servidor (RENAUD,1994).

Para sistemas Cliente/Servidor na camada de aplicação, a camada Serviços do Sistema oferece somente um mecanismo de IPC (InterProcess Comunication) para troca de mensagens. Por outro lado, a camada Serviços do Sistema configurada como Cliente/Servidor, é responsável por gerenciar o redirecionamento das solicitações de gravação/leitura, por exemplo. É importante notar que a diferença entre os sistemas Cliente/Servidor nas camadas de Aplicação e Serviços do Sistema, é o equilíbrio entre a quantidade de processamento tanto no lado do Cliente quanto no lado Servidor.

Existem vários sistemas que podem ser baseados na estrutura Cliente/Servidor. O uso mais freqüente são as aplicações de Banco de Dados usando processos *SQL* (*Structured Query Language*) de front-end, para acessar remotamente, as bases de dados.

A figura 9 mostra uma estrutura baseada num Servidor de Arquivos. Esta estrutura ocasiona um maior fluxo de informações na rede, uma vez que todo o arquivo será transferido para o Cliente para então ser trabalhado.

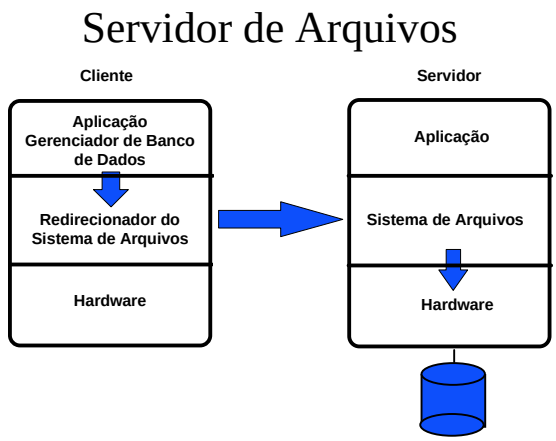


Figura 9 - Arquitetura Cliente/Servidor como Servidor de Arquivo

Neste tipo de estrutura, a camada de Aplicação passa a ser o Cliente do Sistema. Com isto, a camada de Serviço do Sistema é utilizada simplesmente como um redirecionador para acesso à base de dados (RENAUD,1994). Neste tipo de configuração pode-se chamar este Sistema como falso Sistema Cliente/Servidor, por não haver um equilíbrio de processamento entre os dois lados Cliente e Servidor. O lado Servidor somente terá o trabalho de executar as rotinas comuns de I/O, não caracterizando assim, como um processamento intrínseco à aplicação.

A figura 10 demonstra outra possibilidade de se estruturar um Sistema baseado na arquitetura Cliente/Servidor, que consiste na utilização de um Servidor de Banco de Dados dedicado, podendo coexistir normalmente com o Servidor de arquivos. Neste momento, com um Servidor de Banco de Dados exclusivo, o fluxo de informações trafegadas na rede diminui, já que, somente a resposta da consulta é retornada ao Cliente, ao invés de transferir todo o arquivo como era feito anteriormente.

O Cliente, neste tipo de estrutura é o usuário, passando a ter uma visão da aplicação como se as partes, Cliente e Servidor, fossem algo único. A figura 11 exemplifica esta visão.

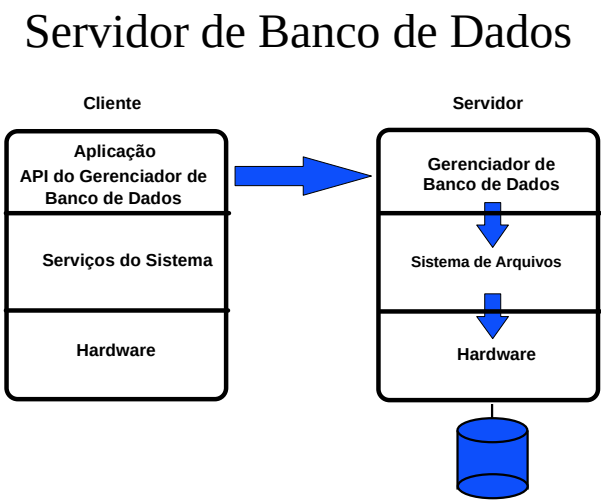


Figura 10 - Arquitetura Cliente / Servidor como Servidor de Banco de Dados

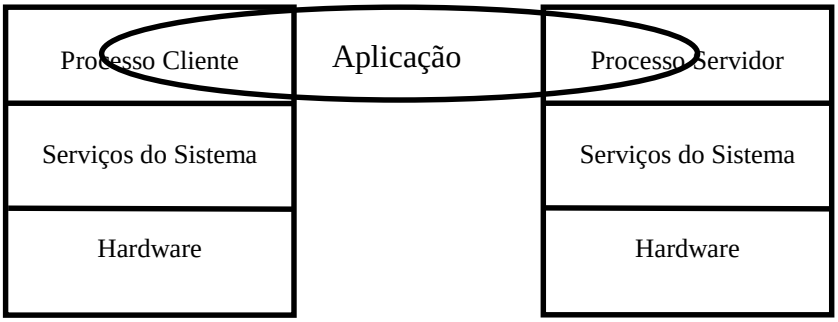


Figura 11 - Integração entre os processos clientes e servidores

Este tipo de estrutura favorece o aumento da performance da rede de comunicação, possibilitando assim, um maior número de ligações simultâneas de diversos Clientes com diversos Servidores (RENAUD,1994).

Embora a arquitetura Cliente/Servidor possa parecer uma nova versão do modelo de arquivos compartilhados, através da Rede Local baseada em microcomputadores, suas vantagens são inúmeras. Fundamentalmente, ambos os modelos provêem capacidade de processamento distribuído e permitem o compartilhamento de informação. Entretanto, no Modelo de Arquivos compartilhados baseado em Servidor de Arquivos, o Cliente além de executar a aplicação, executa também o motor da Base de Dados, que por sua vez acessa essas Bases de Dados remotamente como se fossem locais. O Servidor de Arquivos envia arquivos inteiros através da rede para o Cliente processar localmente, ocasionando congestionamento na rede, enquanto que, no modelo Cliente/Servidor, o Cliente executa parte da aplicação, sendo deixado para o Servidor a tarefa da administração da Base de Dados, comumente exercida por algum SGBD. O Servidor envia para o Cliente, através da Rede, apenas o bloco de dados apropriado, como resultado da consulta efetuada pela aplicação.

Desta forma, na arquitetura Cliente/Servidor, cada Servidor pode suportar um número maior de usuários, uma vez que é o Cliente que gerencia a aplicação e a interface com o usuário. Além do mais, com a crescente conectividade entre máquinas e sistemas operacionais, pode-se escolher para Cliente o ambiente de software e hardware que melhor se adeque às necessidades de cada aplicação do usuário, sem ter que se preocupar com o Servidor.

A melhor divisão de tarefas entre o Cliente e o Servidor depende de cada aplicação em si. Se o Servidor for apenas um Sistema Gerenciador de Banco de Dados, deixando para o Cliente o resto do processamento, ou se algumas tarefas como controle de acesso, análise dos dados, e validação dos comandos é executada pelo Servidor, esta é uma decisão do construtor do Sistema em função das características do negócio do usuário. Estas diferenças serão mostradas no tópico a seguir.

3.2.1 - Sistema de Três Camadas para a Aplicação

Os Sistemas Cliente/Servidor têm sido utilizados basicamente nos Sistemas de Banco de Dados Distribuídos, Processamento de Transações e nos Sistemas de Suporte a Decisão.

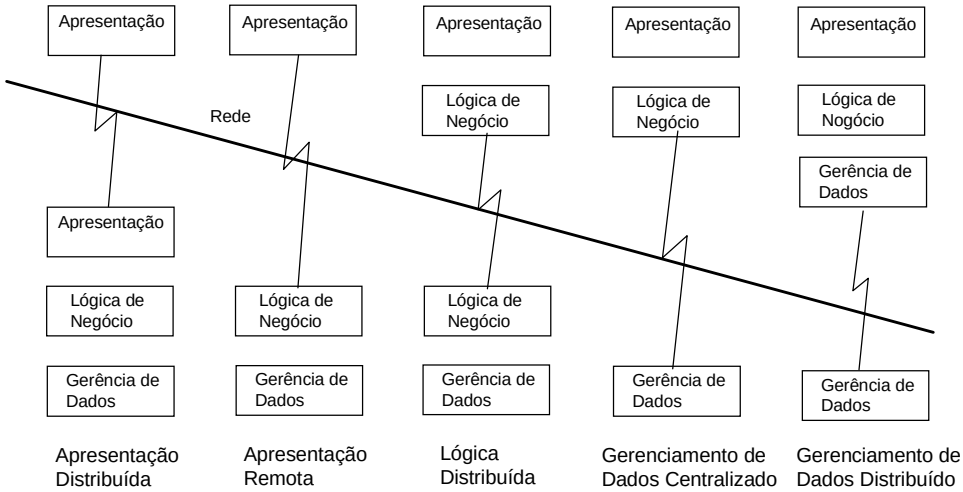
A maioria das aplicações tidas como críticas têm permanecido num mainframe. Pode-se definir como críticas aquelas aplicações cujos resultados são utilizados para decisões estratégicas e que, portanto, variam de empresa para empresa, dependendo do seu negócio. Quase que universalmente, os sistemas de finanças são considerados críticos para todas as empresas. Por outro lado, os sistemas de processamento de transações são críticos para as companhias de aviação.

As aplicações de processamento de transações, apesar de serem tipicamente consideradas como críticas, têm sido freqüentemente migradas para arquitetura Cliente/Servidor, através do processo de "*Downsizing*", como por exemplo os sistemas de entrada de pedidos, pontos de vendas e sistemas de reservas. As transações ocorrem nos Clientes e são formatadas e enviadas ao Servidor para armazenamento.

Seria razoável imaginar que as primeiras aplicações a serem desenvolvidas em Sistemas Cliente/Servidor seriam relativamente simples, garantidas e não críticas. Porém, as empresas que se decidiram pelo modelo Cliente/Servidor testaram-no com aplicações de processamento de transações, e na maioria dos casos ficaram satisfeitas com os resultados.

Como pode ser visto pela figura 12, o Gartner Group apresenta cinco maneiras de se implementar a arquitetura Cliente/Servidor. Cada camada é dividida entre a parte lógica e a parte de gestão. A primeira representação refere-se a distribuição da camada de Apresentação.

Modelo de Distribuição de Processos

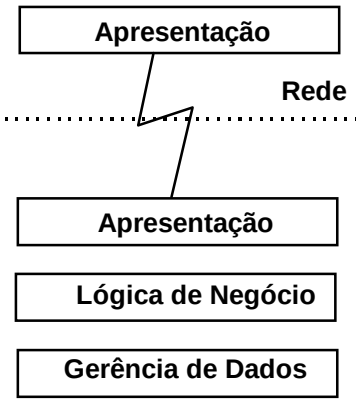


Fonte: Gartner Group

Figura 12- Arquitetura da Aplicação Cliente/Servidor

3.2.1.1 - Apresentação Distribuída

Esta Arquitetura emula uma arquitetura Cliente/Servidor. Toda a gerência da Apresentação é efetuada no Servidor, enquanto que no Cliente, somente a lógica da impressão dos caracteres no monitor é executada. O Cliente não possui qualquer tipo de "inteligência". Existe uma técnica de tratamento cooperativo, denominada de *Revamping*, que é utilizado pela Apresentação Distribuída.



Apresentação Distribuída

Fonte: Gartner Group

O *Revamping* é uma técnica de tratamento cooperativo que está dividida em três tipos: simples, evoluído e o modificado (LEFEBVRE,1994).

Revamping Simples

Consiste em habilitar as telas de modo gráfico das aplicações centralizadas com diálogos em modo gráfico. Cada mapa de tela, gerado corresponde a uma janela de interface gráfica.

Revamping Evoluído

Este tipo não emite simplesmente uma réplica gráfica da janela a ser criada de uma aplicação não gráfica. Constrói uma aplicação inteiramente nova, inteiramente diferente da aplicação original. Assim, um diálogo na nova aplicação pode representar um empilhamento de várias janelas da aplicação centralizada.

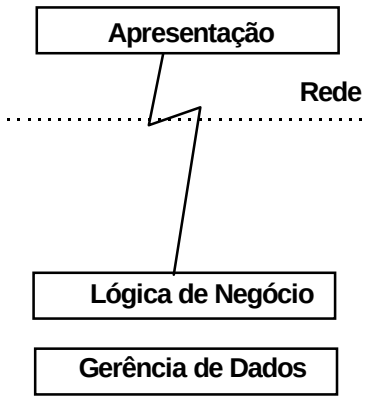
Revamping Modificado

Este terceiro tipo se deriva do segundo, porém com o fato interessante de se poder manipular a aplicação centralizada instalando sistemas gráficos, visando retirar o melhor de sua performance.

3.2.1.2 - Apresentação Remota

A segunda arquitetura é definida como Apresentação Remota, e possui a implementação tanto do módulo de gestão, quanto o de lógica na estação Cliente, ficando responsável, entre outras tarefas, pela manipulação das telas e pelas críticas dos dados que estão sendo inseridos.

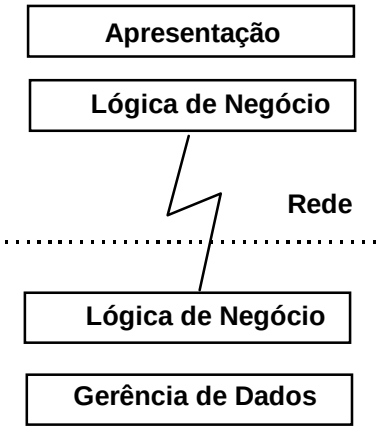
Uma ponto a ser observado, é quando se utiliza um Servidor *X-Windows* : embora o usuário se localize no Terminal *X-Windows*, ele realmente está se utilizando do Servidor *X-Windows*. O terminal real Cliente é onde o usuário está conectado. Esta característica é muito encontrada em ambientes UNIX. O Servidor X-Windows fica responsável pela manipulação das telas e interação com o usuário.



Apresentação Remota

Fonte: Gartner Group

3.2.1.3 - Lógica Distribuída



Lógica Distribuída

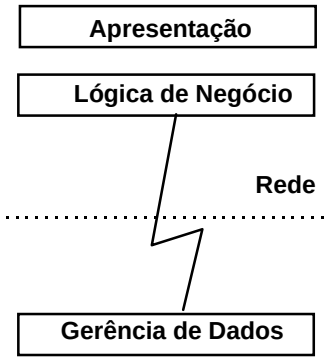
Fonte: Gartner Group

Outra representação é a arquitetura da Lógica Distribuída. Ela representa o melhor modelo de implementação para arquitetura Cliente/Servidor. Nela, o balanço entre os processamentos clientes e servidores são bem determinados. Neste tipo de arquitetura, a implantação de procedimentos armazenados *(Stored Procedures)* facilitam a performance na rede de comunicação, uma vez que somente o nome do procedimento armazenado no Servidor é transmitido pela rede. Um ponto muito positivo para este tipo de estrutura se refere a pré-compilação necessária para os procedimentos armazenados, aumentando consideravelmente o tempo de resposta. Quando os procedimentos

armazenados não são utilizados, é necessário que sejam feitas compilações das sentenças das consultas antes de executá-las, evidenciando uma perda desnecessária no tempo de processamento para enviar a resposta ao Cliente. Porém, um ponto negativo para este tipo de arquitetura, é o fato de se ter que criar antecipadamente as consultas a serem utilizadas pelo sistema. Ao contrário, se o sistema utilizar interações com o usuário este tipo de arquitetura não é o mais recomendado, pois não se pode prever todos os tipos de consultas que os usuários possam a vir a fazer.

3.2.1.4 - Gerenciamento de Dados Centralizado

Neste tipo de Arquitetura, toda a parte de gestão e lógica da Aplicação fica destinada ao Cliente, enquanto que somente o responsável por prover o armazenamento e a persistência dos dados permanece no Servidor, como por exemplo no caso de um SGBD, responsável direto pela gestão e lógica dos dados armazenados. Este tipo de arquitetura é a mais difundida, não criando, teoricamente, qualquer tipo de empecilho na hora da migração de plataformas mainframes para plataformas *desktop* do tipo PC (*Personal Computer*).

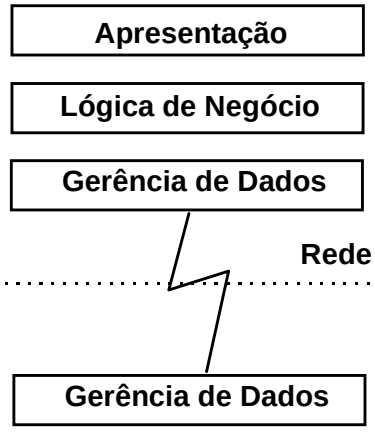


Gerenciamento de Dados Centralizados

Fonte: Gartner Group

3.2.1.5 - Gerenciamento de Dados Distribuídos

Ao contrário do Gerenciamento de Dados Centralizados, o Gerenciamento de Dados Distribuídos permite a replicação e divisão dos dados por diversos sítios (*sites*). Os sistemas distribuídos não deixam de ser microsistemas centralizados, necessitando porém de estruturas como metadados para determinar a localização dos dados e suas replicações. Já existem no mercado sistemas que se instalam entre a parte lógica da aplicação e a gerência de dados, com a função de controlar a localização real dos dados.



Gerenciamento de Dados Distribuídos

Fonte: Gartner Group

Outro ponto muito importante a ser verificado é o gerenciamento da concorrência entre processos, quase sempre a cargo dos Sistemas Operacionais das plataformas, tais como o Windows NT e o UNIX (CUSTER,1993) (PHAM,1996), (TANENBAUM,1995). O trabalho de gerenciamento para o compartilhamento dos dados se deve exclusivamente aos SGBDs.

Os tipos de arquiteturas apresentadas não são mutuamente exclusivas, mas sim complementares. É possível fazer vários tipos de associações entre Servidores e Clientes, dentro de uma rede de computadores.

A tabela 1 mostra um resumo das características de sistemas Cliente/Servidor.

Atributo	Cliente	Servidor
Modo	Ativo	Reativo
Execução	Início e final fixos	Roda o tempo todo
Finalidade Principal	1. Manipulação de tela / janela 2. Interpretação de menu / comando 3. Entrada de mouse / teclado 4. Entrada de dados e validação 5. Processamento de ajuda 6. Recuperação de erro	1. Oferecer serviços funcionais 2. Compartilhamento de dados na aplicação 3. Compartilhamento de dispositivos
Transparência	Oculta rede e servidores	Oculta detalhes de implementação dos serviços
Inclui	Comunicação com diferentes servidores	Comunicação com diferentes clientes
Exclui	Comunicação Cliente - Cliente	Comunicação Servidor - Servidor

Tabela 1 - Principais tópicos de uma Arquitetura Cliente/Servidor

4 - Redes de Computadores

Normalmente várias perguntas são feitas para se tentar entender como clientes e servidores se comunicam, que tipo de protocolo é utilizado, e como um cliente encontra o servidor desejado dentro de uma rede de computadores. Embora estas perguntas possam parecer muito difíceis, na realidade elas são relativamente fáceis.

Existem vários conceitos e aspectos comuns às comunicações Cliente/Servidor necessários ao entendimento do funcionamento da comunicação. Nesta seção será dada uma visão geral sobre os conceitos básicos de protocolos, passagem de mensagem, os aspectos de conexão e sincronismo de processo.

4.1 - Protocolos

Há duas décadas, diversos esforços vêm sendo traçados para se estabelecer um padrão único para redes de computadores. No entanto, foram definidos não apenas um, mas vários modelos de referência.

4.1.1 - O Modelo de Referência *OSI/ISO*

Em março de 1977, foi constituído pela *ISO* um grupo de trabalho para estudar a padronização da interconexão de sistemas de computação. Foi definida, inicialmente, uma arquitetura geral, denominada *MODELO DE REFERÊNCIA OSI* (*Open System Interconection*), para servir de base para a padronização da conexão de sistemas abertos (TAROUCO,1986) (TANENBAUM,1996)

O modelo *OSI* possui sete camadas, como apresentado na figura 13. Cada camada especifica um grupo específico de tarefas de comunicação. A norma descreve o escopo funcional de cada camada e os requisitos para a interface com as camadas adjacentes (serviços).

Camada de Aplicação	- Seleção de serviços apropriados à aplicação.
Camada de Apresentação	- Formatação/Reformatação de dados. Ex.: Criptografia.
Camada de Sessão	- Interface do usuário para o estabelecimento de conexões.
Camada de Transporte	- Controle de intercâmbio de mensagens entre usuários.
Camada de Rede	- Controle de intercâmbio de mensagens na rede.
Camada de Enlace	- Transmissão livre de erros.
Camada de Física	- Interface elétrica para transmissão bit a bit.

Figura 13 - Camadas do Modelo de Referência *OSI*

A figura 14 mostra que quando a mensagem passa da camada n +1 para a camada n são acrescidos outros dados relevantes à camada n . Estes dados são retirados quando a mensagem chega na camada de mesmo nível na estação destino. Estes acréscimos podem ser informações tais como: tipo da mensagem, endereços, tamanho da mensagem , código de detecção de erro, etc.

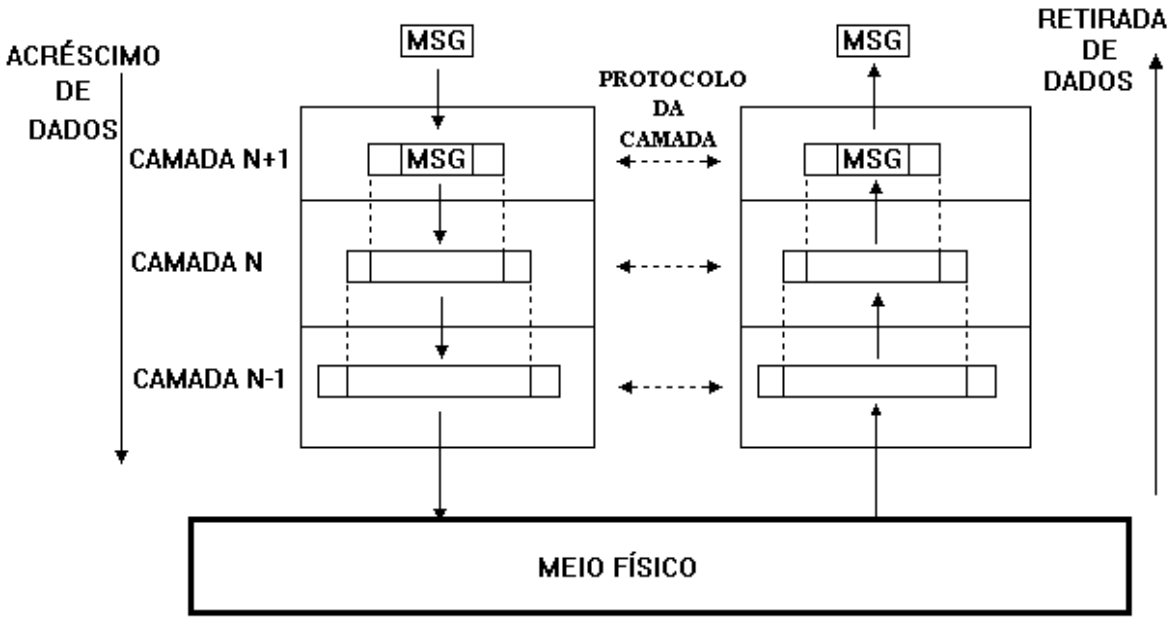


Figura 14 : Arquitetura de protocolos em camada

O modelo OSI/ISO especifica todas as primitivas de comunicação para que haja troca de mensagens entre as camadas (AMARAL,1993) (COMER,1993) (DUMAS,1995) (RENAUD,1994) (TANENBAUM,1996). Cada camada adiciona um cabeçalho para que haja uma identificação na máquina destino.

A figura 15 mostra os principais protocolos usados nas comunicações Cliente / Servidor baseados na Microsoft, Internet e IBM.

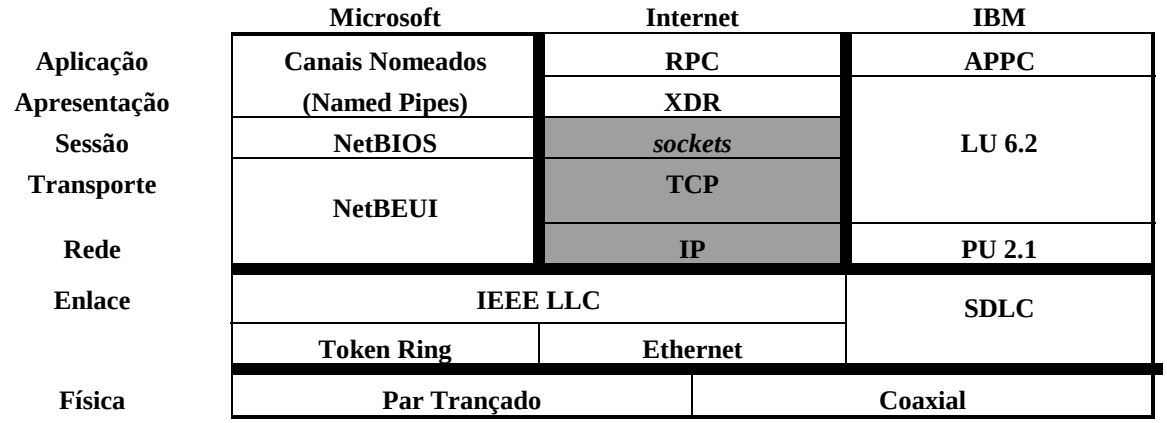


Figura 15 - Comparação entre o modelo OSI / ISO e outros Protocolos

Tendo em vista os objetivos do trabalho pretendido pela presente tese, somente o protocolo de comunicação TCP/IP será descrito, em função da sua interação com *sockets*, já que este será o meio de comunicação adotado na implementação do protótipo aqui proposto. O leitor poderá referenciar-se a (COMER,1993) (DAVIS,1994) TANEMBAUM96] para obter maiores detalhes sobre os demais protocolos.

4.1.2 - TCP/IP

O TCP/IP distingue-se dos demais protocolos pelo seu endereçamento universal. Cada máquina em uma rede possui um endereço que a identifica. A camada TCP é orientada à conexão, enquanto a camada IP trabalha sem conexão (COMER,1993) (DAVIS,1994) (DUMAS,1995). Cabe à camada IP o trabalho de distribuir os datagramas entre as máquinas de uma rede. Para fazer este serviço, ele possui um único endereço de 32 bits que contém informações suficientes para identificar univocamente uma rede e um determinado computador.

Este endereço é comumente escrito em decimal de quatro bytes separados por um ponto. Por exemplo:

11011111, 00000001, 00000010, 00000101, representaria o endereço 223.1.2.5.

Existem 5 classes onde estes endereços são divididos. A figura 16 mostra estas classificações.(QUINN,1996) (RENAUD,1994)

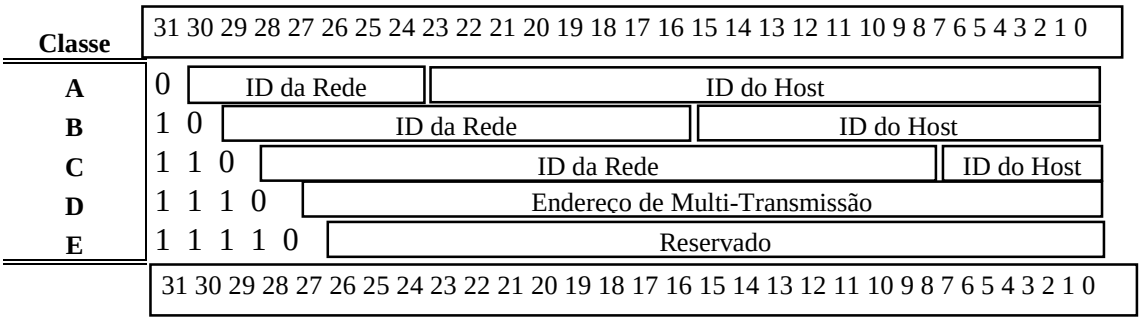


Figura 16 - Classes do protocolo TCP/IP

O número de redes e *hosts* que podem ser endereçados pelas Classes *A, B, C, D* e *E*, bem como o escopo de seus endereços IP são definidos na tabela 2.

Por exemplo, é possível endereçar até 65.136 *hosts* para cada rede da classe *B*. Portanto, 65.136 x 16.382 = 1.067.057.952. Isto quer dizer que mais de 1 milhão de *hosts* podem ser endereçáveis pela classe *B*, já que certos endereços IP não podem ser utilizados por serem destinados a trabalhos específicos, como o endereçamento de *Gateways* e de *Broadcasting*.

Classes	Num. de Redes	Endereço de Rede	Num. de Hosts	Endereço de Hosts
A	*128*	*0.0.0.0 - 127.0.0.0*	*+ 16M*	*0.0.0 - 255.255.255*
B	*16.284*	*128.0.0.0 - 191.255.0.0*	*65.136*	*0.0 - 255.255*
C	*+ 2M*	*192.0.0.0 - 223.255.255.255*	*256*	*0 - 255*
D		*224.0.0.0 - 239.255.255.255*		
E		*240.0.0.0 - 255.255.255.255*		

Tabela 2 - Limite do número de redes e hosts em cada endereço IP

Com isto, quando se tem um endereço de rede na classe *B* e deseja-se aumentar o número de redes possíveis, em detrimento da capacidade de máquinas que podem ser instaladas, ou vice-versa, utiliza-se a técnica da máscara de rede *(NetMask)*. Esta técnica permite a criação de subredes através da modificação local da linha divisória que separa os bits de identificação das redes e dos *hosts*. Cada subrede também é representada pela notação de ponto decimal. Os bits da *NetMask* definidos como “um” identificam a porção da rede, enquanto os definidos por “zero”, representam os *hosts*.

Por exemplo, caso uma empresa possua o endereço IP 166.78.4.139 da classe *B*, isto indica que o computador reside na rede 166.78 e que tem uma identificação de *host* 4.139. Porém, caso os administradores da rede decidam utilizar a *NetMask* 255.255.255.0 para aumentar o número de redes internas à empresa, é necessário que se faça uma combinação AND, do endereço IP original com a *NetMask*, alterando a identificação de rede para 166.78.4 e a identificação de host para 139. Pode-se dizer com isso, que o computador é o *host* 139 na subrede 166.78.4.0 (endereço de *gateway*).

4.2 - Aspectos de Conexão

Existem dois tipos de conexões usados para a comunicação. O primeiro é chamado *sem conexão*, onde cada mensagem encontra seu próprio caminho até o seu destino (COMER,1993) (DUMAS,1995) (QUINN,1996). Cada mensagem é independente de todas as outras mensagens, podendo atingir rotas diferentes até o seu destino. A Internet utiliza este tipo de conexão.

Logo, cada mensagem deve ter toda a informação de endereçamento necessária para a sua entrega. Dependendo do serviço de entrega usado, as mensagens podem chegar fora de ordem ou serem mal encaminhadas no trânsito.

As mensagens numa comunicação *sem conexão* são chamadas de *datagramas*. Essa forma de comunicação também é conhecida como *comutação de pacotes*. Pode-se fazer uma analogia deste tipo de comunicação com o serviço de entrega dos correios.

Não existe uma rota preestabelecida quando se utiliza o envio de mensagem por meio de datagrama. A figura 17 mostra os diversos caminhos que as mensagens podem percorrer.

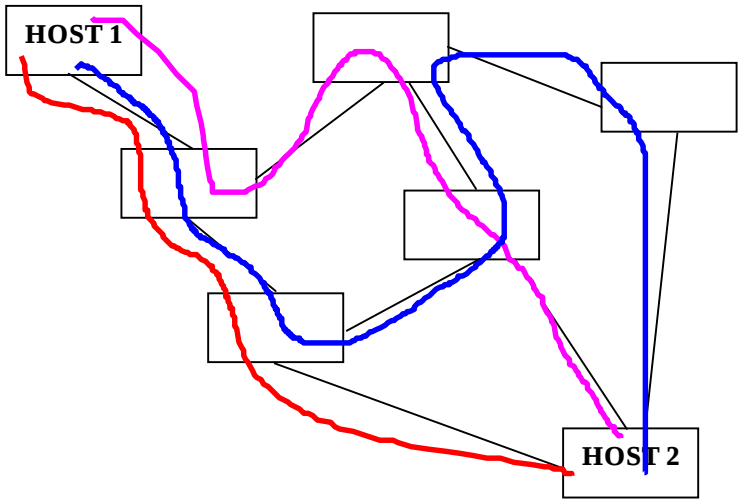


Figura 17 - Mensagem por Datagrama

A mensagem, que é dividida em vários pacotes, pode seguir caminhos distintos, tanto do *host 1* para o *host 2*, quanto no sentido inverso, podendo, inclusive, chegar ao seu destino em tempos diferentes, havendo assim, a necessidade de reordenação da mensagem pela parte responsável da comunicação pela aplicação do *host* de destino.

O outro tipo de comunicação é o *baseado em conexão*, onde um *circuito* prévio é estabelecido antes que a troca de dados se efetue. Quando um circuito estiver conectado, cada mensagem segue uma seqüência e sempre é direcionada ao longo do mesmo circuito (COMER,1993) (DUMAS,1995) (QUINN,1996).

Logo, cada mensagem só precisa de um identificador de circuito para ser direcionada para o seu destino. O recebimento de cada mensagem normalmente é confirmado e, se for o preciso, o controle de fluxo é usado para regular a velocidade com que as mensagens são enviadas.

A comunicação *baseada em conexão*, conforme apresentado na figura 18, garante a transmissão das mensagens. Estas são também divididas em pacotes, sem perda para o seqüenciamento dos pacotes nos *hosts* destinos, diminuindo assim o trabalho de implementação na parte de comunicação do aplicativo.

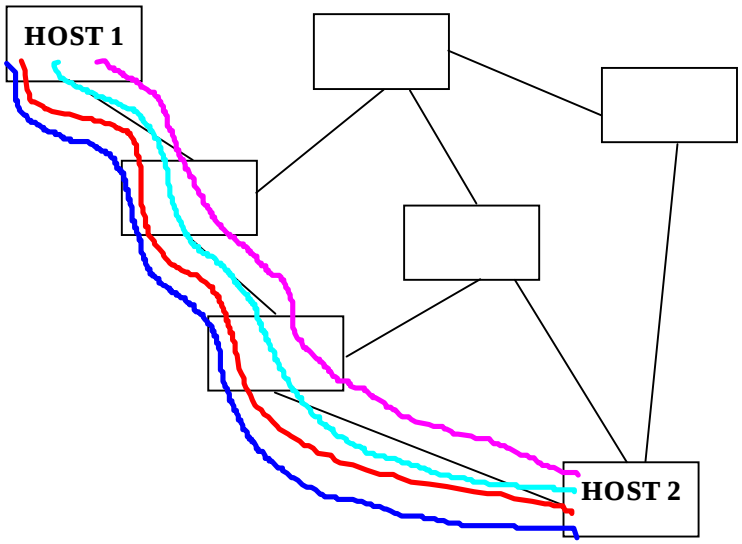


Figura 18 - Mensagem baseada em conexão

Quando a comunicação é finalizada, torna-se necessário desfazer a conexão para poder liberar os recursos de rede utilizados.

Deve-se notar que as mensagens sempre chegam ordenadas neste tipo de conexão. Um exemplo de comunicação baseado em conexão é o sistema telefônico.

A tabela 3 a seguir compara os dois tipos de comunicação.

Característica	Baseado em Conexão	Sem Conexão
Tipo de Mensagem	Fluxo de dados	Datagrama
Rota	Estática	Dinâmica
Endereçamento de Mensagem	Endereço de destino completo para estabelecer circuito; depois somente o ID do circuito	Endereço de destino completo em todas as mensagens
Confiabilidade	Seqüenciado, controle de erro e fluxo, entrega garantida	Sem garantias: mensagens podem ser perdidas ou chegar fora de ordem
Opções	Podem ser negociadas durante a preparação	Não disponíveis
Sincronismo	Explícito	Implícito
Overhead	Preparação e liberação do circuito	Rota da mensagem

Tabela 3 - Tipos de Conexão

4.3 - Aspectos de Sincronismo e Passagem de Mensagem

A comunicação entre Cliente e Servidor procede de forma implícita. Quando o Cliente espera a resposta da mensagem enviada para continuar o seu processamento, diz-se que o protocolo utilizado é um protocolo com *bloqueio*, onde o sincronismo entre Cliente e Servidor está implícito no mecanismo de passagem de mensagem (TANENBAUM,1995).

Caso o Cliente possa continuar suas tarefas, enquanto espera a resposta da mensagem, o protocolo de comunicação é um protocolo *sem bloqueio*. Isto ocorre quando o sistema operacional do Cliente é multitarefa ou multiprocessamento, possibilitando ao Cliente executar outras tarefas enquanto aguarda a resposta do Servidor.

A teoria de programação concorrente é baseada na noção de processos de comunicação sendo executados em paralelo a outros processos. Esses processos se comunicam

compartilhando memória ou passando mensagens por meio de um canal de comunicação compartilhado (AMARAL,1993) (VIDAL,1994). O termo IPC (Interprocess Comunication) se refere às técnicas utilizadas na passagem de mensagem.

No compartilhamento de memória, os processos concorrentes compartilham uma ou mais variáveis, e utilizam a mudança de estados dessas variáveis para se comunicarem. Essa técnica inclui espera ocupada, semáforos, regiões críticas condicionais e monitores (TANENBAUM,1995). Como esta técnica exige que os processos estejam na mesma máquina, não são considerados base para a programação Cliente/Servidor.

Em técnicas baseadas na passagem de mensagem, os processos enviam e recebem mensagens explicitamente, em vez de examinar o estado de uma variável compartilhada (AMARAL,1993) (COMER,1993). O benefício principal da passagem de mensagem é que existe pouca diferença entre o envio de mensagens a processos remotos ou locais. Portanto, a passagem de mensagem é poderosa para criação de aplicações em rede. Outra vantagem é que mais informações podem ser trocadas numa mensagem do que por mudança no estado de uma variável compartilhada.

A figura 19 mostra a troca de mensagem entre cliente e servidor sem o bloqueio por parte do cliente.

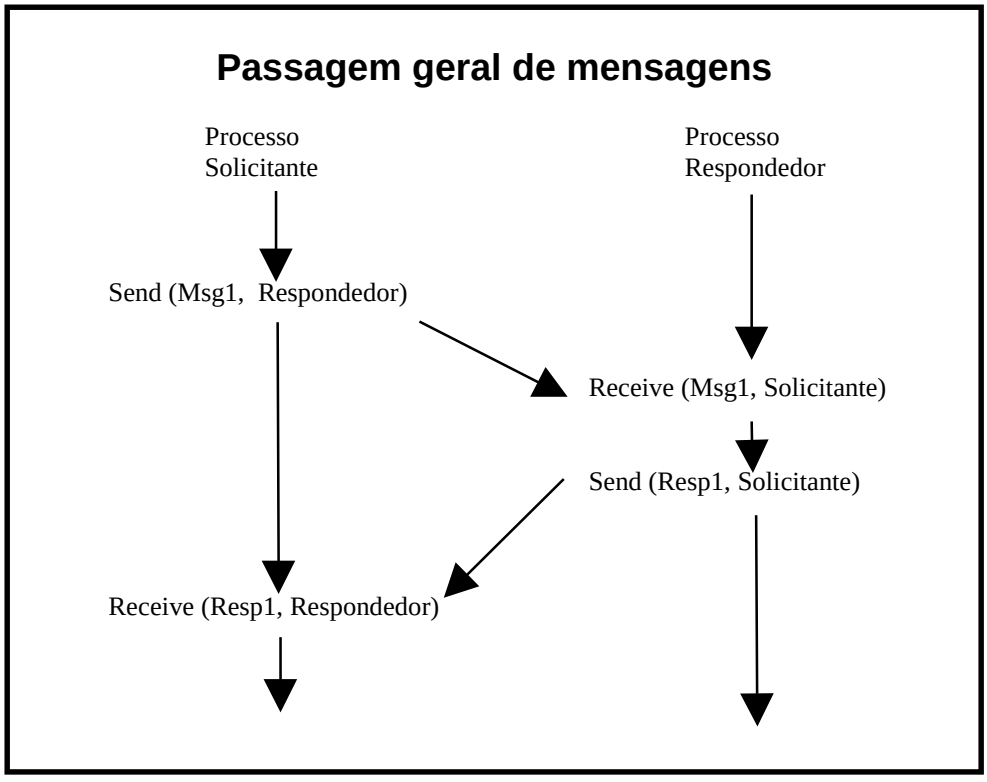


Figura 19 - Troca de mensagem sem bloqueio

4.4 - Conexão TCP/IP

Uma vez que a base de comunicação iria ser efetuada por meio de *sockets*, foi estabelecido também que o protocolo de comunicação utilizado seria o TCP/IP, possibilitando assim uma possível conexão futura com a plataforma UNIX.

Para este desenvolvimento, a plataforma a ser utilizada é o Windows NT 3.5 ou 3.51, que apresenta a robustez necessária para os processos multitarefas. A escolha deste ambiente também se deve ao fato dele suportar vários tipos de protocolos para comunicação, entre eles o TCP/IP (QUINN,1996).

A figura 20 mostra os protocolos de comunicação do Windows NT.

O protocolo TCP/IP é o protocolo padrão utilizado pela Internet, que possibilita a interligação de diversas plataformas. Existem vários serviços que utilizam o protocolo TCP/IP, entre eles pode-se encontrar o FTP, Telnet, E-mail, Ping e Finger, como pode ser visto na figura 21, junto com sua referência à camada do modelo OSI/ISO.

Na camada de Transporte pode-se notar a existência de dois padrões, o TCP, que é a comunicação com conexão e o UDP, que é a comunicação sem conexão, como já foi mencionado anteriormente na seção 4.2. Para o desenvolvimento deste protótipo foi adotado o padrão TCP, onde o controle de erro e o seqüenciamento dos pacotes das mensagens são automaticamente tratados, o que não ocorre no padrão UDP, necessitando que estes tratamentos sejam desenvolvidos.

O padrão TCP permite a abertura de uma conexão virtual entre a máquina fonte e a destino em todas as camadas do modelo OSI/ISO.

<table>
<tr><td rowspan="4">7

6

5</td><td rowspan="4">Aplicação Windows sockets</td><td rowspan="2">Aplicação Windows sockets</td><td>Aplicação Windows sockets</td><td>Aplicação Windows sockets</td></tr>
<tr><td>APIs proprietárias (opcional)</td><td>APIs proprietárias (opcional)</td></tr>
<tr><td rowspan="2">APIs proprietárias (opcional)</td><td rowspan="2">Aplicação (cont.)</td><td>Aplicação (cont.)</td></tr>
<tr><td>Aplicação (cont.)</td><td>APIs proprietárias (opcional)</td></tr>
<tr><td></td><td colspan="4">Aplicação (cont.)</td><td>Aplicação (cont.)</td></tr>
<tr><td></td><td colspan="5">Windows Socket API</td></tr>
<tr><td></td><td colspan="5">Windows sockets Dynamic Link Library (Winsock.DLL)</td></tr>
<tr><td></td><td colspan="5">APIs proprietárias</td></tr>
<tr><td>4
3</td><td>TCP/IP</td><td>DECNet</td><td>NetBEUI</td><td>Appletalk</td><td>SPX/IPX</td></tr>
<tr><td></td><td colspan="5">Driver de APIs padrões</td></tr>
<tr><td>2</td><td colspan="5">Driver de Protocolos Múltiplos (ODI, Packet Driver, NDIS)</td></tr>
<tr><td></td><td colspan="5">APIs de Hardware proprietário</td></tr>
<tr><td>1</td><td colspan="5">Interface de Rede (Ethernet, Token Ring, Serial, etc.)</td></tr>
</table>

Figura 20 Protocolos do Windows NT

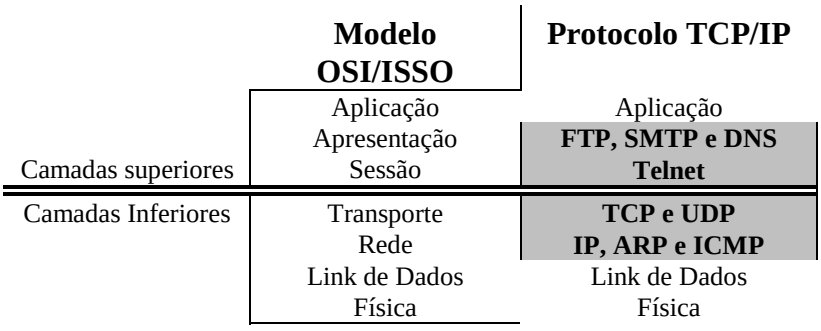


Figura 21 Protocolo TCP/IP

A Internet utiliza o padrão UDP, uma vez que as comunicações baseadas no protocolo TCP/IP se utilizam de portas de comunicação que, associadas aos *sockets*, permitem a troca de mensagens.

O protocolo TCP/IP disponibiliza 999999 portas (comprovação empírica), sendo que as portas de número 0 até a porta de número 1023 são reservadas para serviços pré-determinados, como por exemplo a porta 23, para o serviço Telnet, a porta 21 para o FTP, e assim por diante. Devido a este limite, caso a Internet utilizasse o padrão TCP, as máquinas que respondessem a um número muito grande de acessos, acabariam limitando a sua utilização.

No padrão UDP, por sua vez, quando existe uma solicitação de comunicação, o endereço IP do remetente da mensagem segue junto com a mensagem para o destinatário, de forma que o disponibilizador do serviço possa posteriormente enviar a resposta do serviço solicitado.

É importante salientar que, para que as comunicações transcorram normalmente, as APIs não necessariamente precisam ser as mesmas, muito embora os protocolos devam ser os mesmos, como pode ser visto na figura 22.

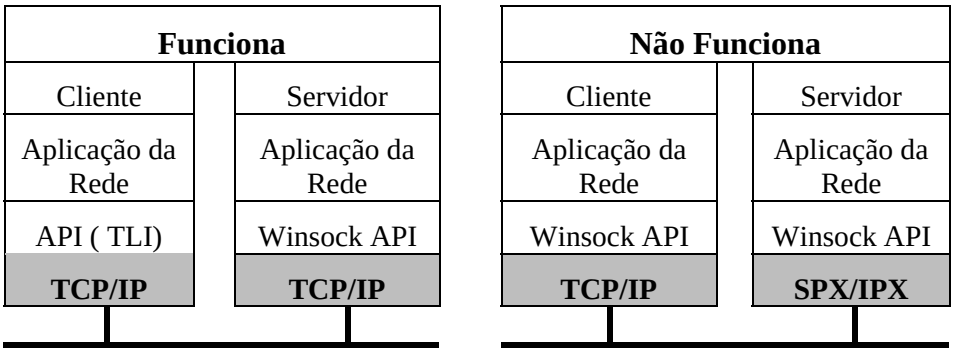


Figura 22 - Mesmo protocolo para uma comunicação correta

Os demais protocolos suportados pelo Windows NT, permitem a conexão com outros tipos de plataformas, como pode ser visto na tabela 4.

Protocolo	Plataforma
NetBEUI	Microsoft
SPX/IPX	Novell
Apple Talk	Mcintosh

Tabela 4 - Demais protocolos do Windows NT

O protocolo NetBEUI é de propriedade da Microsoft e permite a conexão com máquinas que estejam com os ambientes Windows 3.x, Windows NT 3.x / 4.0 e Windows 95 (DAVIS,1994). Ele é o responsável pela camada de Transporte e Rede, acrescentando uma extensão para a camada de Link de Dados. Conecta-se diretamente com o NetBIOS na camada de sessão (RENAUD,1994), que oferece as funcionalidades de transporte e rede do NetBEUI, como por ser visto na figura 15. Possui dois tipos de frames, um para fornecer um fluxo de dados confiável e o outro para frames de informação não numerados usados em Datagramas.

Os protocolos SPX/IPX são uma implementação dos protocolos XNS da Xerox. O protocolo SPX, igual ao SPP da Xerox, oferece um serviço de fluxo de dados confiável,

enquanto que o IPX, igual ao IDP, oferece um serviço de datagrama, permitindo a conexão entre máquinas do tipo PC que estejam rodando o Sistema Operacional de Rede da Novell (RENAUD,1994).

Por sua vez, o protocolo Apple Talk permite a conexão com máquinas da plataforma Macintosh, e é um protocolo totalmente particular. Os principais protocolos usados por máquinas Macintosh são os DDP - Datagram Delivery Protocol, ATP - AppleTalk Transaction Protocol e ADSP - AppleTalk Data Stream Protocol (RENAUD,1994).

4.5 - Sockets

O *Socket* teve origem na Universidade de Berkeley, como sendo a API (Application Programming Interface) de desenvolvimento do protocolo TCP/IP para o ambiente UNIX (COMER,1993). Ele é um dos mecanismos que o IPC possui para a troca de mensagens entre processos.

Um *Socket* é similar a um descritor de arquivo. Ele identifica o ponto final *(endpoint)* para a comunicação e é implementado como um inteiro positivo (ROBERTS,1995).

Existe uma diferença sutil, porém importante, entre descritores de arquivos e *socket*. Um descritor de arquivo é ligado a um arquivo ou dispositivo específico quando criado pelo comando *open*. Um descritor de *socket* não é ligado a local algum quando criado pelo comando socket. Uma aplicação pode decidir ligar-se a um endereço explicitamente usando um comando *bind*, ou pode fornecer endereços dinamicamente quando envia datagramas usando o comando *send*. Portanto, *sockets* podem ser usados como uma interface para transportes de rede baseados *em conexão* e *sem conexão* (COMER,1993) (DUMAS,1995).

O s*ocket*, que é um manipulador *(handle)*, está associado a um largo conjunto de dados armazenados na implementação do protocolo de rede. O termo *socket* é usado no S.O. UNIX para fornecer uma interface tipo arquivo às redes. Quando uma operação para criar um *socket* é chamada, o sistema retorna um manipulador, como descritor de *socket*. Esse descritor é usado em todos os outros comandos relacionados ao *socket* e é intercambiável com o descritor de arquivo usado em funções *read* e *write*.

Os dados associados ao *socket* incluem várias informações, como o endereço IP das máquinas que estão se conectando, as portas dos dois lados da conexão TCP e o estado da conexão corrente.

A camada de *sockets* dentro do contexto da comunicação corresponde a camada de Sessão do Modelo OSI/ISO, que tem como função o gerenciamento das sessões de comunicação processo a processo entre os *hosts*. Ela é também responsável por estabelecer e encerrar as conexões entre os aplicativos cooperantes (DUMAS,1995).

Dentro do protocolo TCP/IP, a camada TCP abrange a camada de Sessão e Transporte do Modelo OSI/ISO, favorecendo diretamente a programação dos *sockets* sobre o protocolo TCP/IP.

Para o ambiente Windows foi desenvolvido o *Windows Sockets - Winsock*, visando facilitar a interconexão entre as estações Windows, através do protocolo TCP/IP, que é base para o acesso a Internet. *Windows Sockets* é uma interface aberta para programação em rede sob o Microsoft Windows (QUINN,1996).

A especificação da interface do Windows sockets define claramente a divisão entre a aplicação de rede e o protocolo da rede.

A sua versão mais recente é o Winsock 2 (revisão 2.0.8 - 19/05/95). A versão 2.0 da especificação do Windows sockets se associa a arquitetura que o Windows NT utiliza para suportar múltiplos protocolos de fornecedores variados (QUINN,1996).

O Winsock API aumenta a funcionalidade dos *sockets* de Berkeley, ao acrescentar extensões específicas do Windows para suportar a natureza orientada a mensagens do S.O. baseado no Windows.

Todas as aplicações que hoje em dia acessam a Internet diretamente da residência do usuário, usando FTP, E-mail, Finger, Telnet, SMTP, entre outros, utilizam os sockets como base de comunicação, através do protocolo TCP/IP (ROBERTS,1995).

Os Sistemas Operacionais Windows NT e Windows 95 já possuem dentro de suas bibliotecas, as rotinas para suportar a programação para o protocolo TCP/IP via *sockets* (DAVIS,1994).

Para o ambiente Windows, o relacionamento entre as aplicações e o ambiente de rede, pode ser demonstrado na figura 23.

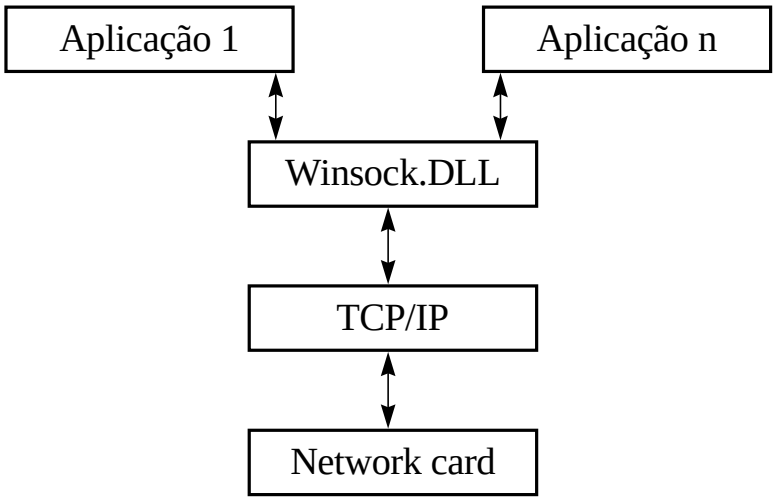


Figura 23 - Relacionamento da biblioteca dos *sockets* no ambiente Windows

Na API do *sockets* é possível determinar se a comunicação que se vai estabelecer é *com conexão - STREAM*, ou *sem conexão - DATAGRAMA*.

Para se trabalhar numa comunicação *baseada em conexão*, na camada de transporte do TCP/IP, o protocolo utilizado é o TCP, determinando que somente após o estabelecimento de uma comunicação é possível efetuar troca de mensagens.

Para a abertura de uma conexão com *sockets*, é necessário que várias funções da biblioteca Winsock sejam chamadas. A figura 24 demonstra a seqüência de operações tanto no lado Servidor quanto no lado Cliente.

No caso de se estabelecer uma conexão Stream, o Servidor é primeiramente inicializado. A função Socket() define o descritor no qual a aplicação se associará quando desejar transmitir uma mensagem. Posteriormente a função Bind() interliga o descritor ao endereço IP da máquina servidora e a porta TCP/IP na qual as transmissões irão ocorrer. A função Listen() permite ao Servidor ficar “escutando” qualquer solicitação de conexão.

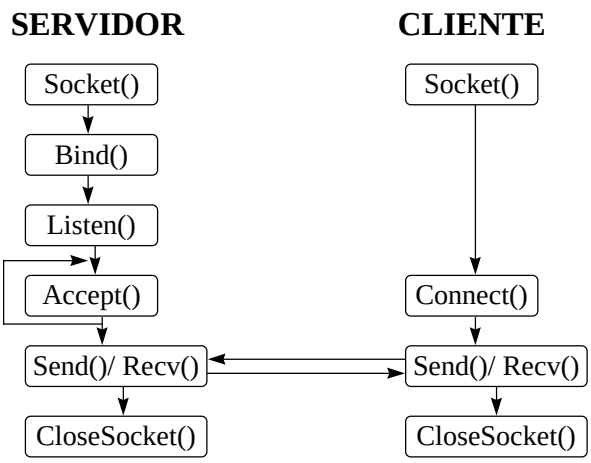


Figura 24 - Conexão Stream

Quando uma solicitação chega de um Cliente após ativar a função Connect(), o Servidor cria um processo filho mediante a função Accept(), numa nova porta TCP/IP. O pedido do Cliente é associado a esta porta, permitindo assim a transferência de dados pelas funções Send()/Recv(), deixando, desta forma, a porta original de conexão do Servidor livre para efetuar novas conexões.

Ao término da comunicação, o Cliente utiliza a função CloseSocket() para fechar a conexão, liberando a porta do processo filho do Servidor para ser ligada a outro processo de comunicação. O Servidor só irá utilizar esta função quando for desligado.

Por sua vez, para uma comunicação *sem conexão*, figura 25, a utilização das funções de *sockets* tornam-se mais fáceis de se implementar. Porém, é necessário lembrar que a utilização de comunicação baseada em Datagramas determina que as rotinas de recuperação das seqüências dos pacotes entregues a rede devem ser programadas, aumentando assim a possibilidade de falhas na implantação do sistema.

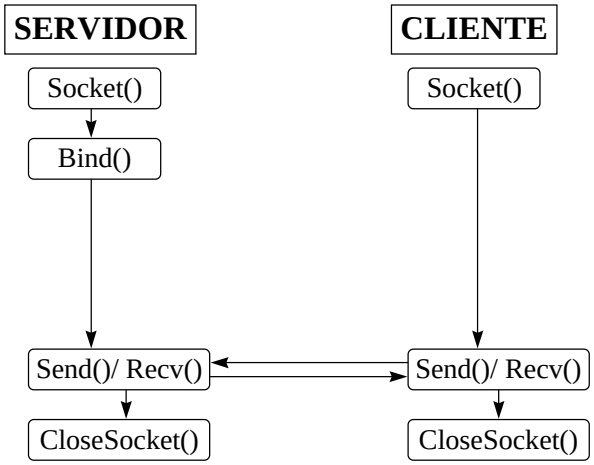


Figura 25 - Conexão Datagrama

A seqüência e as funções definidas para uma conexão Datagrama são as mesmas da conexão Stream, excluindo apenas as funções Listen(), Accept() e Connect(). Isto ocorre pelo fato do protocolo baseado em Datagrama não utilizar uma conexão real entre as máquinas origem e destino, implementando, automaticamente dentro das funções de troca de mensagens, Send()/Recv(), um cabeçalho com o endereço IP da máquina no qual será feita a comunicação.

Um ponto a ser observado, é que, para a transmissão, tanto na função Send() ou Recv(), os *sockets* somente operam com dados do tipo Char.

Para a transmissão de dados puramente do tipo caracter, o sistema desenvolvido baseado em *sockets* não impõe nenhuma restrição. Entretanto, para sistemas desenvolvidos sob o paradigma da Orientação a Objeto, que manipula tanto objetos simples, como complexos ou longos, é necessário que, ao se transmitir, se faça uma conversão da classe do objeto para o tipo Char. No momento em que a informação chega ao seu destino, é necessário que o receptor faça a reconversão, isto é, do tipo Char para o tipo original do objeto.

Esta peculiaridade para a transmissão de dados de tipos diferentes de Char, faz com que o destinatário da mensagem possua o conhecimento prévio de todos os tipos possíveis de objetos que podem ser transmitidos.

4.5.1 - Interfaces de Comunicação utilizando Sockets

Windows sockets é um dos mecanismos existentes para o desenvolvimento de aplicações em rede por meio de troca de mensagens. A compatibilização entre as bibliotecas de sockets de diferentes fornecedores permite a troca de bibliotecas sem afetar as aplicações. Como dito anteriormente, a API do Winsock representa uma barreira bem definida entre as aplicações de rede e os protocolos utilizados.

Para o desenvolvimento das interfaces de comunicação entre Cliente e Servidor foram utilizadas bibliotecas (PEDRIANA,1996) disponibilizadas na Internet, e, consequentemente, adaptadas para a comunicação entre os objetos do SIGO, uma vez que, a comunicação original se baseava somente na troca de mensagens de dados do tipo Char. Esta biblioteca foi construída em Borland C++ 4.5 e se constitui da comunicação entre máquinas Clientes e Servidor por intermédio de *sockets*. Esta biblioteca veio ao encontro dos interesses do projeto, já que o modo de comunicação baseado em RPC (*Remote Procedure Call*) não se constituía num ambiente confiável para operações que envolvessem objetos (AQUINO,1995).

A figura 26 mostra a relação do Modelo Windows *sockets* com o Modelo OSI/ISO.

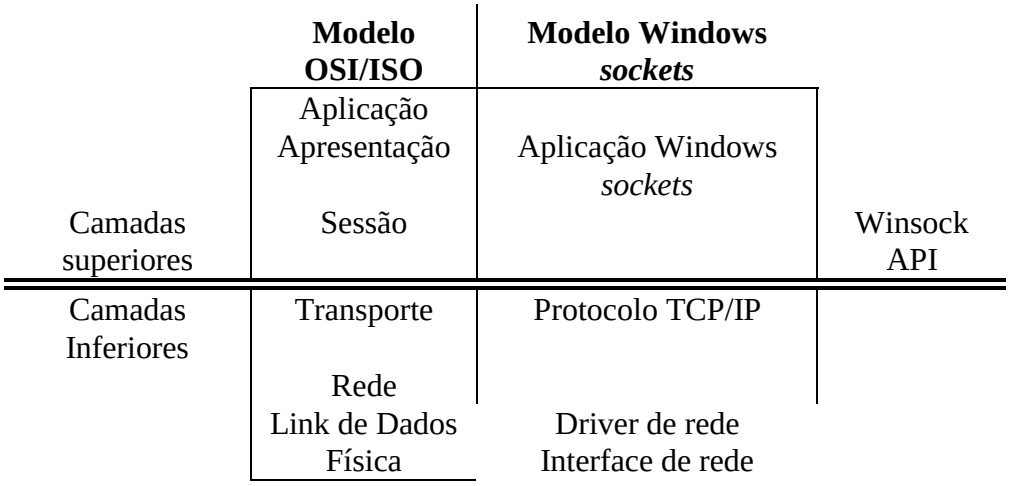


Figura 26 - Relação do Modelo Windows *sockets* com o Modelo OSI / ISO

Pode-se notar que a interação da aplicação com o protocolo TCP/IP faz-se somente pela API que o Windows *sockets* disponibiliza, minimizando assim maiores possibilidades de geração de erros.

Para que se estabeleça uma comunicação no padrão TCP, faz-se necessário seguir uma seqüência de operações pré-definidas como ilustrado na figura 2.21. O diagrama de estado da seqüência de conexão Stream Socket, pode ser visto na figura 27.

O Servidor deve ser sempre colocado “no ar”, antes de qualquer tentativa de comunicação. Para esta abertura de conexão com *sockets*, é necessário que várias funções da biblioteca Winsock sejam chamadas.

A função *Socket* define o *socket*, que é um descritor a partir do qual a aplicação se associará quando desejar transmitir uma mensagem. O Servidor, por sua vez, executa outras duas funções que são disparadas em seqüência: a função *Bind*, que associa o número de uma porta com o endereço IP da máquina servidora, e a função *Listen*, que fica “escutando” na porta selecionada esperando que haja uma solicitação de conexão.

Quando a função *Connect* for ativada pelo Cliente, uma solicitação de conexão para o Servidor é enviada e, caso a resposta a esta solicitação seja afirmativa, o Cliente e o Servidor passam para o estado de *conectado*.

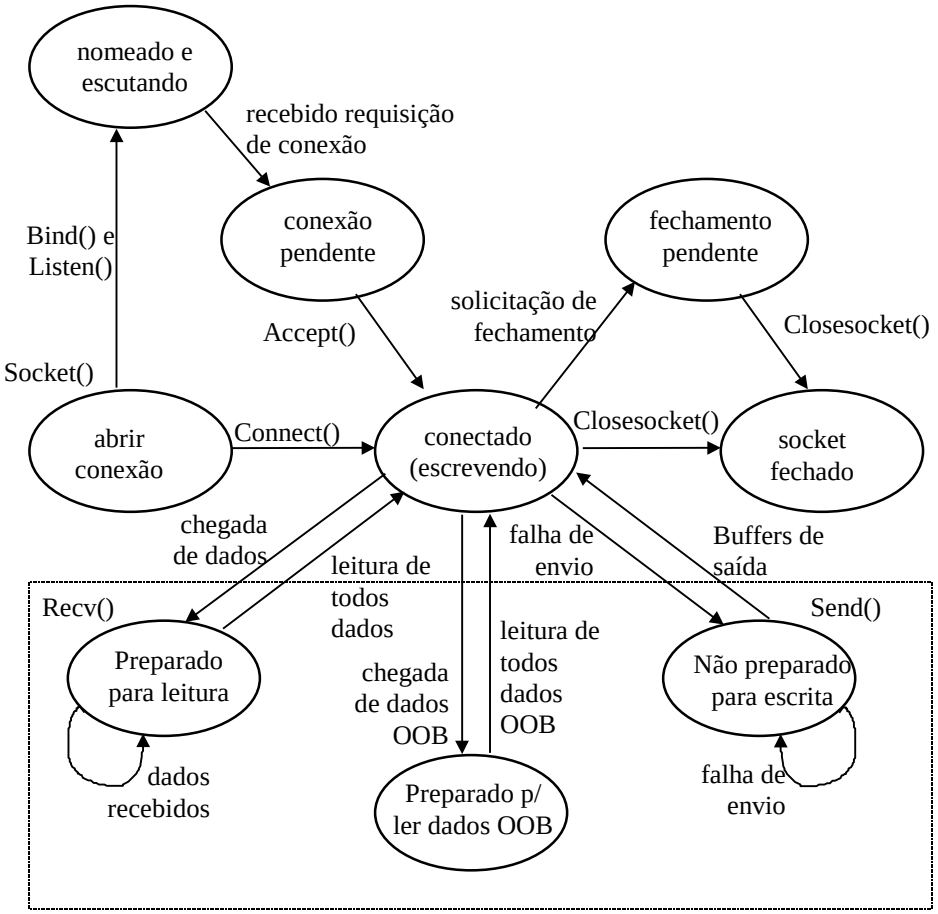


Figura 27 - Diagrama de estado do Stream Socket

Uma vez estabelecida a conexão, a troca de mensagem passa a ser efetuada e a função Recv retira do buffer de leitura as informações que foram trocadas entre o Cliente e o Servidor. Caso não se consiga enviar alguma mensagem pela função Send, por causa do estouro da capacidade do buffer, os dados são armazenados e retransmitidos posteriormente.

Os dados OOB (*Out-of_Band*) possuem um nível de prioridade de transmissão acima dos outros dados. Isto possibilita uma alteração na seqüência a ser transmitida, porém esta interferência na seqüência fica transparente ao usuário.

Ao término da comunicação, o Cliente utiliza a função CloseSocket para fechar a conexão, liberando a porta do processo filho do Servidor para ser ligada a outro processo de comunicação. O Servidor só irá utilizar esta função quando ele for desligado.

Portanto, primeiro o Servidor é posto no “ar” (1), em seguida o Cliente escolhe o nome de Servidor e a porta no qual sera feita a conexão (2a) e solicita uma conexão (2b), quando a conexão é aceita, o Servidor cria um processo filho, numa nova porta TCP/IP (3), associando o pedido do Cliente à esta porta, permitindo assim a troca de mensagens (4), deixando a porta

original da conexão do Servidor livre para efetuar novas conexões, como pode ser visto na figura 28.

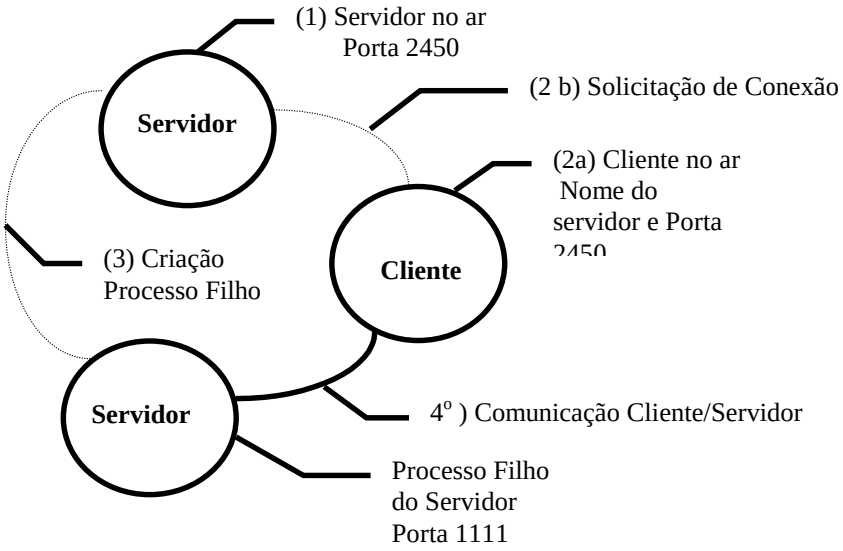


Figura 28 - Criação de Processo Filho para a comunicação

Caso ocorra uma perda de conexão, tanto por parte do Cliente quanto pelo Servidor sem antes ter sido efetuada a função CloseSocket, quem ainda estiver "no ar" não poderá reutilizar a porta que estava sendo utilizada até que o seu fechamento se dê de forma correta. Para evitar que problemas desta natureza possam vir a ocorrer, é necessário o desenvolvimento de uma função de *Time-out*, de modo a proporcionar um nível de segurança ainda maior para as conexões.

A função de *Time-out* fica checando se existe troca de informações nas conexões existentes, e permite que a função CloseSocket seja ativada automaticamente ao se verificar a inexistência de qualquer tipo de troca de mensagem após um determinado tempo, possibilitando assim a otimização das portas do protocolo TCP/IP.

5 - Bancos de Dados Orientados a Objetos

5.1 - Introdução

Os Sistemas Gerenciadores de Bancos de Dados Orientados a Objetos (SGBDOOs) surgiram da necessidade de suportar a programação orientada a objeto. Era necessário um repositório de dados para guardar os objetos criados pelas linguagens OO tais como a Smalltalk e a C++, entre outras. Esses objetos eram conhecidos como objetos persistentes (MARTIN,1994) (NAVATHE,1994), ou seja, dados que permanecem no sistema após a conclusão de um processo.

Os SGBDOOs tornaram-se importantes para certos tipos de aplicações com dados complexos, tais como CAD (*Computer Aided Design*) e CAM (*Computer Aided Manufacturing*) e o SIG (*Sistemas de Informações Geográficas*), servindo também para manipular BLOBs (Binary Large Objects), a exemplo de imagem, som, vídeo, texto e transações de longa duração (MARTIN,1994) (NAVATHE,1994) (KHOSHAFIAN,1993) (RUMBAUGH,1994).

A classificação dos Banco de Dados desde os relacionais (SGBDR) até os OO pode ser vista na figura 29 (POET,1997).

Os SGBDOOs existentes tais como Versant, O_2, Poet, GemStone, Itasca, entre outros, já incorporam características de comunicação baseadas na arquitetura Cliente/Servidor.

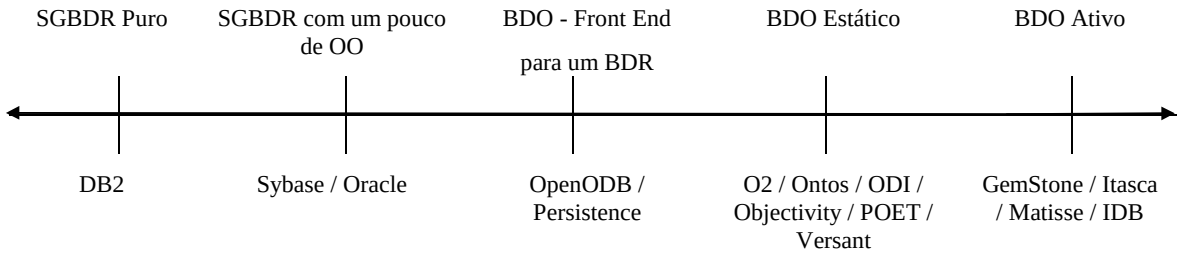


Figura 29 - Classificação dos SGBDR e SGBDOO

5.2 - Alguns SGBDOOs e suas versões Cliente/Servidor

5.2.1 - SGBDOO O_2

O sistema O_2 possui uma versão *workstation* e outra *server*, sendo que as duas versões possuem quase a mesma interface. A principal distinção é na implementação: a versão *workstation* é monousuário, enquanto que o *server* é multiusuário, possuindo total controle nas ações com o disco. Só através do *server* pode-se fazer acesso a qualquer tipo de informação armazenada em disco (BANCILHON,1992).

O O_2 é constituído por 8 módulos funcionais em sua arquitetura. Entre eles pode-se destacar o módulo OM (Object Manager) que é composto pelo COM (Complex Object Manager), MPM (Message-Passing Manager), TM(Transaction Manager), CM (Clustering Manager) e o IM (Index Manager).

O OM possui uma camada intermediária na qual gerencia o buffer de objetos (BM - Buffer Manager) e a camada de comunicação (CM - Communications Manager). No *server*, o gerente de comunicações fica escutando as requisições das *workstations* e dispara o processamento dos módulos correspondentes.

O gerente de comunicação do O2 é desenvolvido baseado em 4 critérios:

- A *simplicidade*, que se traduz na portabilidade, é conseguida pela escolha de ferramentas bem definidas e o protocolo de comunicação utilizado para a conexão entre Cliente e Servidor é o TCP/IP.
- A *transparência* resulta no fato de que o usuário final não reconhece de onde as informações estão sendo obtidas.
- A *performance* é obtida pela otimização do protocolo de comunicação que permite a transferência de objetos sem que resulte na criação de gargalos numa arquitetura Cliente/Servidor.
- Finalmente, a *confiança*, que é obtida pelo mecanismo de detecção de falhas que consegue prevenir o travamento do sistema por possíveis finalizações anormais dos processos envolvidos na comunicação.

Quando um processo é inicializado em uma workstation, um processo espelho é inicializado no server com o intuito de interagir com as camadas mais baixas do sistema. Após a comunicação ser estabelecida, os objetos são transferidos independentemente de um lado

para o outro. Contudo, quando uma quantidade grande de objetos é transferida, primeiramente eles são empacotados para minimizar os acessos a rede.

5.2.2 - SGBDOO GemStone

O OODBMS GemStone (GEMSTONE,1997), que utiliza a técnica de objetos distribuídos, oferece uma arquitetura Cliente/Servidor escalável e robusta com o seu sistema de controle de objetos. A comunicação do GemStone se baseia no ORB (*Object Request Broker*), que por sua vez utiliza o CORBA (*Common Object Request Broker Architecture*) para organizar os padrões dos objetos que servirão para a troca de mensagens. A OMG (*Object Management Group*) garante então, os padrões de interoperabilidade entre os sistemas de objetos gerenciados pelo CORBA, como visto nas figuras 30 e 31.

A OMG deliberou os seguintes padrões de trabalho para o Inter-ORB:

- ◊ o GIOP (*General Inter-ORB Protocol*) que especifica o formato das mensagens e a representação dos dados para toda a intercomunicação do ORB. O GIOP garante, portanto, que a ordenação dos bytes transmitidos não são problemas entre os ORBs existentes.
- ◊ o IIOP (*Internet Inter-ORB Protocol*) que permite a ligação entre o GIOP e o protocolo de comunicação TCP/IP.

O CORBA 2.0 possui ambos os protocolos de comunicação: o GIOP e o IIOP.

A arquitetura de objetos distribuídos do CORBA estende os limites do GemStone para as linguagens de objetos adicionais, interfaces de programação de aplicações e para serviços não objetos.

A adoção da padronização CORBA garante a interoperabilidade entre os diversos sistemas de objetos nas empresas.

A grande migração da indústria de sistemas monolíticos para sistemas distribuídos, fez com que a OMG focasse o CORBA como seu ponto de ação central, reunindo seus esforços para a padronizar os objetos componentes de sistemas de informações distribuídos.

A plataforma Windows desenvolve seus aplicativos baseados na tecnologia de objetos distribuídos através do COM (Component Object Model) e o OLE (Object Linking and Embedding).

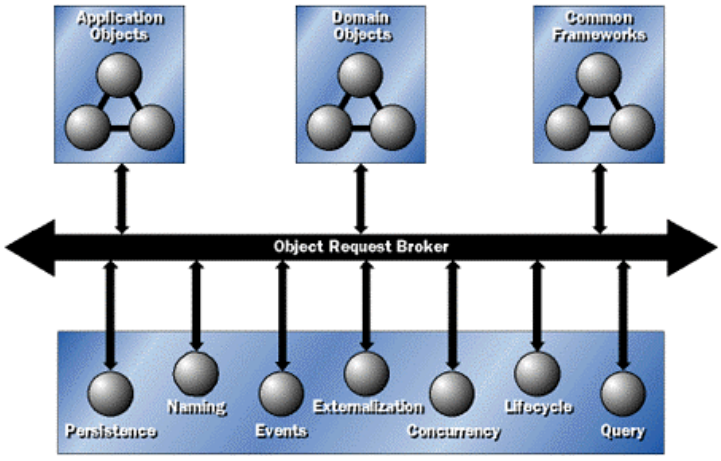


Figura 30 - Camada ORB para comunicação do GemStone

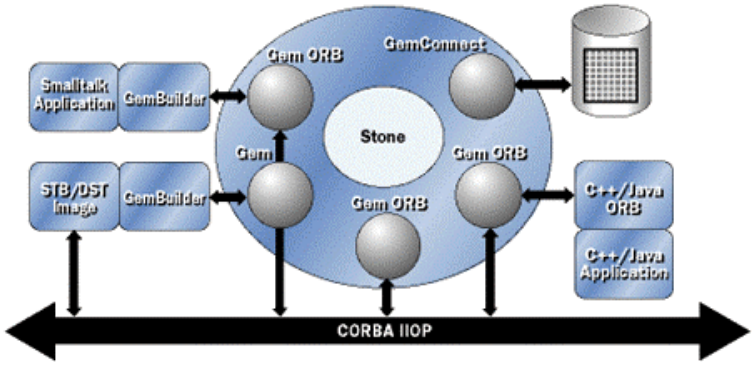


Figura 31 - Padronização CORBA para a Comunicação de Objetos Distribuídos

5.2.3 - SGBDOO POET

Outro OODBMS conhecido que também possui uma arquitetura Cliente/Servidor é o POET. O POET Engine e a aplicação cliente coexistem como um processo sem qualquer sobrecarga na comunicação (POET,1997).

A firmeza no gerenciamento dos dados junto com as facilidades da API disponíveis, permite ao usuário desenvolver aplicativos com características Cliente/Servidor, com uma total transparência da localização dos dados.

Uma aplicação pode abrir múltiplos banco de dados, que podem ser locais ou remotos.

O *UOS* (*Universal Object Server*) do POET emprega uma arquitetura Cliente/Servidor convencional, na qual múltiplos clientes acessam um ou mais servidores através da rede. Esta estrutura de comunicação representa uma arquitetura Cliente/Servidor em dois níveis formando uma comunicação mista podendo ser vista na figura 2.2c.

O UOS utiliza protocolos de rede padrão como o TCP/IP e o IPX/SPX, permitindo assim a operação transparente em redes homogêneas ou heterogêneas, fornecendo ao usuário o máximo de flexibilidade.

O sistema POET suporta tanto *single-threaded* quanto *multi-threaded*. Os clientes *single-threaded* incluem um *thread-safe* API que permite a serialização das chamadas para API do POET. Portanto, quando ocorre uma chamada, esta chamada fica em estado de bloqueio. Os clientes *multi-threaded* suportam sistemas operacionais *multi-threaded,* incluindo o Windows NT, OS/2 e UNIX

Single-threaded é uma operação sequencial na qual são alocadas as informações pertinentes ao processo, tais como identificação do processo e área de memória utilizada em cada *thread* criado. Entretanto, *Multi-threaded* permite o compartilhamento destas informações, quando da criação de processos filhos no processo pai, otimizando assim, a utilização da memória (PHAM,1996).

Os clientes incorporam a arquitetura de múltiplos leitores / único escritor, (*multiple reader/single writer*) usando consultas não bloqueadas e canais de comunicação simultâneos para um ou mais servidores POET

6 - Conclusão

Atualmente, a maioria das empresas está migrando para a computação distribuída, através de uma arquitetura Cliente/Servidor. Por ser uma área nova, é necessário o investimento em equipamentos, pessoal, treinamento e serviços para conseguir responder a demanda deste mercado globalizado.

A importância e a credibilidade desta tecnologia podem ser verificadas em função de grandes empresas, tais como: Oracle, Sybase, Informix, Digital e IBM, que desenvolveram soluções para atender a essa nova plataforma. Esta oferece acesso a dados localizados em diferentes Servidores, sem que o usuário perceba que eles podem estar vindo até de países diferentes, realidade facilmente conseguida através da Internet. Esta facilidade permite uma integração entre micros, *mainframes* e redes, a fim de se obter o melhor que a computação distribuída pode oferecer.

Este relatório apresentou as características básicas da arquitetura Cliente/Servidor.

A arquitetura Cliente/Servidor demonstra que veio realmente para ficar, e nada indica que um novo modelo esteja prestes a tirar o seu lugar.

Para acompanhar o avanço tecnológico nesta área, esta arquitetura foi implementada no SIGO (Sistema Gerenciador de Objetos) (MOURA,1997), transformando-o de um sistema monousuário em um sistema multiusuário (DESCHAMPS,1997).

Referências Bibliográficas

1. (AMARAL,1993) Amaral, W. H. “Arquitetura Cliente/Servidor Orientada a Objeto” Tese de Mestrado, IME, 1993.

2. (ANDREWS,1991) Andrews, G. R. “Concurrent Programming: Principles and Practice” Benjamin/Cummings, Redwood City, CA, 1991.

3. (AQUINO,1995) Botelho, Tomás de Aquino Tinoco, “Análise de Desempenho da Arquitetura Cliente/Servidor Orientada a Objeto”, Tese de Mestrado, IME, Dezembro/1995.

4. (BANCILHON,1992) Bancilhon, F. et al “Building an Object-Oriented Database System: The Story of O_2”, Morgan Kaufmann, 1992.

5. (COMER,1993) Comer, Douglas E. & STEVENS, David L. “Internetworking With TCP/IP Vol.III: Client/Server Programming and Application (Socket Version)”. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1993.

6. (CUSTER,1993) Custer, Helen “Windows NT”, Microsoft Press, Makron Books, São Paulo, 1993.

7. (DAVIS,1994) Davis, Ralph “Windows NT Networking Programming” Addison Wesley, 1994

8. (DESCHAMPS,1997) Deschamps, Eduardo R.P. “Sistema Gerenciador de Objetos em um Arquitetura Cliente/Servidor”, Tese de Mestrado, IME, Abril/1997.

9. (DUMAS,1995) Dumas, Arthur “Programando WinSock”, Axcel Books, Rio de Janeiro, 1995.

10. (GEMSTONE,1997) GemStone Inc. “GemStone’s Role in CORBA System” http://gemstone.com/Products/WP/corbawp.htm.

11. (HULQUIST,1997) Hultquist, Steve <ssh@vnet.ibm.com> “FAQ about Client/Server” http://non.com/news.answers/client-server-faq.html, 1997

12. (KALAKOTA,1997) Kalakota, Ravi “FAQ about Client/Server” <kalakota@uhura.cc.rochester.edu> http://non.com/news.answers/client-server-faq.html.

13. (KHOSHAFIAN,1993) Khoshafian, Setrag “Object Oriented Database”, John Wiley & Sons, Inc. , Canadá, 1993.

14. (LEFEBVRE,1994) Lefebvre, Alain “L’Architecture Client-Serveur”, Armand Colin, 2^{e} édition, 1994.

15. (MARTIN,1994) Martin, James “Princípios de Análise e Projeto baseados em Objetos” Ed. Campus, Rio de Janeiro, 1994.

16. (MCKIE,1997) McKie, Stewart “Everything you wanted to know about Client/Server computing but were afraid to ask” Eye on Technology, http://www.duke.com/controller/Issues/DecJan/Clientse.htm

17. (MOURA,1997) Moura, A.M.C.; Freitas, Luciani et al; “SIGO: Ambiente para Desenvolvimento de Aplicações Não-Convencionais em Banco de Dados” Relatório Técnico, RT016/DE9/ABR97, IME, Rio de Janeiro, Abri/1997.

18. (NAVATHE,1994) Navathe, Shamkant B. & Elmasri, Ramez “Fundamentals of Database Systems” 2nd Ed., Benjamin Cummings, CA, 1994.

19. (PEDRIANA,1996) Pedriana, Paul “OWLSock - Implementation” <paulp@ccnet.com>, http://users.ccnet.com/~paulp/owlsock/html/download.htm.

20. (PHAM,1996) Pham, Thuam Q. & Garg, Pankaj K. “Multithreaded Programming with Windows NT” Prentice Hall, 1996.

21. (POET,1997) POET Software “About Object Database” http://www.poet.com/t_oovsre.htm.

22. (QUINN,1996) Quinn, Bob & Shute, Dave “Windows Sockets Network Programming” Addison Wesley, 1996

23. (RENAUD,1994) Renaud, P. E. “Introdução aos Sistemas Cliente/Servidor” IBPI Press, RJ 1993.

24. (ROBERTS,1995) Roberts, Dave “Developing for the Internet with Winsock” Coriolis Group Books, 1995.

25. (RUMBAUGH,1994) Rumbaugh, James et al “Modelagem e Projetos baseados em Objetos” Ed. Campus, Rio de Janeiro, 1994.

26. (SALEMI,1993) Salemi, Joe. “Banco de Dados Cliente/Servidor” . IBPI Press, 1993.

27. (TANENBAUM,1995) Tanenbaum, Andrew S. “Distributed Operating Systems” Prentice Hall, 1995.

28. (TANENBAUM,1996) Tanenbaum, Andrew S. “Computer Networks” Prentice Hall, Third Edition, 1996.

29. (TAROUCO,1986) Tarouco, L. M. R. “Redes de Computadores”. McGraw-Hill ,SP 1986.

30. (VIDAL,1994) Vidal, Paulo César Salgado. “Modelagem para Arquitetura Cliente/Servidor Orientada a Objeto”. Tese de Mestrado, IME, 1994.