Universidade Federal da Paraíba
Centro de Ciências e Tecnologia
Coordenação de Pós-Graduação em Informática

Compêndio de Computação Distribuída

José Helvécio Teixeira Júnior

Campina Grande - Pb
Junho de 1995

José Helvécio Teixeira Júnior

Compêndio de Computação Distribuída

Dissertação apresentada ao Curso de Mestrado em Informática da Universidade Federal da Paraíba, em cumprimento às exigências para obtenção do Grau de Mestre.

Área de Concentração: Ciência da Computação

Jacques Phillipe Sauvé, PhD
(Orientador)

José Antão Beltrão Moura, PhD
(Co-orientador)

Campina Grande - Pb
Junho de 1995

T266c Teixeira Junior, Jose Helvecio
Compendio de computacao distribuida / Jose Helvecio Teixeira Junior. - Campina Grande, 1995.
il.

Dissertacao (Mestrado em Informatica) - Universidade Federal da Paraiba, Centro de Ciencias e Tecnologia.

1. Sistemas de Processamento Distribuido 2. Computacao Distribuida 3. Computacao Centralizada 4. Dissertacao I. Sauve, Jacques Phillipe, Dr. II. Moura, Jose Antao Beltrao, Dr. III. Universidade Federal da Pariaba - Campina Grande (PB) IV. Título

CDU 004.75(043)

COMPÊNDIO DE COMPUTAÇÃO DISTRIBUÍDA

JOSÉ HELVÉCIO TEIXEIRA JÚNIOR

DISSERTAÇÃO APROVADA EM 08.06.95

JACQUES PHILIPPE SAUVÉ, Ph.D
Presidente

JOSÉ ANTÃO BELTRÃO MOURA, Ph.D
Examinador

JORGE CESAR ABRANTES FIGUEIREDO, Dr.
Examinador

JOAQUIM CELESTINO JÚNIOR, Dr.
Examinador

Campina Grande - Pb.

Para Branca

"Ao lidar com questões importantes da vida, que nos afiguram tão difíceis que mesmo a prudência não nos pode ensinar o que fazer, sinto que possuímos fortes razões para seguir o que nos aconselha nosso espírito, e que é útil ter uma forte convicção de que as coisas que fazemos com boa vontade, e com a liberdade que acompanha a alegria, não deixarão de ter um resultado satisfatório."

René Descartes, em "O Discurso do Método"

Agradecimentos

Aos meus orientadores, pelo apoio e confiança em nosso trabalho.

À Suzana, por tudo.

Resumo

A Computação Distribuída é uma realidade para os dias atuais. Os principais usuários da tecnologia da informação estão se movendo de modelos de computação que têm como característica um processador central e recursos totalmente dependentes, para modelos que distribuem por muitos computadores conectados em uma rede a apresentação, os dados, e as tarefas de processamento de dados associadas com uma aplicação.

A busca pela Computação Distribuída já apresenta profundos efeitos na maneira como usamos os computadores para suportar as operações de negócios. Muitas organizações estão construindo e empregando aplicações distribuídas, objetivando obter uma maior abertura, funcionalidade, e produtividade dos usuários como resultado. No entanto, estas organizações estão também deparando-se com novas características técnicas, econômicas e de negócios, à medida que se movem em direção aos novos modelos encontrados nos ambientes organizacionais modernos. Conquanto mais e mais aplicações corporativas são construídas em arquiteturas distribuídas, o impacto irá mostrar-se ainda maior. ***A nossa tese objetiva fornecer informações que permitam que os mais diversos profissionais de Informática possam dominar esta transição da computação centralizada para a computação distribuída.***

Abstract

Distributed Computing is a reality nowadays. The main users of Information Technology are moving from computing paradigms based on a central processor and its dependent devices, towards new paradigms which distribute the presentation, the data, and the processing tasks associated with one application across many computers connected to a network.

Distributed Computing impacts the way we use computers to support business operations. Many organizations are building and applying distributed applications, in order to obtain greater openness, funcionality, and user productivity. Nevertheless, those organizations are also facing new technical, economics and business characteristics as they move towards new paradigms adopted by the modern business environments. As more and more corporate applications are built upon distributed architectures, the impact will be even greater. This thesis provides information that may help computer professionals harness the transition of a centralized computing model to a distributed computing one.

Visão Geral

Compêndio de Computação Distribuída

Conteúdo

Capítulo 4. Serviços de Correio Eletrônico

Capítulo 5. Serviços de Nomes e Diretórios Distribuídos

Capítulo 6. Sistemas de Arquivos Distribuídos

Capítulo 7. Bancos de Dados para a Arquitetura Cliente/Servidor

Capítulo 8. Interfaces Gráficas de Usuários

Capítulo 9. Gerência da Rede Corporativa

Lista de Figuras

Capítulo 2. Computação Distribuída: Contexto e Serviços

Capítulo 3. Serviços Não-Transparentes

Capítulo 4. Serviços de Correio Eletrônico

Capítulo 5. Serviços de Nomes e Diretórios Distribuídos

Capítulo 6. Sistemas de Arquivos Distribuídos

Capítulo 7. Bancos de Dados para a Arquitetura Cliente/Servidor

Capítulo 8. Interfaces Gráficas de Usuários

Capítulo 9. Gerência da Rede Corporativa

Capítulo 10. Segurança em Ambientes Distribuídos

Capítulo 11. Sistemas de Processamento de Transações Em Linha - Sistemas OLTP

Capítulo 12. O Uso de Objetos Distribuídos

Lista de Tabelas

Capítulo 1: Introdução

1.1. Motivação

A Computação Distribuída já é uma realidade para os dias de hoje. Os principais usuários da tecnologia da informação estão se movendo de modelos de computação que têm como característica um processador central e recursos totalmente dependentes para modelos que distribuem por muitos computadores conectados em uma rede a apresentação, os dados, e as tarefas de processamento de dados associadas com uma aplicação. Este movimento da computação centralizada para a computação distribuída já apresenta profundos efeitos na maneira como usamos os computadores para suportar as operações de negócios.

A maior parte das organizações está construindo e empregando aplicações distribuídas, e buscando maior abertura, funcionalidade, e produtividade dos usuários como resultado. No entanto, estas organizações estão também se deparando com uma série de novas características técnicas, econômicas e organizacionais à medida que se movem em direção da computação distribuída. À medida que mais e mais aplicações corporativas são construídas em arquiteturas distribuídas, o impacto irá mostrar-se ainda maior.

Nosso trabalho está centrado na ideia de fornecer informações que permitam que os mais diversos profissionais de Informática possam dominar esta transição da computação centralizada para a computação distribuída.

1.1.1. O Problema

No clima de negócios altamente competitivo dos dias de hoje, os sistemas de informações tornaram-se um recurso crítico. As grandes empresas estão investindo centenas de milhões de dólares a cada ano nos seus sistemas de informações. Para maximizar o retorno de seus investimentos, estas organizações estão mudando seus sistemas de informações para buscarem menores custos operacionais, produtividade melhorada, e a vantagem competitiva. Na maioria dos casos, esta busca significa a implementação de ambientes de computação e de informações distribuídos e seguros. Para construir este tipo de ambiente, uma organização precisa criar uma infra-estrutura de computação distribuída que forneça cesso global às informações e aos serviços de suporte, tais como o serviço de gerenciamento de dados distribuídos, o serviço de gerenciamento de arquivos distribuídos, serviços de mensagens, e serviços de diretórios.

Para os usuários finais, o termo ambiente da computação distribuída significa um ambiente de computação e de informações sem remendos, e que podem ser acessados por uma interface gráfica de usuário feita sob medida para estes usuários finais. Para os administradores de sistemas, um ambiente de computação distribuída significa a tecnologia em evolução, com uma complexidade crescente, e a falta (ainda) de ferramentas de gerenciamento. Para os desenvolvedores de aplicações, um ambiente de computação distribuída fornece serviços de suporte e uma infra-estrutura que oferece complexidade reduzida, interoperabilidade, portabilidade e a habilidade de se reutilizar código. As empresas esperam obter flexibilidade, escalabilidade, acesso em tempo real às informações, e o desenvolvimento e uso mais rápido de soluções para os problemas de seus negócios.

Durante os últimos anos, a rápida proliferação de computadores pessoais, de estações de trabalho a preços acessíveis e de redes de comunicações de dados confiáveis têm mudado a maneira como as empresas estão enxergando a computação. A maioria das grandes empresas, inclusive no Brasil, já implementaram ambientes de computação baseados em redes que compreendem mainframes, servidores Unix, e computadores pessoais. Estas empresas têm se deparado com o desafio de integrar ambientes dispersos em ambientes de computação distribuída escaláveis, confiáveis, seguros, e gerenciáveis.

1.1.2. Objetivo da Tese

O propósito de nossa tese é fornecer uma visão geral das tecnologias da computação distribuída, descrevendo desde as experiências com os primeiros sistemas operacionais distribuídos, passando por todos os conceitos relevantes e apresentando as estratégias de implementação e gerenciamento que irão beneficiar todos aqueles interessados em conhecer com mais detalhes os ambientes da computação distribuída. Nossa tese pretende ser útil a todos os profissionais de Informática que estejam envolvidos com o projeto, implementação ou gerenciamento dos ambientes da computação distribuída. Alunos de cursos de Ciências da Computação em universidades de todo o Brasil serão largamente beneficiados com a leitura e o estudo de nossa tese, pois sairão para um mercado de trabalho ávido por novas tecnologias, as quais muitas vezes não são apresentadas de maneira integrada no decorrer destes cursos. Os alunos de cadeiras como Redes de Computadores, ou de Sistemas Distribuídos podem utilizar a nossa tese para compreenderem melhor os assuntos relacionados com a implementação e o gerenciamento de todo um ambiente de computação distribuída.

O autor da tese e os orientadores são profissionais com experiência profissional e acadêmica de muitos anos na área, e todos já estiveram envolvidos com o projeto, a implementação e o gerenciamento de ambientes de computação distribuída. O grande diferencial do nosso trabalho baseia-se na apresentação de forma coesa, didática e integrada de uma série de conceitos sobre tecnologias tão diversas quanto o Gerenciamento Integrado da Rede ou o Uso de Objetos Distribuídos. Nossa perspectiva é de sermos úteis para leitores que tenham que suportar um grande número de usuários em um ambiente de computação distribuída autêntico, e não para aqueles leitores com uma visão menor do todo, como os que estão envolvidos com pequenas redes locais departamentais.

É bom que enfatizemos o aspecto da originalidade de nosso trabalho, o qual está baseado no mais criterioso trabalho de pesquisa endossado por uma grande disponibilidade de fontes de referências bibliográficas, todas recentes, publicadas em sua grande maioria a partir de 1993. Nosso trabalho se estendeu por mais de dois anos, e incluímos como parte de nosso esforço a oportunidade de termos aproveitado a aplicação de nossos conhecimento de maneira prática, tendo atendido a consultorias especializadas para grandes empresas envolvidas com as questões de projeto, implementação, escolha de ferramentas e migração de aplicações para o novo ambiente da computação distribuída. Consideramos que os principais tópicos tenham sido escolhidos de maneira a suprirem a maior parte de todo o vasto conteúdo de conceitos relacionados.

1.2. Organização do Trabalho

Esta tese pretende fornecer um tutorial para as tecnologias da computação distribuída e ajudar o leitor a compreender aspectos envolvidos na implementação e no gerenciamento de um ambiente de computação distribuída. Assumimos que o leitor compreende os conceitos básicos de comunicações de dados, sistemas operacionais e arquitetura de computadores. Apesar de cada capítulo ser auto contido, a organização dos mesmos em sua seqüência original é a mais recomendada. Os capítulos estão distribuídos da seguinte maneira:

Capítulo 1. Introdução
Capítulo 2. Computação Distribuída: Contexto e Serviços
Capítulo 3. Serviços Não-Transparentes
Capítulo 4. Serviços de Correio Eletrônico
Capítulo 5. Serviços de Nomes e Diretórios Distribuídos
Capítulo 6. Sistemas de Arquivos Distribuídos
Capítulo 7. Bancos de Dados para a Arquitetura Cliente/Servidor
Capítulo 8. Interfaces Gráficas de Usuários
Capítulo 9. Gerência da Rede Corporativa
Capítulo 10. Segurança em Ambientes Distribuídos
Capítulo 11. Sistemas de Processamento de Transações Em Linha - Sistemas OLTP
Capítulo 12. O Uso de Objetos Distribuídos
Capítulo 13. Conclusão
Bibliografia

Adicionalmente, fazem parte de nosso trabalho um índice completo, mais uma lista de figuras e de tabelas. A bibliografia apresentada representa o que existe de mais atual em termos de tópicos relacionados com as tecnologias da Computação Distribuída. Todas as citações no trabalho são consistentes, inclusive a nível de figuras e tabelas.

1.3. Trabalhos Futuros

Como o nosso objetivo maior é de publicarmos nossa tese em forma de livro didático, sugerimos alguns trabalhos que podem ser feitos com o objetivo de complementar o que fizemos até aqui. São sugestões que têm valor para uma publicação, mas que certamente são dispensáveis em uma tese acadêmica, e, portanto, consideramos completo o nosso trabalho. Nossas sugestões são:

- índice remissivo;
- referências bibliográficas por capítulo;
- bibliografia comentada para os textos clássicos;
- lista de siglas e abreviações;
- um glossário.

Como continuação de nosso trabalho, propomos que seja escrito um segundo volume para tratar particularmente das peculiaridades envolvidas com o desenvolvimento de aplicações para os ambientes da Computação Distribuída.

Capítulo 2. Computação Distribuída: Contexto e Serviços

2.1. O Papel da Tecnologia da Informação

O uso da "tecnologia pela tecnologia" pouco ou nada significa nos ambientes empresariais modernos. Simplesmente "jogar" computadores sobre um problema empresarial existente, não irá resolvê-lo. De fato, o uso inadequado da tecnologia pode bloquear totalmente o processo de modernização de uma empresa caso venha a reforçar velhas formas de pensamento e velhos padrões de conduta. Uma empresa incapaz de mudar o seu pensamento organizacional não irá fazer uso adequado das funcionalidades e benesses da computação distribuída como solução moderna fornecida pelos avanços recentes da tecnologia da informação ([Bur92] [Gil91] [Gow91]). A tecnologia da informação é um instrumento de transformação dos processos de negócios das empresas, visando aprimorá-los. O foco não pode concentrar-se exclusivamente nela [Ber94].

Estudos recentes [MP94] mostraram que a produtividade por funcionário nas empresas americanas e europeias aumentou de apenas 1% nos anos 80, um período claramente marcado pelo crescente uso de computadores. Com dados como estes em mãos, algumas pessoas chegaram mesmo a pôr em dúvida a eficácia da tecnologia para resolver os problemas de negócios nas empresas.

Nos anos 80, as empresas gastaram quase 1 trilhão de dólares em tecnologia de informação. Ainda assim, a produtividade não apresentou sinais de melhorias consideráveis, e em alguns setores, como o de serviços, chegou a diminuir. O prêmio Nobel dos laboratórios Bell, Dr. Arno Penzias [Man92], afirma que há uma perda real em produtividade quando os profissionais, individualmente, precisam trabalhar por si, isolados de seus colegas, como por exemplo, usando microcomputadores pessoais isoladamente e não conectados em rede. Felizmente, constatou-se que a raiz do problema da baixa produtividade não estava nos equipamentos e programas de computadores, mas em seu uso inadequado, o que impede que as empresas extraiam todo o potencial de benefícios que a tecnologia da informação tem a oferecer.

Atualmente, mais experientes, as empresas em número cada vez maior estão reconstruindo seus procedimentos internos e revendo a estrutura de seus sistemas. As mudanças são radicais, e é neste contexto que se inserem perfeitamente as soluções fornecidas pela computação distribuída. A busca por soluções da computação distribuída, como o uso de ferramentas para a melhoria da produtividade em grupos de trabalho, conhecidas como "groupware" ([Ray93] [Ros90] [Ste93]), tem sido essencial para a melhoria dos níveis de produtividade nas empresas. Existem estudos [Man92] que mostram ganhos de produtividade em até 90 % pelo uso de ferramentas deste tipo.

A produtividade organizacional [Sey93] constitui o retorno de investimentos que está faltando, e sem o qual, a indústria de computadores, as empresas, e a economia global, não poderão progredir. As novas tecnologias da informação, sozinhas, não conseguirão resolver este problema da baixa produtividade, porque sua fonte é de natureza organizacional. Mudanças importantes são necessárias nas práticas organizacionais, no comportamento e na cultura das empresas. Uma empresa que encoraje a disseminação de novas idéias, uma empresa que deseje aprender e tentar experiências novas com seus empregados, terá como retorno a sua própria transformação, permitindo que sejam criadas e atendidas as novas condições de mercado [Sey93].

Com os conceitos modernos de administração de empresas, verdadeiras revoluções vêm ocorrendo no mundo dos negócios, de onde certamente sairão empresas mais ágeis, e departamentos de informática com o perfil totalmente renovado. A moderna tecnologia da informação, agindo como um capacitador, permite que as empresas passem a realizar seus trabalhos de formas radicalmente diferentes, possibilitando um total rompimento com antigas regras, e permitindo que se criem novos modelos para seus processos [HC94].

Assim, a tecnologia da informação está comandando uma busca em direção a uma crescente complexidade social [Bur92]. Em toda a história da humanidade, melhorias radicais na habilidade de se gerar e compartilhar informações têm sempre disparado uma grande onda de informações adicionais, o que por sua vez tem sempre gerado rajadas de inovações que dão vida a novas entidades, novos tipos de comportamentos pessoais e novas maneiras de se viver. Quanto mais heterogêneo um ecossistema, mais formas inteiramente novas de vida são geradas [Gou89].

Esta onda de informações torna possível que novas idéias sejam colocadas lado a lado, sob um novo ponto de vista. Portanto, sem sombra de dúvidas, a tecnologia da informação será a responsável pelo próximo estágio de nossa própria evolução social.

2.2. As Mudanças nos Ambientes de Negócios

Nos anos 80 a responsabilidade pelas contas de lucros e prejuízos nas empresas foi repassada para os departamentos [Vil91]. O resultado foi a dispersão dos sistemas de informações e uma maior autoridade local, a nível de departamentos, na aquisição de equipamentos (micros, redes, software).

Como resultado, houve também uma grande dispersão do poder de processamento, das informações de negócios e das aplicações, que espalharam-se por toda empresa. Desta forma, diminuiu-se a possibilidade de um único fornecedor ter controle total sobre as contas de informática de uma organização.

Nos anos 90, as empresas começam a deparar-se com uma demanda cada vez mais sofisticada por parte dos seus clientes, a concorrência cada vez mais acirrada, e necessidades de respostas rápidas às mudanças do mercado global ([HC94] [Sta94a]). Para atender a estas três exigências, é necessária uma interação e cooperação crescentes entre grupos de trabalho e departamentos nas empresas, assim como entre as próprias empresas.

No novo ambiente de negócios, os vendedores já não têm mais o controle. São os clientes que mandam. São eles agora que informam aos fornecedores o que desejam, como desejam, e quanto pagarão. A idéia tradicional de "mercado de massa" já não existe mais. Os clientes individuais - firmas ou consumidores - exigem um tratamento individualizado. Eles buscam produtos configurados para suas necessidades, cronogramas de entrega adaptados aos seus planos de fabricação ou às suas horas de trabalho, e prazos de pagamento que lhes sejam convenientes.

Em um passado recente, aquelas empresas de bom desempenho eliminariam as empresas inferiores, pois seus preços menores, melhor qualidade, e o melhor nível no atendimento aos clientes tornar-se-iam o padrão para todos os concorrentes [Eng94b]. Um padrão adequado não é mais suficiente. Se uma empresa não consegue se equiparar entre as melhores em sua categoria, logo não mais terá o seu lugar ao sol.

Uma consideração interessante é o fato de que neste mundo de concorrência sofisticada, empresas novatas não obedecem a regras, elas definem as novas regras de como se organizar um negócio, e esta qualidade lhes fornece um diferencial fundamental [Bal92]. Por exemplo, a inovação das estações de trabalho da empresa americana Sun Microsystems mudou totalmente o curso da história de todos os fabricantes de computadores do mundo. A Sun foi uma das primeiras empresas engajadas na divulgação e viabilização das funcionalidades oferecidas no contexto da computação distribuída, provando que a tecnologia pode alterar a natureza da concorrência de forma inesperada para a maioria das empresas.

Com a globalização da economia, as empresas enfrentam um número maior de concorrentes, cada um deles capaz de introduzir novos produtos ou serviços no mercado. A rapidez da mudança tecnológica também promove a inovação. Os ciclos de vida dos produtos passaram de anos para meses. O tempo-para-mercado (time to market) também foi bastante reduzido, acirrando ainda mais a concorrência [Vil91]. Por exemplo: com o uso de ferramentas CAD/CAM (Computer Aided Design/Computer Aided Manufacturing) para o projeto e manufatura concorrentes busca-se coordenar estruturadamente grupos diversos e dispersos para se reduzir o período entre a concepção do produto e a entrega do produto acabado para os clientes.

Peter Drucker, o pai da moderna administração de empresas já previa esta rapidez nas mudanças quando escreveu em 1975 o clássico artigo "Managing for Business Effectiveness" [Dru91], onde ele afirmava: "...as mudanças são tantas e tão rápidas, que cada produto e cada atividade de uma empresa começam a tornar-se obsoletos tão logo sejam inicializados. Cada produto, cada operação, e cada atividade em um negócio deveria ser avaliada a cada dois ou três anos. Esta avaliação deveria gerar uma proposta para um novo produto, uma nova operação ou uma nova atividade". Naquela época os executivos consideravam esta sugestão algo impensável. Estamos falando atualmente em meses para ciclos de vida de produtos, e não mais em anos.

Com a intensificação da competição a nível global, a maioria das empresas considera a melhoria na produtividade - medida em termos de estruturas de custos enxutas, tempo-para-mercado (time-to-market) reduzido, e ciclos de vida menores para produtos - absolutamente essenciais para a sua sobrevivência ([Man92] [Dru91]).

A tecnologia da informação, particularmente, através do uso da computação distribuída, irá fornecer muitas das facilidades para este novo ambiente de negócios, porém o seu uso deve ser coordenado a nível de mudanças maiores nas próprias estruturas organizacionais, incluindo o rompimento com velhas tradições [Gil91].

2.3. Reengenharia - A Revolução nas Empresas

James Martin e Clive Filkenstein, dois renomados professores do mundo da Ciência da Computação, autores de vários livros sobre os mais diversos assuntos em Informática e palestrantes de renome internacional, definiram um termo que foi muito divulgado nos anos setenta: *Engenharia da Informação* [Dew93]. Em essência, a Engenharia da Informação enfocava o conhecimento dos negócios - o que deveria ser tratado e como - e não a tecnologia em si. Representou uma mudança do modelo de processos *com dados* para o modelo de processos *baseados em dados* (data-based). Sistemas desenvolvidos com o uso deste enfoque poderiam responder mais rapidamente às mudanças de negócios por que a arquitetura do sistema não era dirigida para o uso de uma tecnologia em particular. Esta arquitetura era baseada nas necessidades de negócios.

Foi a partir das idéias iniciais da Engenharia da Informação que surgiram os conceitos modernos de ***Reengenharia Empresarial*** [HC92]. A reengenharia redesenha e repensa todos os sistemas de negócios.

A reengenharia ([HC94] [Ber94] [Hal93A] [Ueh94]) ("reengineering") é um termo que foi popularizado pelo consultor americano Michael Hammer para explicar que os princípios administrativos adotados atualmente tornaram-se inadequados no novo cenário mundial dos negócios, marcado por mudanças rápidas e acirrada competição: "As tecnologias avançadas, a derrubada das fronteiras entre os mercados nacionais, e as expectativas alteradas dos clientes, que contam agora com mais alternativas do que em qualquer época anterior, combinaram-se para tornar as metas, os métodos e os princípios organizacionais básicos da clássica corporação tristemente obsoletos".

O que se busca hoje em dia é uma organização suficientemente flexível para se ajustar rapidamente às condições de mudanças do mercado, suficientemente enxuta para derrotar o preço de qualquer concorrente, suficientemente inovadora para manter-se tecnologicamente atualizada em seus produtos e serviços, e suficientemente dedicada para fornecer o máximo de qualidade e atendimento aos clientes [Gow91]. É comum que estes resultados não sejam alcançados pelas empresas.

A definição formal de reengenharia, segundo o próprio Hammer seria: "o repensar fundamental e a reestruturação radical dos processos empresariais que visam alcançar drásticas melhorias em indicadores críticos e contemporâneos de desempenho, tais como custos, qualidade, atendimento e velocidade" [HC94]. Desta definição, quatro palavras-chaves destacam-se:

- *Fundamental*: a reengenharia determina primeiro o que uma empresa precisa fazer, depois como fazê-lo;
- *Radical*: a reengenharia trata da reinvenção da empresa, não de sua melhoria, de seu aperfeiçoamento ou de sua modificação;
- *Drástica*: melhorias marginais exigem um ajuste fino, melhorias drásticas requerem a destruição do antigo e sua substituição pelo novo;
- *Processos*: a reengenharia tem que voltar-se para a redefinição processos organizacionais básicos, e não para departamentos ou outras unidades organizacionais.

A Reengenharia não busca reformular o que já existe ou fazer mudanças tímidas que deixem a estrutura básica de uma empresa intacta. Não se trata de fazer remendos, de retocar os sistemas existentes para funcionarem melhor. A Reengenharia busca abandonar velhos sistemas e começar de novo. A reengenharia prevê esforços que gerem saltos quânticos de desempenho, pequenas melhorias não interessam.

A computação distribuída adequa-se perfeitamente como fornecedora de soluções para as empresas que buscam os modelos adotados pela Reengenharia. Para o nosso estudo, citar a reengenharia é mostrar o que dissemos no início do capítulo, "o uso da tecnologia pela tecnologia pouco ou nada significa". A moderna tecnologia da informação como um capacitador essencial no tocante às soluções fornecidas pela computação distribuída, tem uma importância fundamental para o processo de reengenharia. Entretanto, as empresas não devem enxergá-la como o único elemento essencial deste processo.

As empresas não podem tomar conhecimento de uma nova tecnologia hoje e aplicá-la amanhã. É preciso tempo para estudá-la, para compreender a sua importância, para viabilizar seus usos e potenciais, para vender esses usos dentro da empresa, e para planejar a sua aplicação. Uma organização capaz de executar estes passos preliminares antes que a tecnologia realmente se torne disponível conquistará uma posição de vantagem significativa entre seus concorrentes, conquistará o que se convencionou chamar de "diferencial de negócios" (business advantage) [HC94]. Aquelas empresas mais capazes de reconhecer e perceber os potenciais de novas tecnologias desfrutarão de uma vantagem constante e crescente com relação aos concorrentes.

Como declarou em entrevista para a revista Harvard Business Review o presidente da empresa americana Silicon Graphics, Ed McCracken [Pro93]: "A chave para a vantagem competitiva não é reagir ao caos; é produzir o caos." Ele acredita, como muitos visionários da indústria de informática, que estar no topo da onda das inovações tecnológicas é a única maneira de se chegar à vantagem competitiva (ou seja, de se conseguir chegar ao diferencial de negócios que levará à vantagem competitiva sobre os concorrentes).

É por este motivo, a busca pelo diferencial de negócios, que estamos interessados na computação distribuída, e é com este intuito que falamos durante o desenvolvimento de nosso trabalho de todos os aspectos técnicos da computação distribuída, para que a mesma não seja vista como um simples modismo ou fruto de elocubrações acadêmicas com poucas soluções objetivas para os problemas práticos enfrentados pelas empresas.

2.4. Computação Distribuída: Visão Geral

À medida que um número crescente de empresas iniciam a transição para os ambientes da computação distribuída elas precisam modificar suas estruturas organizacionais atuais, na sua maioria centralizadas, para poderem implementá-la. As promessas da computação distribuída são muitas e baseiam-se em processadores poderosos que utilizam a tecnologia RISC (SPARC, Alpha, Power-PC, etc.) ou CISC, que oferecem desempenho superior, e um preço extremamente competitivo e atraente; em servidores escaláveis e em periféricos adaptáveis a uma vasta gama de aplicações fornecem flexibilidade; na descentralização dos recursos, evitando a total dependência das máquinas hospedeiras enforça a confiabilidade; na adaptação das aplicações aos seus ambientes mais apropriados de software, hardware e necessidades pessoais, o que gera imediata economia de custos ([Cra93a] [Lan92]).

A computação distribuída consiste de elementos chaves como redes locais que podem interconectar-se, interfaces gráficas de usuários (GUIs) para o acesso facilitado aos usuários, módulos de SGBDs (Sistemas de Gerenciamento de Bancos de Dados) distribuídos para a recuperação eficiente de informações, linguagem SQL e ferramentas SQL ("Structured Query Language") para o acesso padronizado aos SGBDs, e adicionalmente uma variada gama de ferramentas de software que ajudam ao usuário na construção das aplicações distribuídas.

Mudanças drásticas nos produtos da tecnologia da informação também validam a busca pela computação distribuída [Vil91]. Os principais fornecedores de software comerciais que atuam ativamente no mercado de software para plataformas de grande porte estão fazendo grandes revisões em seus produtos, para que estes passem a suportar a computação distribuída [MS94]. A oferta de software de banco de dados, e ferramentas de desenvolvimento para suportar sistemas de bancos de dados distribuídos na arquitetura cliente/servidor, tem crescido sensivelmente, o que ajuda na migração de arquiteturas centralizadas para distribuídas ([Ski94] [Skr94] [Hur93f] [Col93d]). Por sua vez, os sistemas operacionais modernos implementam características que melhoram o processo de implantação da computação distribuída. Aliados a todas estas mudanças, a disseminação de hardware servidor na forma de multiprocessadores (com multi-processamento simétrico) e RAIDs ("Redundant Array of Inexpensive Disks") ([Per93b] [Tay93b]), melhoram consideravelmente o desempenho e reduzem a vulnerabilidade dos sistemas.

2.4.1. Computação Distribuída - Conceituação Formal

A Computação Distribuída envolve múltiplas interpretações - existem definições técnicas, de marketing, de esforços de vendas, etc. Apresentamos aqui uma definição do ponto de vista de implementação [Vil91]:

A ***Computação Distribuída*** engloba todos os produtos, interfaces e infra-estrutura necessários para permitir que aplicações sejam construídas e passem a funcionar de maneira consistente, seja em um ambiente de rede distribuída, e eventualmente heterogênea, ou em um ambiente centralizado.

A **computação distribuída** ([Kha94a] [Mil91] [Nad94a] [OSF90b] [OSF92c] [UI90a] [Vil91] [Cha92] [Cha94]), fornece a infra-estrutura necessária para a construção e operação efetiva das aplicações distribuídas. Não se define uma metodologia de software específica para o desenvolvimento de aplicações distribuídas. O desenvolvimento deve se adaptar ao nível de serviços oferecidos em qualquer ambiente. A abrangência da infra-estrutura requerida varia, devido à escalabilidade na implementação das aplicações.

Observe que, as arquiteturas de rede e de gerenciamento necessárias para suportar um pequeno grupo de usuários em uma aplicação especializada, são bem diferentes daquelas requeridas para aplicações departamentais ou orientadas para a empresa.

A infra-estrutura para a computação distribuída não precisa basear-se necessariamente em sistemas abertos. Podem ser suportados elementos abertos ou proprietários [Gra91]. Porém, à medida que a complexidade e o número de aplicações construídas em uma infra-estrutura de computação distribuída for crescendo, então certamente os usuários vão estar em múltiplos tipos de sistemas (diferentes ambientes operacionais e plataformas de hardware). Nestas circunstâncias, é prudente o uso das interfaces o mais "abertas" possiveis ([HM91] [Ben92]).

Uma arquitetura de rede eficiente constitui a pedra fundamental sobre a qual a computação distribuída e as aplicações distribuídas terão sucesso ou então estarão fadadas ao fracasso [Vil91]. As empresas não podem construir aplicações distribuídas se suas redes não oferecem capacidade suficiente para transmissão de informações, não são confiáveis, ou não suportam todos os sistemas e aplicações de maneira apropriada.

O número de usuários e a complexidade das aplicações desempenham um papel fundamental na determinação do nível de funcionalidade da rede, e também no custo de implementação da arquitetura de rede distribuída. A figura 2.1 a seguir ilustra a relação entre o número de usuários e a complexidade das aplicações.

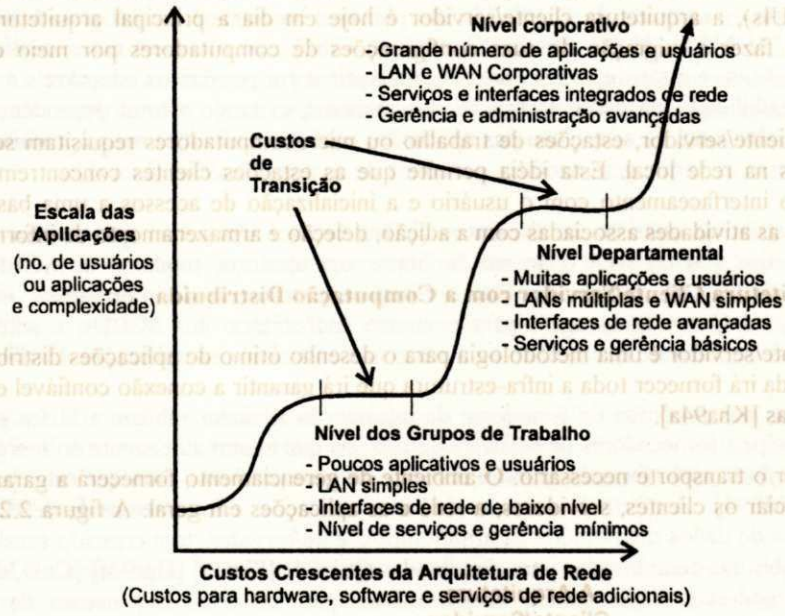


Figura 2.1. A Implementação da Arquitetura da Rede [Vil91]

Três níveis de implementação são facilmente vislumbrados ([Vil91] [HM91, [Gra91] [Cas93]): grupos de trabalho (workgroups); nível departamental; nível corporativo.

A *nível dos grupos de trabalho* (Workgroups), temos ambientes caracterizados por aplicações isoladas e com um número restrito de usuários (até 50 usuários), implementando-se uma topologia de rede simples em um ambiente homogêneo, onde não são necessários serviços avançados. As aplicações isoladas são desenvolvidas sob demanda para atender às necessidades urgentes de pequenos grupos de usuários.

No *nível departamental* encontramos de 50 a 300 usuários, com um número considerável de aplicações, caracterizados por uma nova arquitetura de rede baseada em cabeamento estruturado, arquiteturas físicas padrões, protocolos padrões, e possívelmente com interligação entre redes locais. Utilizam-se sistemas operacionais para redes avançados [Gun93], que fornecem os serviços básicos como segurança, diretórios ou correio-eletrônico. Buscam-se as interfaces "abertas". Passam a existir ferramentas para gerenciamento e desenvolvimento de aplicações mais sofisticadas ([DJ93c] [Bau94a] [Bau94c]).

No *nível corporativo* o número de usuários é considerável, geralmente acima de 300, e as aplicações são mais sofisticadas. Caracteriza-se pelo uso de redes locais e redes de longo alcance inteligentes, onde podemos encontrar uma melhor administração do ambiente distribuído, com a contabilização do uso de recursos nos vários segmentos e redes, uso de concentradores inteligentes, suporte a vários tipos de tráfegos de dados (transferência de arquivos, troca de mensagens e outros), uso de protocolos padrões e "gateways" para ambientes proprietários.

2.4.2. A Arquitetura Cliente/Servidor

A *Arquitetura Cliente/Servidor* (C/S) ([McL92a] [Sch92a] [Cas93] [Com93a] [Dew93] [Dew94] [Hay93d] [Hur93c] [SUN92a]), é um método usado para se projetar e implementar aplicações que estão funcionalmente separadas em processos distintos. Estes processos podem "rodar" no mesmo processador, ou então podem ser distribuídos em múltiplos processadores em uma rede. A localização das aplicações deve ser transparente para os usuários finais.

Em essência, a arquitetura cliente/servidor é uma metodologia para se construir aplicações distribuídas. Deste ponto de vista, estamos falando de software (de sistema ou de aplicação) residente nas estações clientes e servidoras.

A arquitetura cliente/servidor é uma das principais tecnologias desenvolvidas nos anos 90. Sustentada por um crescimento geométrico na capacidade de processamento dos micro-computadores de mesa (desktops), juntamente com um maior uso das redes locais, das técnicas de acesso a bases de dados, e das interfaces mais amigáveis de usuários (GUIs), a arquitetura cliente/servidor é hoje em dia a principal arquitetura escolhida por empresas que começam a fazer a migração de suas configurações de computadores por meio das soluções da computação distribuída.

Em um sistema cliente/servidor, estações de trabalho ou microcomputadores requisitam serviços a um ou mais servidores espalhados na rede local. Esta idéia permite que as estações clientes concentrem-se naquilo que sabem fazer melhor, i.e., o interfaceamento com o usuário e a inicialização de acessos a uma base de dados. Os servidores gerenciam todas as atividades associadas com a adição, deleção e armazenamento de informações.

2.4.3. A Relação da Arquitetura Cliente/Servidor com a Computação Distribuída

A arquitetura cliente/servidor é uma metodologia para o desenho ótimo de aplicações distribuídas, enquanto que a computação distribuída irá fornecer toda a infra-estrutura que irá garantir a conexão confiável e consistente das estações clientes e servidoras [Kha94a].

A rede irá fornecer o transporte necessário. O ambiente de gerenciamento fornecerá a garantia de se poder de maneira confiável gerenciar os clientes, servidores, a rede e as aplicações em geral. A figura 2.2 a seguir ilustra esta relação.

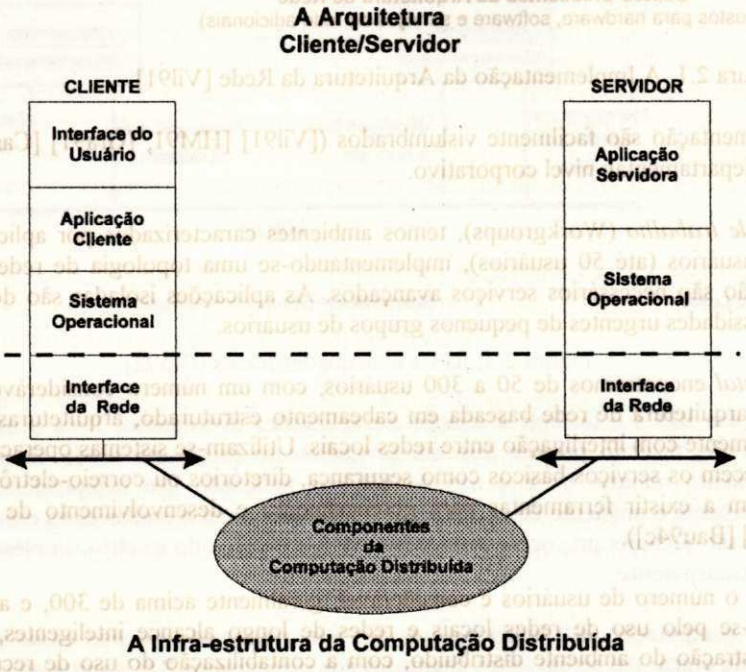


Figura 2.2. O Relacionamento da arquitetura cliente/servidor com a Computação Distribuída

2.5. Computação Distribuída - Principais Componentes

Apresentamos alguns dos componentes da computação distribuída, nesta seção, de maneira introdutória. Adiante, estes componentes serão abordados com maior riqueza de detalhes. Esta apresentação inicial visa despertar o leitor para os temas mais importantes dentro de uma perspectiva pragmática, com o intuito de responder à questão que ele deve estar fazendo a si mesmo: será que a computação distribuída fornece as soluções que estou procurando?

2.5.1. Redes Locais

As redes locais ([McL92a] [Red92c]) representam um componente vital para a implementação da computação distribuída, e fornecem o meio de comunicações entre os vários ambientes distribuídos. A previsão para 1995 (IDC) é que tenhamos 600.000 redes locais espalhadas pelo mundo. Para 1996, o Gartner Group estima um total de 2 milhões de redes locais no mundo, com 75 % de todos os micros empresariais conectados em redes [Red92b].

2.5.2. Chamadas a Procedimentos Remotos - RPCs

Uma chamada a procedimento remoto em uma linguagem de alto nível é um mecanismo para a transferência de controle e de dados de um programa principal para outro (uma sub-rotina) que esteja executando em um mesmo ambiente operacional (mesmo hardware e software). Este mesmo conceito pode ser empregado para transferir mensagens entre processos clientes e servidores através de uma rede. Neste último caso, o mecanismo recebe o nome de ***Chamada a Procedimento Remoto***, ou simplesmente **RPC** (*Remote Procedure Call*) ([Blo95] [Cla92b]). Assim, chamadas RPCs são chamadas a funções que estão fora do espaço de endereçamento corrente, porém parecem com chamadas a procedimentos locais como mostrado na figura xxx. As funcionalidades das RPCs são de vital importância para a implementação dos serviços e aplicações em ambientes da computação distribuída. Uma comparação entre uma chamada local e uma RPC está ilutsrada na figura 2.3.

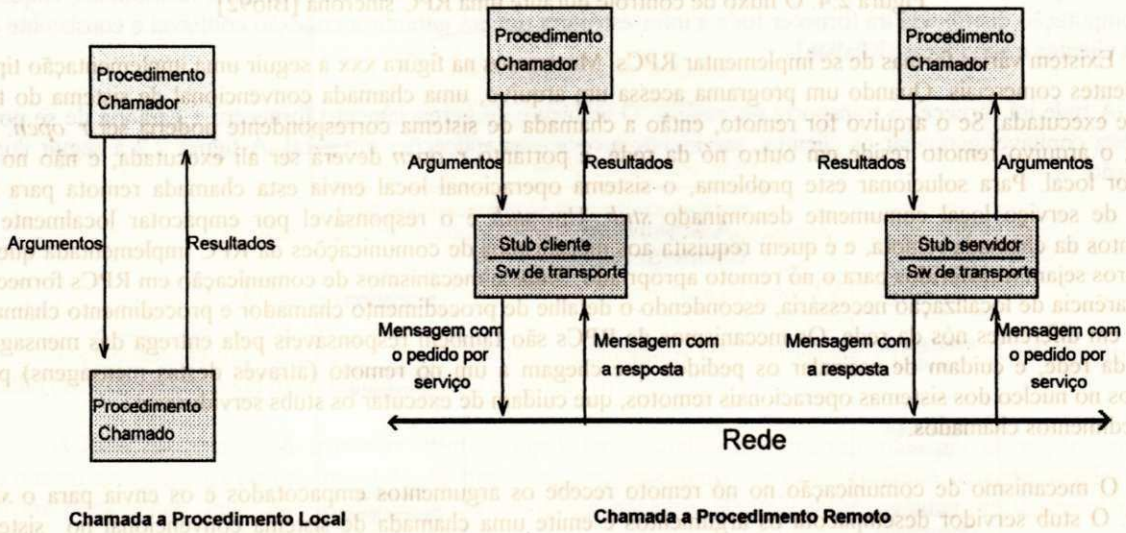


Figura 2.3. RPCs e chamadas locais [Blo92]

O principal objetivo das RPCs é facilitar a criação de aplicações distribuídas. Para o usuário, uma RPC esconde a complexidade da rede, fazendo parecer que um procedimento local está sendo chamado, quando na realidade este é um procedimento remoto. Com o uso de RPCs a aplicação fica independente da camada de transporte, de tal forma que somente o ambiente de computação distribuída precisa conhecer as especificidades desta camada. Neste caso, pode-se usar qualquer protocolo de transporte. A aplicação do usuário simplesmente inicializa o mecanismo RPC, o restante é transparente.

Uma configuração de computação distribuída faz com que a montagem de sistemas casualmente conectados tomem a forma de uma entidade única. Ao serem feitas chamadas aos mecanismos RPCs neste ambiente distribuído, o usuário pode enxergar todo o agregado de recursos como se estes estivessem em um único domínio. Assim, programar aplicações distribuídas torna-se uma atividade mais fácil.

RPCs são particularmente apropriadas para o modelo cliente/servidor. Por exemplo um servidor de banco de dados pode fornecer dados requisitados por vários usuários e aplicações através do processamento das RPCs emitidas pelas entidades clientes.

Quando uma RPC é executada, o ambiente chamador é suspenso depois que o procedimento chamador envia uma mensagem através da rede exatamente para o nó onde o procedimento chamado será executado. A figura 2.4 ilustra o conceito. Quando o procedimento chamado completa sua execução, os resultados gerados por ele são enviados de volta para o ambiente chamador, onde o procedimento chamador cuida de retomar a sua execução. Em termos do modelo cliente/servidor, dizemos que o procedimento chamador é o cliente, e o procedimento chamado é o servidor.

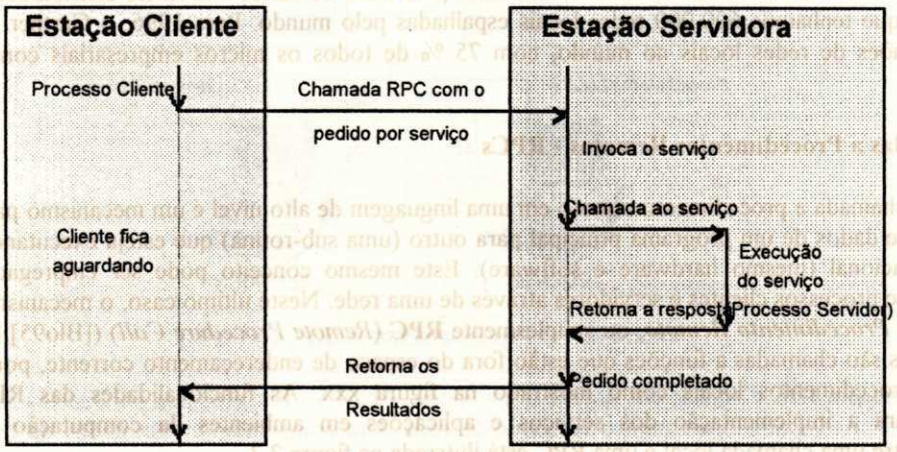


Figura 2.4. O fluxo de controle durante uma RPC síncrona [Blo92]

Existem várias formas de se implementar RPCs. Mostramos na figura xxx a seguir uma implementação típica de ambientes comerciais. Quando um programa acessa um arquivo, uma chamada convencional de sistema do tipo *l_open* é executada. Se o arquivo for remoto, então a chamada de sistema correspondente poderia ser *r_open*. No entanto, o arquivo remoto reside em outro nó da rede, e portanto *r_open* deverá ser ali executada, e não no nó chamador local. Para solucionar este problema, o sistema operacional local envia esta chamada remota para um módulo de serviço local comumente denominado *stub*. Um *stub* é o responsável por empacotar localmente os argumentos da chamada remota, e é quem requisita aos mecanismos de comunicações da RPC implementada que os parâmetros sejam transferidos para o nó remoto apropriado. *Stubs* e mecanismos de comunicação em RPCs fornecem a transparência de localização necessária, escondendo o detalhe de procedimento chamador e procedimento chamado estarem em diferentes nós da rede. Os mecanismos de RPCs são também responsáveis pela entrega das mensagens através da rede, e cuidam de assinalar os pedidos que chegam a um nó remoto (através destas mensagens) para processos no núcleo dos sistemas operacionais remotos, que cuidam de executar os stubs servidores e os procedimentos chamados.

O mecanismo de comunicação no nó remoto recebe os argumentos empacotados e os envia para o *stub* servidor. O stub servidor desempacota os argumentos e emite uma chamada de sistema convencional no sistema local, que invoca o procedimento correto (l_open) no servidor.

Em alguns sistemas os *stubs* são gerados automaticamente através do uso de interfaces especiais. Estas interfaces geram o código necessário para empacotar ou desempacotar argumentos e resultados. Cuidam também de inicializar os procedimentos corretos a partir das chamadas que chegam aos *stubs* servidores em nós remotos. Normalmente, uma *linguagem de definição de interface*, ou **IDL** *(Interface Definition Language)* é usada para especificar as informações necessárias para a interface. Estas informações podem incluir nomes de procedimentos, tipos de argumentos, ou resultados da execução de um procedimento remoto.

Os principais componentes do mecanismo de RPCs incluem: uma linguagem/compilador e facilidades "run-time", que implementam os mecanismos reais de chamadas representados por uma RPC, o que oferece transparência relativa aos protocolos de transporte mais baixos. A figura 2.5 ilustra um exemplo de implementação de mecanismos RPCs e define uma linguagem específica para a criação de RPCs.

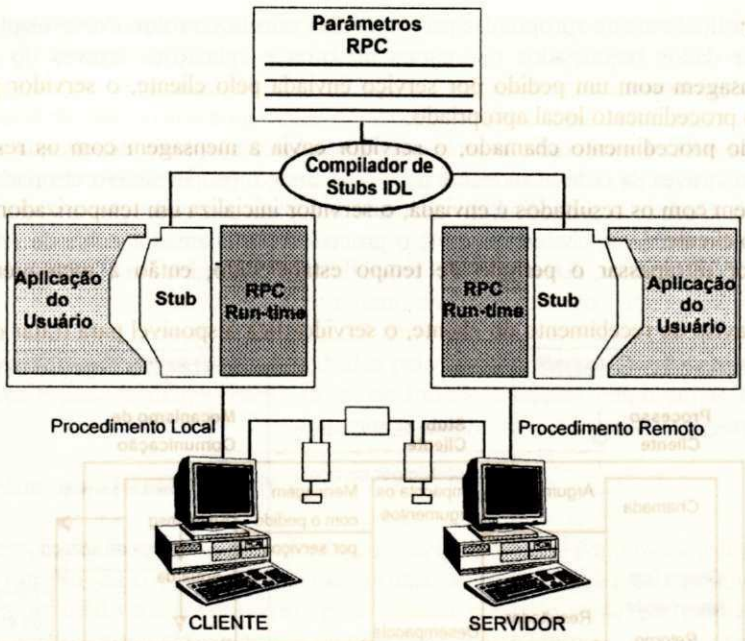


Figura 2.5. O Processamento de RPCs [Cha92]

A linguagem IDL é similar em sintaxe à linguagem ANSI C. Ela processa parâmetros de RPCs e fornece construções apropriadas adicionais para o ambiente da rede. Um compilador traduz dados da linguagem IDL em *stubs*. Para o procedimento local, um *stub* representa o processo servidor que se está tentando acessar. Similarmente, no lado do servidor, o procedimento remoto acredita que seu *stub* é o processo cliente que ele está tentando servir. Assim, os *stubs* são, eles mesmos, servidores para seus respectivos "clientes". Um *stub* faz para o programador o que de outra forma teria que ser feito manualmente: copiar argumentos de-para pacotes RPC, convertendo formatos de dados, se necessário, e invocando as facilidades run-time do mecanismo de RPCs.

Durante a execução da RPC o processo cliente é suspenso (adormece) e fica aguardando pelo retorno do resultado, como mostrado na figura 2.6. A natureza desta comunicação cliente/servidor é sempre síncrona, ou seja, o cliente fica aguardando até que o servidor envie uma resposta. Quando o procedimento local no servidor termina, o controle é devolvido para o *stub* servidor, e os resultados são enviados para o stub cliente, desempacotados, e passados para o procedimento chamador (antes, é claro, o processo cliente é acordado). O mecanismo de comunicação implementado (na camada de transporte) é responsável por questões como a retransmissão das mensagens, entregas de avisos de recebimento ("acknowledgments"), roteamento de mensagens, e criptografia. O processo cliente é responsável pela retransmissão da mensagem até que um aviso de recebimento seja retornado. No entanto, o próprio resultado da execução de uma RPC é suficiente para informar que a mensagem enviada com a chamada ao procedimento remoto foi recebida.

O fluxo de controle para a execução de uma RPC é mostrado a seguir para os processos clientes e servidores (estejam estes em nós remotos, ou mesmo locais).

No Processo Cliente

1. O cliente envia uma mensagem com um pedido de serviço mais seus argumentos.
2. Ao enviar esta mensagem, o cliente inicializa um temporizador, e fica aguardando pela mensagem com o resultado. O processo então adormece.
3. Se o temporizador ultrapassar o período de tempo estabelecido antes do recebimento da mensagem com o resultado, então o cliente reenvia esta mensagem.
4. Ao receber a mensagem enviada pelo servidor contendo os resultados, o processo cliente acorda e grava os argumentos retornados, enviando em seguida um aviso de recebimento para o servidor.

No Processo Servidor

1. Ao receber a mensagem com um pedido por serviço enviada pelo cliente, o servidor grava os argumentos de entrada e chama o procedimento local apropriado.
2. Após o término do procedimento chamado, o servidor envia a mensagem com os resultados de volta para o cliente.
3. Quando a mensagem com os resultados é enviada, o servidor inicializa um temporizador e aguarda por um aviso de recebimento do cliente.
4. Se o temporizador ultrapassar o período de tempo estabelecido, então a mensagem com os resultados é retransmitida.
5. Quando recebe o aviso de recebimento do cliente, o servidor fica disponível para tratar outros pedidos de outros clientes.

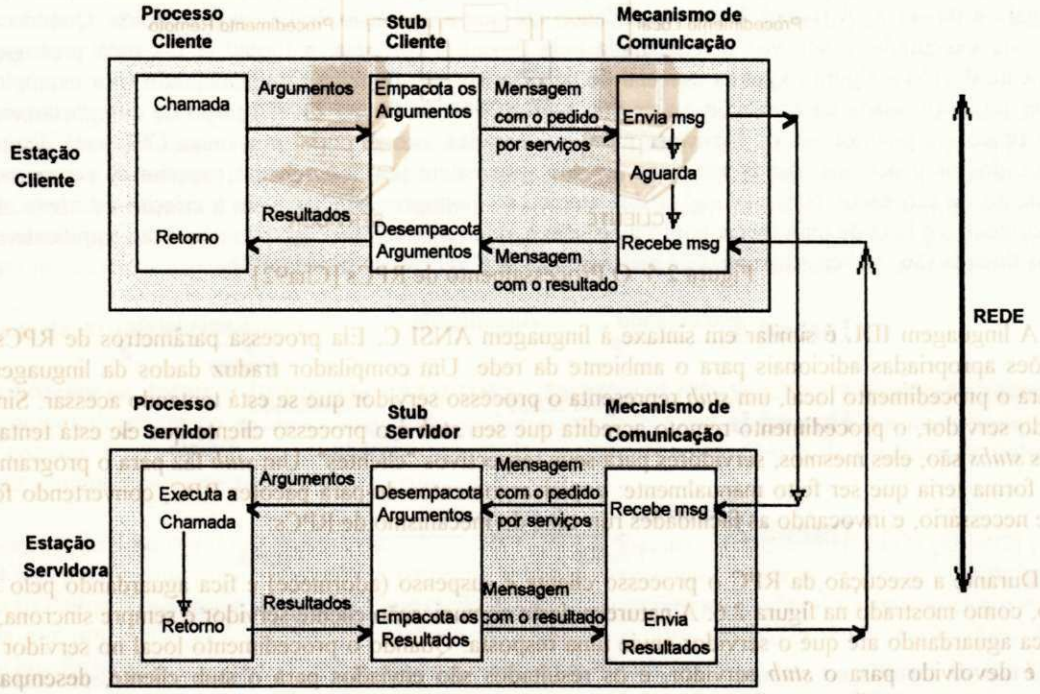


Figura 2.6. A Arquitetura Básica para o Mecanismo de Chamadas a Procedimentos Remotos [Cla92b]

RPCs e Threads

As RPCs precisam criar processos, e envolvem a troca de contextos em sistemas operacionais. Estes tipos de operações podem ser bem custosas em termos de "overhead" gerado. Uma maneira de se reduzirem estes custos é através da manutenção de um *pool* de processos inativos que podem tratar as mensagens que forem chegando, o que elimina o custo para a criação destes processos. Neste caso, quando um processo servidor termina sua tarefa, ele volta para um estado inativo, ao invés de morrer. Este é o tratamento convencional fornecido pelos sistemas operacionais monoprocessados.

Em ambientes multiprocessados o sistema operacional pode utilizar o conceito de threads. Threads são de fundamental importância para os ambientes distribuídos para facilitar as tarefas de controle de entrada/saída assíncrona, ou para atenderem a vários pedidos por serviços simultaneamente. Atualmente, vários sistemas operacionais disponíveis comercialmente implementam o suporte a execuções multi-threaded (por exemplo, a maioria das versões do Unix).

As principais funções implementadas pelos gerenciadores de processos nestes sistemas operacionais incluem: criação, escalonamento, sincronização, e gerencia simultânea de múltiplas threads em um único processo. Depois de criadas, as threads podem estar em quatro possíveis estados: aguardando, pronta, rodando (executando) ou concluída, como mostrado na figura 2.7.

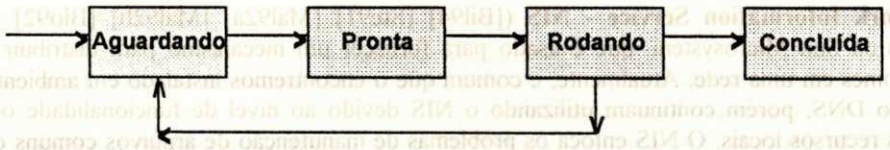


Figura 2.7. Estados de uma thread [Blo95]

Na figura 2.8 por exemplo, as threads de A a Z podem estar concorrendo pelo uso de tempo de CPU, onde a distribuição deste recurso valioso (o tempo de CPU) pode estar sendo controlado por um dos algoritmos de escalonamento preemptivos que estiverem disponíveis (por exemplo, first-in, first-out, round-robin, ou time-slicing com prioridades). Quando a thread A é executada ela efetua uma operação de leitura em um arquivo em disco, ficando neste momento bloqueada em um estado de "esperando pela operação de entrada/saída (E/S)", e passa o controle para a thread Z. A thread A fica aguardando até que a operação de E/S seja atendida. Quando a thread Z começa a ser executada, a sua vez é requisitada pela thread B (ou seja, a thread B faz uma preempção em Z) possivelmente devido a algum esquema especial de partilhamanento de tempo implementado (por exemplo, a thread B pode ter uma prioridade de execução maior que a prioridade da thread Z). Este tipo de comportamento para as threads é típico em plataformas de hardware monoprocessadas, isto é, onde uma única CPU está disponível para atender a todos os processos. Ainda assim, as threads apresentam um desempenho superior se comparadas com o gerenciamento de processos convencional nestes ambientes monoprocessados, pois a criação de novas threads e a troca de contextos é feita de uma forma muito mais suave, onde o "overhead" gerado não é tão significativo. Por este motivo, as threads são também chamadas de processos leves ("lightweigh processes").

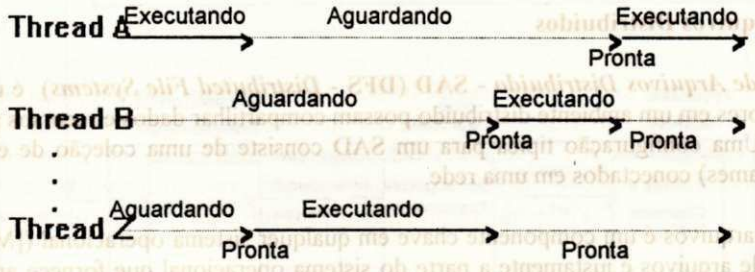


Figura 2.8. Threads em plataformas monoprocessadas [Blo95]

O uso de threads no entanto é altamente adequado para plataformas de hardware multiprocessadas (onde existem duas ou mais CPUs disponíveis e cooperantes para atenderem aos vários processos). Neste caso, tipicamente teríamos as três threads do exemplo anterior sendo atendidas simultaneamente por três processados distintos como mostrado na figura 2.9. Para que sejam implemantadas estas funcionalidades os sistemas operacionais devem disponibilizar facilidades de programação assíncronas (a maioria o faz) para que se possa implementar mecanismos RPCs não bloqueáveis, incluindo operações fork, multithreading e lightweigh processes; chamadas de serviço de E/S assíncronas, não bloqueáveis; ou facilidades de programação dirigida a eventos, como por exemplo sinalizadores, temporizadores, e primitivas X11, dentre outras.

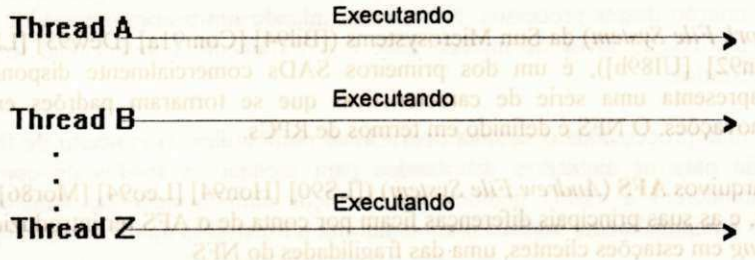


Figura 2.9. Threads em plataformas multiprocessadas [Blo95]

2.5.3. Serviços de Nomes e Diretórios Distribuídos

O ***Serviço de Diretórios*** (ou de ***Nomes***) é o componente da computação distribuída que permite que usuários e programas possam localizar e descrever pessoas, lugares, aplicações, e serviços em geral que fazem parte dos ambientes distribuídos. Mais formalmente, o Serviço de Diretórios é o componente que permite que aplicações e serviços localizem e façam referência a informações em um ambiente de computação distribuída. Os principais serviços de nomes e diretórios abordados em nosso livro são o NIS, o DNS e o X.500.

O **Network Information Service - NIS** ([Bil94] [Ste91] [Mal92a] [Mal92b] [Blo92] [Der92b]), é o serviço de nomes da Sun Microsystem, que é usado para fornecer um mecanismo para distribuir os serviços de manutenção de nomes em uma rede. Atualmente, é comum que o encontremos instalado em ambientes Internet que também utilizam o DNS, porém continuam utilizando o NIS devido ao nível de funcionalidade oferecido para a administração dos recursos locais. O NIS enfoca os problemas de manutenção de arquivos comuns de configuração em uma rede Unix [Ste91]. Arquivos de senhas, de grupos de usuários, e de nós na rede são mantidos e propagados para todos os nós na rede. O NIS é um sistema de banco de dados distribuído que substitui as cópias de arquivos de configuração comumente replicados por uma facilidade de administração centralizada [Mal92a].

O **Domain Name Service - DNS** ([Com91a] [Ste91] [Hal93b] [Mal92a] [Mal92b] [OT92] [Ros91] [Dav89] [Cha92]) fornece serviços de nomes e diretórios na Internet. Podemos considerá-lo como um sistema de banco de dados distribuído a nível de mundo, que traduz nomes de estações em endereços, fornecendo informações de roteamento, e informações sobre domínios ou outros dados. O DNS é essencial para cada sitio que se conecta diretamente à Internet. Porém, mesmo para redes locais isoladas que utilizem protocolos da pilha TCP/IP, sempre faz sentido que se use o DNS.

O padrão **X.500** ([Hal93b] [Gra91] [CTR92b] [Mal92b] [Bla91] [Cha92]) é uma das recomendações do ITU-T, e devido aos objetivos comuns, foi desenvolvido em parceria com a ISO. O sistema completo é conhecido simplesmente como o ***Diretório*** [Bla91]. Ele fornece serviços de diretórios para os processos de aplicações na pilha de protocolos OSI. A estrutura e os serviços associados com o Diretório foram definidas de tal forma que ele possa ser usado também para fornecer serviços para outras aplicações como processos de gerência em aplicações de gerenciamento de sistemas e agentes de transferência de mensagens (ATMs) no sistema MHS X.400 [VN90b].

2.5.4. Sistemas de Arquivos Distribuídos

Um ***Sistema de Arquivos Distribuído*** - **SAD** (**DFS** - ***Distributed File Systems***) é usado para permitir que usuários de computadores em um ambiente distribuído possam compartilhar dados e recursos por meio de um sistema de arquivos comum. Uma configuração típica para um SAD consiste de uma coleção de estações de trabalho (e, eventualmente, mainframes) conectados em uma rede.

O sistema de arquivos é um componente chave em qualquer sistema operacional ([Mil87] [Don92] [Tan92] [Mae87]). O sistema de arquivos é justamente a parte do sistema operacional que fornece armazenamento de longa duração para os dados. É o sistema de arquivos que cuida dos objetos armazenados, de tal forma que eles possam ser referenciados por um nome. O sistema de arquivos deve garantir que um arquivo e seu nome existam desde o momento em que forem criados, até o momento em que forem descartados.

Os SADs são implementados como parte do sistema operacional em cada computador conectado em uma rede (a nível de um processo de usuário, em sistemas operacionais com micro-núcleos). São bastante adequados para estruturas dispersas e descentralizadas, e preocupam-se principalmente em fornecer soluções para as questões de transparência, tolerância a falhas, e escalabilidade ([Cas93] [OSF91c]). Eles baseiam-se no modelo cliente/servidor [Ull93], onde um ou mais servidores de arquivos cooperam com clientes do sistema de arquivos, de tal forma que estes clientes possam acessar os arquivos gerenciados pelos servidores. Os SADs que abordamos em nosso livro incluem o NFS, o AFS, o DFS, o Sprite, e o CODA.

O **NFS** (*Network File System*) da Sun Microsystems ([Bil94] [Com91a] [Dew93] [LS90] [Mal92b] [Mil91] [Ste91] [SUN89] [Tan92] [UI89b]), é um dos primeiros SADs comercialmente disponíveis, e um dos mais conhecidos. O NFS apresenta uma série de características que se tornaram padrões entre os SADs, e que introduziram grandes inovações. O NFS é definido em termos de RPCs.

O sistema de arquivos **AFS** (*Andrew File System*) ([LS90] [Hon94] [Leo94] [Mor86]) possui muitos pontos em comum com o NFS, e as suas principais diferenças ficam por conta de o AFS ter introduzido no mercado suporte de alto nível para *caching* em estações clientes, uma das fragilidades do NFS.

O ***Distributed File System*** - **DFS** ([OSF90b] [OSF92c]) é o sistema de arquivos distribuído da infra-estrutura do DCE da OSF, e foi desenvolvido a partir do AFS, e portanto, os dois são muito parecidos. O DFS suporta invalidação de cache em cliente pelo uso de um mecanismo de fichas (tokens) de acesso [Hon94], onde um tipo especial de ficha fornece acesso para que clientes modifiquem dados em uma parte específica de um arquivo DFS. Com uma destas fichas em mãos, o cliente tem garantido o acesso exclusivo a esta parte do arquivo. Estas fichas de acesso permitem gravações concorrentes sem causar overhead.

O *Sprite* ([Wit91] [Hon94] [LS90] [Tan92] [TVR85]) é um sistema operacional desenvolvido na Universidade da California no início dos anos 80. Sua arquitetura prevê o uso de estações de trabalho sem disco, e com alta capacidade de armazenamento em memória real. O sistemas de arquivos Sprite merece destaque por ser inovador em duas áreas: nomes e caching.

O *sistema de arquivos distribuído CODA* [Hon94], em desenvolvimento na Universidade Carnegie-Mellon, também tem muito em comum com o AFS. Porém, o CODA adiciona duas características importantes ao AFS: total replicação de arquivos por vários servidores, e suporte a clientes desconectados.

2.5.5. Sistemas de Gerenciamento de Bancos de Dados Distribuídos

Um SGBD pode ser considerado como um dos maiores patrimônios de uma empresa. À medida que os processos de downsizing e descentralização se iniciam, o nível de acessos aos recursos do banco de dados por microcomputadores e estações de trabalho cresce rapidamente. A tendência é que aplicações dispersas, localizadas em estações clientes, passem a interagir com as bases de dados corporativas [Wat93b]. Estas bases de dados podem estar localizadas em servidores na rede, em sistemas de médio-porte ("midrange") como o AS/400 da IBM ou em máquinas VAX da Digital, ou ainda em mainframes atuando como grandes servidores [Hay93b].

A interação entre os usuários e os bancos de dados deve ocorrer de maneira uniforme. Por isto a importância da linguagem SQL (Structured Query Language), ou seja, fornecer um padrão de acesso às bases de dados relacionais e distribuídas. SQL foi criada em 1972. Seu objetivo inicial era criar uma linguagem de aplicação padrão para o acesso a bancos de dados relacionais. Na figura 2.10 a seguir podemos ver um exemplo de uma consulta SQL sendo processada.

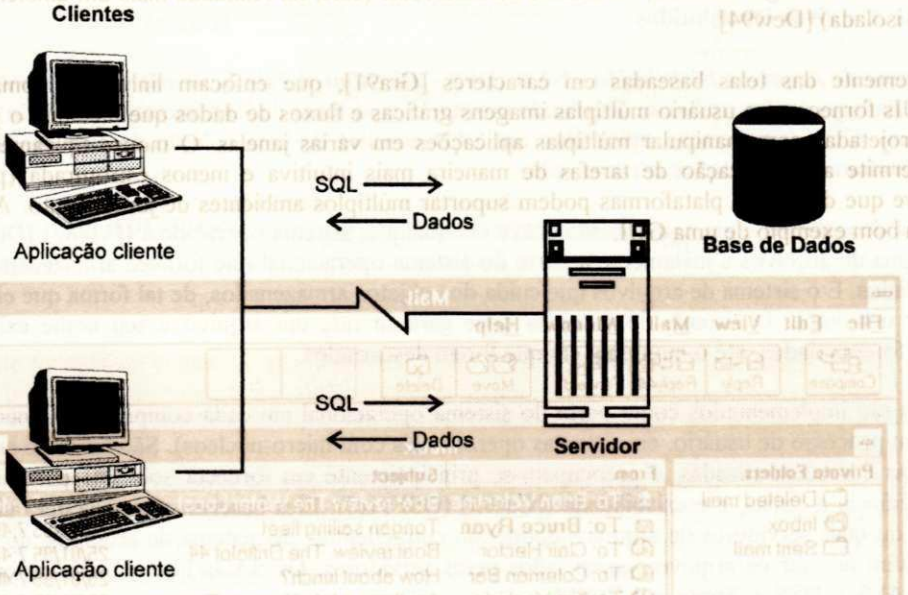


Figura 2.10. Consulta SQL para o modelo múltiplos clientes/único servidor [Cas93]

2.5.6. Sistemas de Correio Eletrônico

Os *sistemas de correio eletrônico* são cruciais para os negócios de muitas empresas. Inicialmente usados isoladamente, atualmente a comunicação via correio eletrônico entre empresas dispersas é algo trivial. Esta comunicação inclui mais do que a simples troca de mensagens entre amigos. Atualmente podem ser vistas aplicações de correio eletrônico para atender a pedidos de compras, ordens de pagamento, transferências bancárias, e outros tipos de trocas de documentos administrativos, que substituem os documentos equivalentes em papel.

Atualmente, os sistemas de correio eletrônico estão disponíveis para milhões de pessoas por todo o mundo, e mensagens são freqüentemente trocadas entre diferentes plataformas operacionais em diferentes empresas ([Eng94a] [ES93]).

Os tipos de aplicações que fazem melhor uso das funcionalidades oferecidas pelos sistemas de correio eletrônico são os programas aplicativos centrados em mensagens. Os principais aplicativos desta categoria são os programas de agendas e escalonamento de pessoal ([Sto93b] [Ham93]). Estes aplicativos são mais comumente conhecidos como *programas para produtividade de grupos de trabalho* ([Sch92a] [Rad93a] [Der92a] [Cas93]).

O terceiro serviço mais usado em uma rede fica atualmente por conta do correio-eletrônico. Só perde para o partilhamento de impressoras e para o acesso partilhado a mainframes (via gateways) [BG93].

O desejo de comunicação constitui a essência de uma rede. As pessoas sempre desejaram corresponder-se umas com as outras da maneira mais rápida possível. Os sistemas de correio eletrônico representam exatamente este desejo nas redes de computadores. Eles permitem que pessoas enviem e recebam mensagens sem terem que perder muito tempo preocupados em como a mensagem chegará realmente ao seu destino [Hay92].

Os dois grandes fornecedores de padrões para sistemas de correio eletrônico de longa distância são a ISO/ITU-T com o protocolo **X.400** ([HM91] [VN90b] [Bla91]) e a Internet com o protocolo **SMTP** (*Simple Mail Transfer Protocol*) ([Com93a] [Com91a] [BRI94b] [Dva89]).

2.5.7. A Interface Gráfica de Usuários (GUI)

Este é um componente inerente das redes inteligentes. A GUI (Graphical User Interface) ([Sey89] [Wat93a]) é a interface para estas redes.

Basicamente, as GUIs são imagens bit-mapped que atuam como "front-ends" para os sistemas operacionais. Sua funcionalidade inclui ícones para representar aplicações e recursos, menus do tipo "pull-down", caixas de diálogo e entidades escaláveis de janelamento que se sobrepõem.

Dos produtos mais importantes, podemos citar os seguintes: OSF/Motif (padrão "de Jure"); Open Look da SunSoft; Presentation Manager da IBM; Windows da Microsoft (este, na realidade mais um ambiente operacional do que uma GUI isolada) [Dew94].

Diferentemente das telas baseadas em caracteres [Gra91], que enfocam linhas de comandos e texto associado, as GUIs fornecem ao usuário múltiplas imagens gráficas e fluxos de dados que facilitam o processamento da tarefa. São projetadas para manipular múltiplas aplicações em várias janelas. O mouse tipicamente substitui o teclado - isto permite a inicialização de tarefas de maneira mais intuitiva e menos estruturada (para não dizer bitolada). Observe que diferentes plataformas podem suportar múltiplos ambientes de janelamento. A figura 2.11 a seguir mostra um bom exemplo de uma GUI.

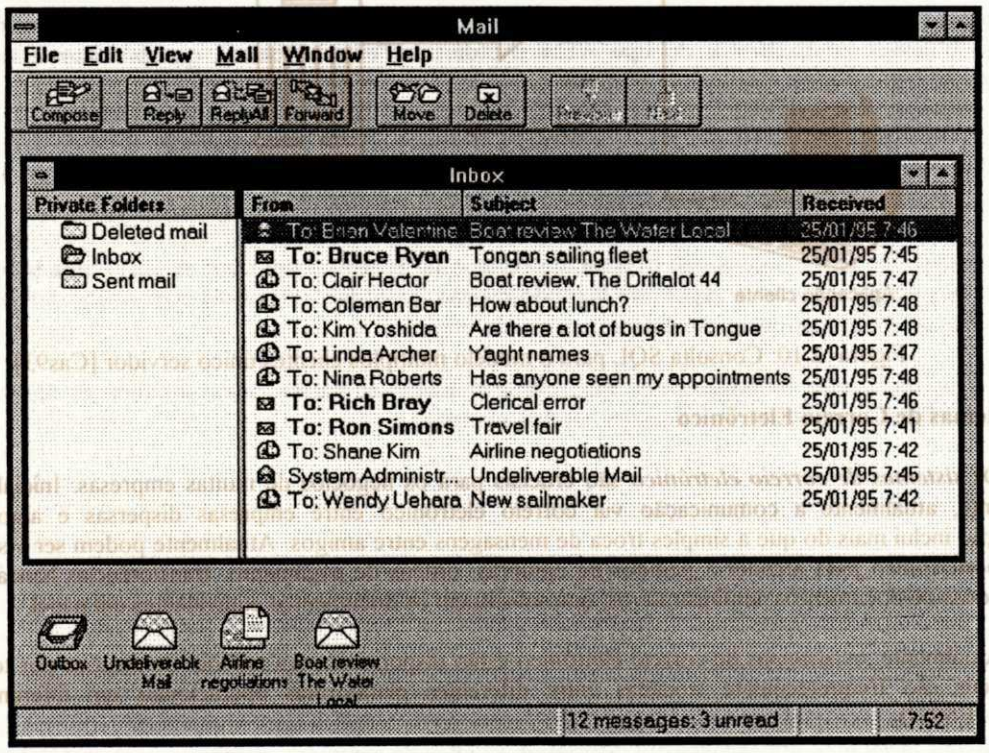


Figura 2.11. O cliente Microsoft Mail para Windows

2.5.8. Sistemas de Gerenciamento de Redes

O principal objetivo do gerenciamento de redes [Sta93b] é monitorar e controlar eventos, i.e., quando os recursos da rede tornam-se disponíveis ou indisponíveis, esta informação deve ser capturada e avaliada, e eventualmente, modificada.

A definição mais clássica da funcionalidade dos sistemas de gerenciamento de redes é encontrada no modelo CMIS/CMIP da ISO ([SOIEC89] [SOIEC90] [SOIEC91a] [SOIEC91b] [SOIEC91c] [SOIEC91d]) (Common Management Information Services/ Common Management Information Protocol) , que prevê:

- *Gerenciamento da configuração*: geralmente apresentado aos usuários em forma visual gráfica - indicando tipicamente o estado de cada um dos componentes da rede;
- *Gerenciamento de falhas*: a detecção e correção de falhas em qualquer parte da rede é executada a este nível. Isto inclui de interfaces de hardware aos protocolos suportados. Quando a correção não for imediatamente possível, o isolamento do problema pode ser requisitado;
- *Gerenciamento do desempenho*: ajuda ao pessoal da rede monitorar, modificar e controlar o uso de recursos, incluindo a vazão (throughput), o tempo de resposta médio, e o fluxo de dados como um todo;
- *Gerenciamento da segurança*: esta característica está associada com a integridade dos dados na rede. É uma área fraca atualmente, principalmente em arquiteturas distribuídas. Os aspectos de segurança ficam melhor definidos ao nível de sistemas operacionais, SGBDs e interfaces de usuários;
- *Gerenciamento da contabilização:* mede o uso médio e os custos da rede, podendo cobrar dos usuários correspondentes pelo uso dos recursos.

2.5.9. Sistemas de Processamento de Transações Em-Linha (Sistemas OLTP)

Os sistemas OLTP (On-Line Transaction Processing) ([Cla92b] [Mar93a]), permitem o desenvolvimento de aplicações tipicamente associadas com tarefas críticas: aplicações que implementam tarefas que dependem de respostas rápidas para um grande número de usuários; transações que envolvem acessos freqüentes a bases de dados; transações que necessitam de integridade de dados. Por exemplo: o ambiente de automação bancária - principalmente quando utiliza-se equipamentos do tipo ATM (Automatic Teller Machine), que precisam estar no ar 7 dias na semana, 24 horas por dia.

As transações convencionais diferem das transações OLTP com relação ao grau de garantia de integridade de dados, recuperabilidade, controle e desempenho. As principais características dos sistemas OLTP são as seguintes (chamadas características ACID [Cas93]):

- *Integridade:* as transações que alteram um recurso no sistema, como por exemplo uma base de dados, devem fazê-lo de acordo com a regra "ou-faz-tudo-ou-nao-faz-nada", i.e., se uma transação for abortada, a base de dados deve retornar imediatamente ao seu estado original.
- *Consistência*: uma transação deve sempre migrar a base de dados para o próximo estado válido. Não podem existir brechas enquanto modificações parciais ocorrem.
- *Isolamento:* uma transação só deve impactar seu próprio ambiente. Múltiplas transações são transparentes umas para as outras, e não podem afetar as outras transações.
- *Durabilidade:* o tratamento com sucesso executado pelas transações gera modificações permanentes nos dados. Falhas subseqüentes não podem reverter os processos anteriores.

2.5.10. Segurança

As facilidades da computação distribuída, incluindo o acesso global aos dados, transferência de arquivos e informações facilitadas, armazenamento disperso dos dados da empresa, geram enormes problemas de segurança. As soluções de tecnologia na área de segurança são normalmente diferentes de empresa para empresa. Não existem padrões [Far94b].

Um dos aspectos mais importante dos ambientes de computação distribuída é a sua ***segurança.*** Disponibilizar mecanismos de segurança em ambientes distribuídos é muito diferente de se fornecer segurança em outros ambientes. A diferença chave fica por conta de nos ambientes mais tradicionais ser mais fácil definir-se o perímetro de proteção aos recursos, o que é bem mais complicado em ambientes distribuídos, devido à própria estrutura da rede que os suporta [Sch94].

A maioria dos problemas de segurança começam a surgir quando um ambiente tradicionalmente centralizado começa a se tornar aberto por meio do estabelecimento de conexões em redes com o mundo exterior [CB94]. Nas inter-redes, existem conexões com outras redes fora do controle do domínio local, o que aumenta consideravelmente as de ameaças para estas redes [CB94]. Portanto, a segurança das redes é imprescindível para a confiabilidade e privacidade das informações armazenadas nos computadores hospedeiros.

Algumas precauções são essenciais para a garantia da segurança em ambientes da computação distribuída ([Kay94a] [Sch94] [OSF92d]):

- *Consciência*: deve-se estabelecer uma política organizacional para acesso aos computadores, às informações, e para a distribuição de responsabilidades. Os usuários devem estar conscientes de sua participação na proteção das informações da organização.
- *Acesso*: precauções extras devem ser tomadas com relação às conexões externas.
- *Autenticação*: as senhas devem ser trocadas regularmente. Se possível devem ser utilizados *passes* ou geradores aleatórios de senhas.
- *Confidencialidade*: os dados a serem transmitidos pela rede devem ser criptografados. Informações armazenadas em disco que sejam sensiveis (por exemplo, arquivos de senhas) ou críticas (por exemplo, arquivos de configuração) devem ser criptografadas.
- *Administração*: deve-se estar apto a administrar a segurança a partir de qualquer ponto na rede.
- *Disponibilidade*: todos os servidores devem ser equipados com equipamentos do tipo no-breaks.

Algumas das tecnologias empregadas em sistemas de segurança envolvem a criptografia, a administração de chaves de acesso, o uso de portas corta-fogo ("firewalls") e o sistema de autenticação Kerberos. A busca por mecanismos de segurança deve ser executada nos níveis de aplicações, bancos de dados, redes e do próprio sistema operacional.

2.5.11. Objetos Distribuídos

No centro da computação distribuída estão os objetos distribuídos, que vão além das barreiras tradicionais impostas por linguagens de programação, espaço de endereçamento de processos, sistemas operacionais ou plataformas de hardware. A tecnologia de orientação a objetos irá fornecer muitos dos meios que irão disponibilizar os sistemas distribuídos de fato. Cada uma destas barreiras envolve considerações com alto grau de complexidade, e o uso de objetos é a melhor forma de se abstrair e lidar com esta complexidade. Os objetos fornecem uma forma interessante para se organizar a complexidade nos modernos sistemas operacionais, e simplificam o processo de distribuição das aplicações. Com sua combinação natural de dados e comportamento e a separação explícita da interface de sua implementação, os objetos formam uma solução ótima para a distribuição de dados e processos.

Os objetos distribuídos são organizados em uma arquitetura de três camadas [Bet94], incluindo um Modelo de Objetos, um Modelo de Componentes e as Aplicações e Sistemas.

Os **Modelos de Objetos**, incluem o ***SOM/DSOM*** *(Distributed System Object Model)* da IBM [Ude93b], *o* ***COM*** *(Component Object Model)* da Microsoft [Bet94] *ou o* ***CORBA*** *(Commom Object Request Broker Architecture)* da OMG (Object Management Group) [Sol94]. Estes modelos são definidos para servir como um meio de ligação entre as implementações de objetos distribuídos.

Os **Modelos de Componentes** incluem o *OLE da Microsoft, o OpenDoc da Apple, e o OpenStep da Next, ou o CORBA CF do consórcio CORBA* ([Bet94] [Sol94] [Dew93] [Dew94]). Estes componentes baseiam-se nos modelos de objetos da camada inferior, de tal forma que possam implementar funções como a ligação e aninhamento, a ativação local, e outras relacionadas com objetos (*scripting, drag-and-drop*).

As **Aplicações e Sistemas** representam os módulos desenvolvidos pelos usuários finais, através das facilidades fornecidas pelas ferramentas disponíveis nas camadas inferiores.

CORBA (Common Object Request Broker Architecture) ([Sol94] [Dew93] [Dew94] [Bet94] [Pyl94]). representa uma arquitetura voltada para fornecer interoperabilidade a nível da camada de aplicação entre os sistemas que estiverem executando em plataformas com diferentes sistemas operacionais, linguagens, protocolos de redes, e arquiteturas de hardware. O serviço básico fornecido pelo CORBA é a entrega de mensagens de um processo para outro, e a entrega de uma resposta para o processo chamador. Atualmente, muitas empresas de grande relevância para a indústria de Informática, incluindo a IBM, a Hewlett-Packard, Olivetti, NEC, Sun Microsystems e muitas outras, e também vários consórcios da indústria, incluindo a OSF, International Multimedia Association, X/Open, etc., têm expressado seu endosso ao padrão CORBA, e estão incluindo implementações da API CORBA em seus produtos [Bet94].

O modelo de objetos *System Object Model* (**SOM**) ([Ude93b] [Bet94]) da IBM é um padrão binário para objetos que são neutros com relação aos sistemas operacionais e às linguagens de programação, e cujas interfaces estão de acordo com as definições do CORBA expressas por sua IDL. A versão distribuída do SOM, chamada **DSOM** (*Distributed System Object Model*) ([Ude93b] [Bet94]) compõem uma infra-estrutura aderente ao CORBA.

O *Component Object Model* (**COM**) [Bet94] da Microsoft executa algumas das mesmas tarefas de um servidor de objetos como o CORBA, só que em escala diferente, e com o uso de diferentes técnicas. O COM baseia-se em objetos de janelas, que definem entidades funcionais que obedecem aos princípios OO de encapsulamento.

2.6. Os Produtos para a Computação Distribuída

Duas plataformas para o desenvolvimento de aplicações distribuídas estão comercialmente disponíveis no mercado. Estas plataformas são na realidade um conjunto de tecnologias que formam um ambiente de computação distribuída. Falamos um pouco sobre o Open Network Computing da empresa americana Sun Microsystems e também do Distributed Computing Environment da OSF.

2.6.1. ONC da Sun

A SunSoft, empresa coligada à Sun Microsystems, é a responsável pelo desenvolvimento e comercialização do **ONC** (*Open Network Computing*) ([Mil91] [SUN89]). A versão mais recente chama-se ONC+ [Bil94]: um pacote que fornece um conjunto de serviços de computação distribuída corporativa, como: NFS multi-threaded; serviços de diretórios X.500; Transport Independent RPC (TI-RPC) para facilitar o desenvolvimento de aplicações distribuídas; e, Kerberos do MIT para segurança (autenticação).

Todas estas características fazem parte do ambiente operacional Solaris 2.X [Mal92a], que suporta computação heterogénea distribuída. O objetivo do ONC é fornecer blocos de construção para que programadores desenvolvam e implementem aplicações distribuídas. São também fornecidas ferramentas para administração de redes.

O código fonte para ONC (e ONC+) pode ser licenciado pela SunSoft, para permitir a migração para várias plataformas. Existem hoje em dia 100 implementações do ONC no mercado, funcionando em ambientes como: DOS, Microsoft Windows, OS/2, Macintosh, Unix System V release 4.2, AIX, VM/IBM, Ultrix/DEC, HP/UX, NetWare, dentre outros. No mundo, aproximadamente 3 milhões de sistemas usam o ONC.

Principais componentes do ONC+

A seguir falamos um pouco sobre cada um dos componentes do ONC+. As explicações são baseadas em ([Cas93] [Dew93], Dew94] [Bil94]):

- *Transport Independent Remote Procedure Calls (TI-RPC)*: fornece independência de transporte para módulos run-time, permitindo que uma versão binária única de um programa de aplicação distribuída rode em múltiplas camadas de transporte (TCP/IP, SPX/IPX, NetBios,etc). Nenhuma modificação é requisitada na aplicação binária se novos protocolos de transporte forem adicionados ao sistema no futuro. Observação: alguns geradores de código para aplicações distribuídas exigem que os programadores selecionem um protocolo de rede específico no momento da geração do código ou da compilação. Isto amarra para sempre a aplicação binária a um protocolo de transporte específico.

- *eXternal Data Representation (XDR):* fornece um método independente de arquitetura para representar os dados, solucionando diferenças na ordenação dos bytes, tamanho dos tipos de dados, representação e alinhamento. Aplicações utilizando XDR podem permutar dados entre plataformas de hardware heterogéneas.
- *Transport Layer Interface (TLI):* é a camada de comunicação que torna as RPCs independentes de protocolos, permitindo que programas RPC rodem em múltiplos protocolos de transporte na rede (por exemplo: TCP/IP, OSI, NetBios, etc).
- *Compilador RPCGEN*: para ajudar no desenvolvimento de aplicações baseadas em RPCs. Os programadores descrevem os relacionamentos entre os programas clientes e as rotinas servidoras, em uma linguagem de alto nível (com sintaxe similar ao ANSI C), chamada linguagem RPC. Os segmentos cliente e servidor são automaticamente gerados pelo RPCGEN.

2.6.2. O Ambiente de Computação Distribuída - DCE da OSF

O **DCE** (*Distributed Computing Environment*) ([Cha92] [Cha94] [Mas92] [Mil91] [Mil94c] [OSF90b] [OSF92c]) da OSF (Open Software Foundation) é um conjunto de tecnologias fortemente integradas que permite que fornecedores e usuários forneçam computação transparente em ambientes heterogêneos. Isto incluindo serviços requeridos para o desenvolvimento e manutenção de aplicações distribuídas. O DCE contem dois conjuntos de serviços:

- *Serviços Distribuídos Fundamentais:* que fornecem ferramentas para desenvolvedores de software criarem serviços requisitados pelos usuários em um ambiente de computação distribuída.
- *Serviços de Partilhamento de Dados:* fornece serviços de arquivos distribuídos, e suporte para arquivos MS-DOS e programas de controle de impressoras (Spooler).

O DCE é projetado para ser portado para sistemas operacionais que forneçam serviços similares, incluindo: OSF/1, Unix System V, e sistemas operacionais não Unix como OS/2 ou VMS/DEC. Os componentes do DCE estão mostrados na figura 2.12 apresentada a seguir.

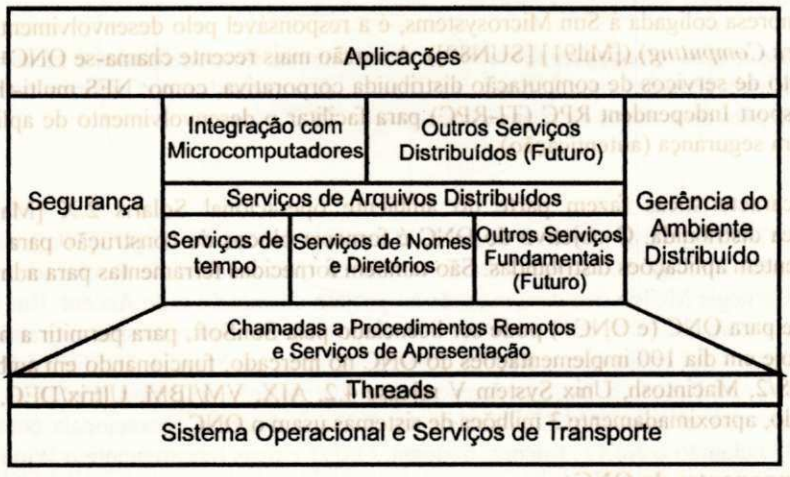


Figura 2.12. A Arquitetura do DCE [OSF92c]

Os Componentes do DCE

- *RPC e Serviços de Apresentação:* linguagens para definição de interfaces e RPCs fornecem a habilidade para que programadores possam transferir comandos e dados pela rede, de maneira transparente para os usuários, o que ajuda a esconder a complexidade da rede;
- *Naming:* nomes orientados para os usuários especificando computadores, arquivos, e pessoas que podem ser acessados facilmente em um ambiente distribuído. Também conhecido como serviço de diretórios, deve ser padrão em aparência e regras para todos os clientes;
- *Segurança*: as aplicações e serviços distribuídos devem estar aptos a identificar usuários, controles de acesso a recursos, e garantir a integridade de todas as aplicações;

- *Threads*: representam um método para suportar a execução paralela, pela administração de múltiplas threads (linhas de controle) dentro de um processo trabalhando em um ambiente distribuído;

- *Serviços de Tempo e Sincronização*: sincroniza os relógios de todos os sistemas em ambientes distribuídos, de forma que as aplicações possam operar corretamente;

- *Sistema de Arquivos Distribuídos*: estende o sistema local de arquivos para a rede, de forma a permitir que os usuários tenham acesso total a arquivos em configurações remotas;

- *Integração com Computadores Pessoais:* permite que microcomputadores utilizando MS-DOS acessem arquivos e serviços de impressão fora do ambiente DOS;

2.7. Sistemas Operacionais Distribuídos

Um ***Sistema Operacional Distribuído*** ([Cra93] [TVR85] [Wit91] [Bal89] [Cha90] [Mul90] [Tan92]) é um sistema operacional que enxerga os usuários da mesma forma que um sistema operacional centralizado tradicional o faria, porém, roda em múltiplas unidades centrais de processamento (UCPs) independentes.

O principal conceito envolvido no projeto destes sistemas é o de *transparência* ([TVR85] [Tan92]). Isto é, o uso dos múltiplos processadores fica invisivel para os usuários, que enxergam o sistema como um "único processador virtual", e não como um conjunto de máquinas distintas.

Vários sistemas operacionais distribuídos foram desenvolvidos nos últimos anos, e falamos de alguns deles rapidamente. Porém, o nosso intuito ao falarmos de sistemas operacionais distribuídos é o fato de muitas das tecnologias desenvolvidas na busca por estes sistemas terem sido essenciais na determinação dos elementos chaves para a computação distribuída. Muitos produtos que começam a ser ofertados comercialmente originaram-se destes projetos [Wit91].

Os sistemas operacionais distribuídos começaram a ser desenvolvidos no início dos anos 80, principalmente em projetos de laboratórios, universidades e instituições de pesquisas. Estes esforços iniciais foram determinantes para caracterizar a computação distribuída [BC91].

Fazemos a seguir uma apreciação, discutindo características, funcionalidades e tecnologias mais marcantes, de cada um dos principais sistemas operacionais distribuídos desenvolvidos nos últimos anos, os quais foram de primordial importância para os avanços da própria computação distribuída.

2.7.1. MACH

As raízes do MACH ([Bar85] [Bla90] [Tan92] [Tan89] [Wit91] [BC91] [OSF90a]) surgiram de um sistema chamado RIG (Rochester Intelligent Gateway) desenvolvido na Universidade de Rochester a partir de 1975. Em 1979 a Universidade Carnegie Mellon deu continuidade ao projeto chamando-o de Accent. Em 1984, foi fornecida ao Accent, compatibilidade com o UNIX, com a intenção de disponibilizar para o mesmo o grande volume de software pronto para este ambiente - foi então que o sistema passou a chamar-se MACH. [OSF91b] Desde então, a agência DARPA (Defense Advanced Research Projects Agency) do departamento de defesa (DoD) do governo dos Estados Unidos é a grande responsável pelo MACH. Atualmente diversos sistemas operacionais comerciais tiveram a sua origem com o MACH, incluindo o NeXT, Encore, Sequent, OSF/1 e mais recentemente o Windows NT da Microsoft [Cus93].

MACH é um sistema operacional voltado para plataformas multiprocessadas, e já na sua primeira versão em 1986, ele suportava uma máquina VAX multiprocessada com 4 UCPs. A versão atual, MACH 3.0, definiu um sistema operacional distribuído, baseado em um micro-núcleo que permite a emulação de UNIX e outros sistemas operacionais. A figura 2.13 a seguir mostra a arquitetura do MACH, juntamente com sua camada de emulação de sw, que inclui dentre outros emuladores, uma camada de abstração de hardware (HAL) para o uso com máquinas multiprocessadas.

No modelo da figura 2.13, a divisão de tarefas entre o núcleo e os vários sistemas operacionais modificados para rodar em modo emulado, a nível de processos de usuários, torna mais simples e fácil a manutenção das partes separadas. De certa forma, esta divisão é uma idéia remanescente da divisão de trabalho do sistema operacional VM/370 para "mainframes" (computadores de grande porte) IBM.

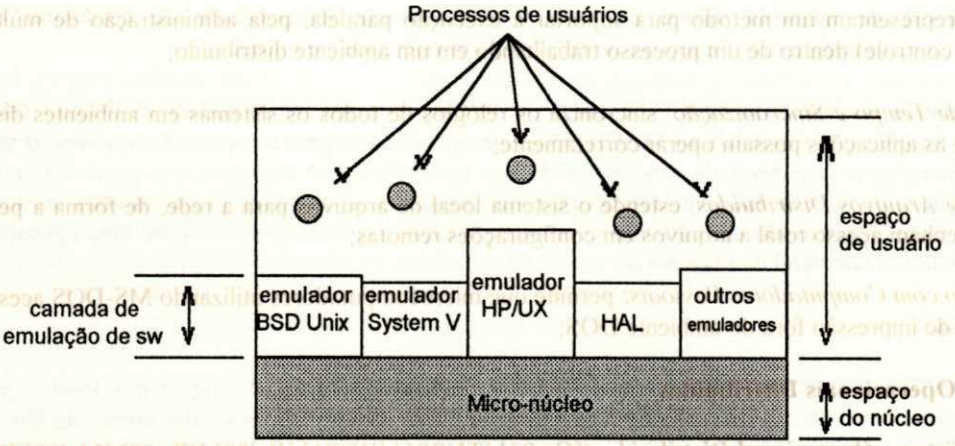


Figura 2.13. O modelo abstrato de emulação do Unix no MACH [Bla90]

O micro-núcleo do MACH [Var94] fornece os serviços essenciais de gerenciamento de processos, memória, e comunicações. Outras funções tradicionais em sistemas operacionais são fornecidas por processos a nível de usuários. O micro-núcleo do MACH, mostrado na figura 2.14 a seguir, constitui sem dúvida nenhuma o melhor exemplo de uma arquitetura de sistema operacional distribuído já implementada [Wit91].

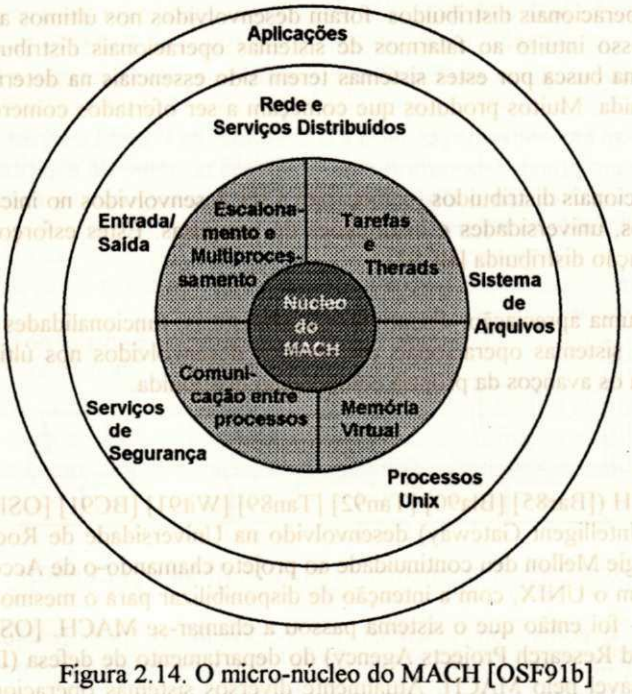


Figura 2.14. O micro-núcleo do MACH [OSF91b]

As principais características do MACH incluem o suporte para o multiprocessamento simétrico: quando uma máquina disponibilizar mais de um processador em sua arquitetura, podemos fazer uso desta funcionalidade por meio do mecanismo de multiprocessamento simétrico, que é capaz de atribuir a cada processador uma linha de controle ("thread") de um programa em execução. Mecanismos de bloqueio finamente ajustados garantem que os processadores disponibilizam a maior quantidade de informações possíveis nas threads, permitindo que os outros processadores acessem estas informações. Um programa multi-threaded, apesar de ser escrito normalmente pelo programador (como se o estivesse escrevendo para uma máquina monoprocessada) pode aproveitar-se dos processadores adicionais disponíveis.

Outra característica importante é a facilidade de portabilidade oferecida. Os aspectos dependentes e independentes de máquina são claramente separados no MACH. Assim portar o MACH para uma nova plataforma de hardware é relativamente simples, pois somente uma pequena parte do código precisa ser reescrita para o novo hardware.

A gerência de memória virtual do MACH é totalmente independente de hardware, com muitas funcionalidades de vanguarda como: "copy-on-write", "map-on-reference", e outras.

Uma grande tendência do MACH, é que, com o tempo, o seu micro-núcleo vá ficando cada vez mais especializado, e funções não essenciais ao MACH poderiam ser passadas para o espaço de usuários, o que facilitaria depurações, e incrementaria o nível de funcionalidade das aplicações. Por exemplo, um gerenciador de memória escrito por um usuário, poderia encriptar/desencriptar automaticamente os dados gravados/lidos de um sistema de arquivos, o que poderia ser usado em conjunto com uma aplicação especial (por exemplo, uma aplicação bancária) que necessitasse de um alto nível de segurança.

2.7.2. AMOEBA

O *AMOEBA* ([Tan92] [TVR85] [TVR90] [Wit91] [Mul90]) teve sua origem em 1984 a partir de um projeto de pesquisa coordenado pelo professor Tannembaum e três de seus alunos de doutorado na Universidade de Vrieje, na Holanda. A versão atual do Amoeba é a versão 5.0.

O sistema operacional Amoeba foi desenvolvido a partir de idéias novas, incluindo características bem peculiares. Para evitar que fosse necessário se reescrever software de aplicação disponível no mundo UNIX, o Amoeba passou a fornecer um pacote para emulação do UNIX.

Uma distinção interessante entre o Amoeba e os outros sistemas operacionais distribuídos é que no Amoeba não existe o conceito de "máquina hospedeira". É utilizado um interessante esquema de uso de recursos distribuídos, de forma totalmente transparente para o usuário, que permite que quando um usuário se conecte a uma máquina ele passe a enxergar o sistema como um todo, e não somente a sua máquina. Todos os programas de um usuário rodam em máquinas arbitrárias, e estas máquinas são escolhidas levando-se em consideração a sua carga de uso.

O Amoeba é típico em ambientes de pesquisa e acadêmicos na Europa e Japão. Não existem implementações comerciais do mesmo. Ele é um grande laboratório para pesquisas no ramo de algoritmos distribuídos e paralelos, e suas características mais importantes são o alto grau de eficiência conseguido na execução de serviços [Wit91]. A arquitetura do Amoeba está definida na Figura 2.15.

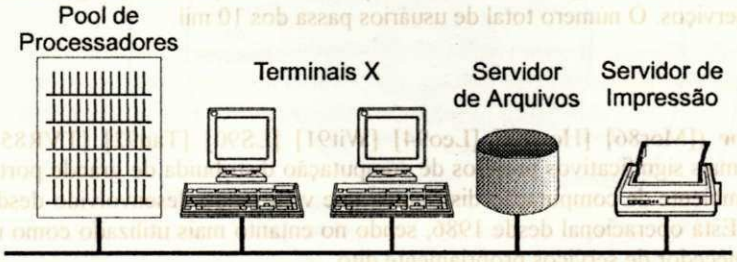


Figura 2.15. A arquitetura do Amoeba ([Mul90] [TVR90])

Neste modelo todo o poder computacional fica localizado no "pool (conjunto) de processadores", que representa os processadores e memória dos nós em uma rede. Portanto, as máquinas no pool podem ser de diferentes arquiteturas. O Amoeba foi projetado para tratar múltiplas arquiteturas e sistemas heterogêneos. O núcleo do Amoeba é muito simples e pequeno, com poucas chamadas ao sistema operacional.

Sistema	No.de Chamadas
Amoeba	30
Unix V 7	45
4.2 BSD	84
MACH	153
SunOS	165

Tabela 2.1. Número de chamadas ao sistema operacional [Tan92]

São suportados dois mecanismos para comunicações no Amoeba: RPCs e comunicações confiáveis entre grupos de trabalhos. O servidor de arquivos anda junto com os servidores de replicação e de tolerância a falhas, o que fornece uma funcionalidade extremamente interessante ao mesmo.

2.7.3. ATHENA

O ***Projeto Athena*** ([Gee94] [Wit91] [Cha90] [LS90] [Tan92] [TVR85), do Instituto de tecnologia de Massachussets (MIT), foi desenvolvido no período de 1983 a 1991, e está operacional desde então, fornecendo serviços sofisticados de sistemas operacionais distribuídos nesta universidade americana.

O projeto Athena iniciou-se com o intuito de fazer proveito do grande e disperso poder computacional disponível nos departamentos do MIT. Nos anos 80 a computação escapou das paredes dos CPDs e começou a proliferar pelos vários departamentos da universidade, na forma de estações de trabalho, computadores pessoais e redes locais. A constatação: "proliferação sem controle é um desastre" [Cha90], fez com que grandes somas de recursos de seu orçamento fossem dirigidos para controlar a proliferação destes recursos computacionais.

O ambiente para o Athena é projetado com interessantes características de tolerância a falhas, tornando-o imune a falhas previsíveis de conexões ou operação nos seus serviços. Todos os aspectos de gerência do sistema são tratados remotamente a partir de um serviço especial de gerenciamento centralizado de sistemas. O ambiente do Athena já foi implementado em várias universidades americanas, e mesmo em CPDs comerciais. A Digital Equipment Corporation fornece uma versão comercial para este sistema, chamado DECathena.

O Athena fornece uma interface uniforme para os serviços disponíveis para todos os computadores a ele conectados, por meio do protocolo de interface X/Windows [HM91], um padrão "de facto" para a indústria.

A nível do sistema de arquivos, pode-se optar por vários produtos, como por exemplo ***NFS (Network File System)*** ([Ste91] [SUN89]) para usos mais simples, ou ***AFS (Andrew File System)*** [Coh93] para usos mais sofisticados. O servidor de nomes cuida de redundâncias e replicações de dados.

Um dos subprodutos mais expressivos advindos do projeto Athena foi o sistema de autenticação ***Kerberos*** [BM90], voltado para o acesso seguro a computadores remotos. Hoje em dia, o Kerberos é a grande vedete na área de segurança em ambientes de computação distribuída.

Atualmente, o Athena roda em uma configuração de 1500 estações de trabalho, suportando mais de 100 Gigabytes de armazenamento em disco rígido, com 100 computadores hospedeiros de grande e médio porte dando suporte aos mais variados serviços. O número total de usuários passa dos 10 mil.

2.7.4. ANDREW

O ***Projeto Andrew*** ([Mor86] [How88] [Leo94] [Wit91] [LS90] [Tan92] [TVR85]), da Universidade Carnegie Mellon é um dos mais significativos projetos de computação distribuída de grande porte realizado nos anos 80. Andrew constitui um ambiente de computação distribuída que vem sendo desenvolvido desde 1983, e é apoiado financeiramente pela IBM. Está operacional desde 1986, sendo no entanto mais utilizado como um teste beta de alta escala do que como um fornecedor de serviços propriamente dito.

Atualmente, o Andrew está suportando mais de 8.000 usuários, e uma grande variedade de aproximadamente 4.000 computadores, que vão desde estações de trabalho de última geração a antiquados microcomputadores. Aliás, um dos pontos fortes do Andrew é a sua preocupação com escalabilidade.

Uma das grandes contribuições deste projeto, foi no campo da especificação de uma interessante infra-estrutura de redes - a base de toda a conectividade física da computação distribuída de hoje. Por exemplo, os hubs para redes locais foram desenvolvidos inicialmente para suportar as necessidades de conectividade do Andrew. Muitas soluções para a interligação efetiva de redes locais também tiveram ali as suas origens.

Porém, sem dúvida nenhuma, a mais importante contribuição do projeto Andrew fica por conta de um sofisticado serviço de arquivos chamado ***AFS (Andrew File System)*** [Coh93]. O AFS é a base para todos os outros serviços distribuídos do Andrew (por exemplo: correio-eletrônico ou serviços de impressão).

A principal diferença do AFS com relação a outros sistemas de arquivos distribuídos é a sua capacidade de poder servir a milhares de usuários. Por exemplo, o NFS (Network File System) só suporta adequadamente algumas centenas de usuários.

Hoje em dia, o AFS é comercializado por uma empresa chamada Transarc Corporation, que foi fundada por alguns dos projetistas originais do Andrew. O AFS é aceito pela Open Software Foundation (OSF), como o padrão para sistemas de arquivos distribuídos.

Recentemente, muitas das características consideradas proprietárias no Andrew foram revistas, e um grande esforço de reengenharia está sendo efetuado para disponibilizar o Andrew (e, principalmente, o AFS) dentro dos grandes padrões da computação distribuída baseada em sistemas abertos. Este esforço gerou o projeto Andrew II [Leo94] (que ainda está em sua fase inicial), e espera-se que o mesmo seja uma grande contribuição para a computação distribuída.

2.7.5. Chorus

O ***Chorus*** [Pou94a] é um sistema operacional distribuído Unix, baseado em micro-núcleos. Ele nasceu com os estudos das redes de comutação por pacotes nos anos 70, no **INRIA** *(Institut National Récherche en Informatique et Automatique)* [Qua90], um laboratório federal do governo francês, que fica lozalizado nos subúrbios de Paris. Nos treze anos em que vem sendo desenvolvido, o Chorus tem passado por várias versões e tem absorvido os principais conceitos dos mais importantes projetos acadêmicos na área de sistemas distribuídos.

Atualmente, o Chorus suporta mutitarefamento, redes (OSI e TCP/IP), tolerância a falhas, multiprocessamento simétrico e paralelismo, além de manter compatibilidade binária com padrões da indústria através de plataformas distribuídas heterogêneas. Como não poderia deixar de ser, suporta também o paradigma de orientação a objetos.

O Chorus absorveu idéias de sucesso de outros sistemas operacionais distribuídos: do MACH aproveitou as threads e o gerenciamento de memória virtual distribuída; do System V [UI89a], a troca de mensagens; do Amoeba o tratamento dado às redes.

Em 1982, a versão 0 do Chorus estabeleceu os princípios básicos para os pequenos núcleos distribuídos, suportando diretamente os mecanismos de IPC (interprocess communication - comunicação entre processos). Em 1986, os engenheiros do time do Chorus sairam do INRIA e montaram seu próprio negócio: foi criada a Chorus Systems, para explorar o Chorus no mercado comercial. A versão atual, Chorus/Mix, é baseada na versão 3 do Chorus do INRIA.

O Chorus tem sido muito bem aceito na França. A empresa estatal francesa de telecomunicações Alcatel, equivalente à AT&T nos Estados Unidos, adotou o Chorus como o sistema operacional padrão para todos os seus equipamentos de PBX. Recentemente, o Chorus tem chamado a atenção nos Estados Unidos, anunciando acordos com a Unisys, Tandem, Cray Research, The Santa Cruz Operation e Unix Systems Laboratories da Novell. Está disponível para uma grande variedade de hardware, desde a família Intel 80x86 até os transputers Inmos, passando pelos Power-PCs da motorola, que anunciou uma versão especial deste processador com o núcleo do Chorus já embutido no chip, para aplicações especiais.

O Chorus é construído com um minúsculo núcleo, de aproximadamente 50 a 60 Kb, capaz de tratar escalonamento, gerência de memória, eventos de tempo real, e comunicações. Tudo o mais no sistema operacional assume o papel de um *servidor*, que roda acima do núcleo, e comunica-se com ele pela troca de mensagens. Gerentes de arquivos, streams, sockets e mesmo os drivers de periféricos são tratados como *servidores*. Um grupo destes servidores forma um subsistema. No caso do Chorus/MiX, a implementação completa do Unix System V é feita na forma de um subsistema.

Este alto nível de modularidade confere muitas vantagens importantes ao Chorus. Por exemplo, no subsistema Unix, somente aqueles servidores que estiverem realmente em uso precisam ser carregados na memória principal. A facilidade na substituição de um servidor por outro simplifica a implementação de características de tolerância a falhas e backups redundantes.

A Chorus Systems tem centrado suas atenções para a orientação a objetos. O projeto COOL (Chorus Object Oriented Layer) está em andamento, atualmente em uma segunda fase, sendo mantido pelo INRIA e por dois projetos Europeus do ESPRIT.

Todas as questões para sistemas operacionais distribuídos de última geração são tratadas no Chorus de maneira muito elegante e atualizada. É pena que seja um produto francês, o que certamente não o faz ser o melhor dos sistemas operacionais distribuídos atuais - o marketing dos americanos é imbatível, e eles sempre dirão que o seus produtos são os melhores. Do Chorus temos uma idéia de seriedade, e o produto está disponível para quem quiser. Creio que em pouco tempo ainda ouviremos falar muito bem deste sistema operacional distribuído no uso cotidiano.

A USL da Novell já anunciou que irá oferecer o Chorus/MiX como uma implementação do SVR4. A USL e a Chorus Systems já estão trabalhando em parceria para desenvolver uma versão do Chorus/MiX V.4 que irá redirecionar o mundo Unix [Var94].

2.7.6. Outros Sistemas Operacionais Distribuídos

Outros projetos também contribuíram para o desenvolvimento de soluções pragmáticas para a computação distribuída, e vale a pena serem citados:

- O sistema ***Cambridge Distributed Computing System*** [TVR85] do laboratório de computação da Universidade de Cambridge, na Inglaterra. Constitui um dos mais antigos esforços de desenvolvimento em redes e sistemas distribuídos já levados adiante, tendo seu inicio em meados dos anos 70.
- O sistema ***Eden*** [TVR85] da Universidade de Washington em Seattle. É um protótipo baseado em programação paralela utilizando a linguagem ADA.
- O projeto ***Unix United*** [LS90], da Universidade de New Castle Upon Tyne, na Inglaterra. Constitui uma das primeiras tentativas de se estender o sistema de arquivos do Unix para um ambiente distribuído. Foi implementado em antigos computadores PDP-11 da Digital.[91]
- O projeto ***LOCUS*** [LS90] da Universidade da California em Los Angeles (UCLA). Importante na definição de padrões para aplicações dirigidas por transações. Está operacional desde 1986 e é usado como um grande laboratório para testes de modelos distribuídos na UCLA. É um sistema baseado em Unix.
- O ***Sprite*** [LS90] é um sistema operacional distribuído experimental, da Universidade da California em Berkeley. Foi implementado preliminarmente em estações de trabalho SUN interligadas (1986). Contribuiu para a definição de engenhosos mecanismos de "caching" objetivando melhorar o desempenho dos sistemas distribuídos.

2.8. Os Desafios para a Computação Distribuída

A integração de todas as plataformas de computação corporativas e suas aplicações em um ambiente de computação distribuída é uma tarefa cheia de desafios ([Kha94b] [KP92] [Ros93]). Os profissionais de informática que até então têm utilizado ambientes centralizados, baseados em mainframes, começam a perceber claramente como é difícil obterem-se resultados rápidos e adequados para atender ao "backlog" destes ambientes. Nos ambientes da computação distribuída a solução dos problemas é mais rápida, graças às ferramentas de alto nível voltadas para facilitar a vida dos usuários finais, e ao uso de facilidades disponíveis em estações de trabalho clientes. No entanto, a complexidade para se gerenciar sistemas, segurança e desempenho em ambientes de computação distribuída é muito maior. Falamos a seguir de alguns dos desafios para se implementar e gerenciar as infra-estruturas da computação distribuída .

2.8.1. A Complexidade

A complexidade dos ambientes da computação distribuída aumenta significativamente devido a fatores como a grande abrangência, heterogeneidade, segurança, distribuição, nomes e sincronização [KP92]. Ferramentas para a gerência integrada destes ambientes ainda deixam a desejar. Nos ambientes da computação distribuída a responsabilidade dos usuários aumenta consideravelmente devido ao uso de computadores de mesa (desktops) ao invés de terminais "mudos".

2.8.2. Segurança e Integridade de Dados

Segurança pode ser considerada um dos aspectos mais importantes da computação distribuída. Em um dia de trabalho, um usuário típico pode fazer acessos a até 50 serviços diferentes. Sem um serviço de autenticação que possa ser usado por toda a rede corporativa, um usuário terá que ser autenticado e autorizado para todos os serviços requisitados. A manutenção de arquivos de auditoria também é muito importante.

Assim, mecanismos de segurança como por exemplo a autenticação, o controle de acessos, criptografia, e logs de auditoria devem ser projetados e implementados como uma parte inerente ao próprio ambiente de computação distribuída de uma empresa. O mais fraco destes pontos pode representar o calcanhar de Aquiles para o ambiente de computação distribuída implementado. Tais mecanismos devem ser derivados a partir de uma combinação de tecnologias e políticas acordadas para todas as unidades organizacionais de uma empresa.

2.8.3. Desempenho

As dúvidas que surgem quando falamos de desempenho em ambientes da computação distribuída envolvem as seguintes questões: como medir o desempenho dos ambientes da computação distribuída? [How88] Deve-se usar TPMs? (Transactions per Minute). Qual a vazão para o acesso a um arquivo remoto? É muito importante que se gerenciem as expectativas com relação à confiabilidade e desempenho. Decisões sobre a necessidade de largura de faixa adicional, ou sobre a substituição das estações clientes (ou servidores) por plataformas mais poderosas, devem ser muito bem planejadas [Hur93d].

2.8.4. Mudanças Organizacionais

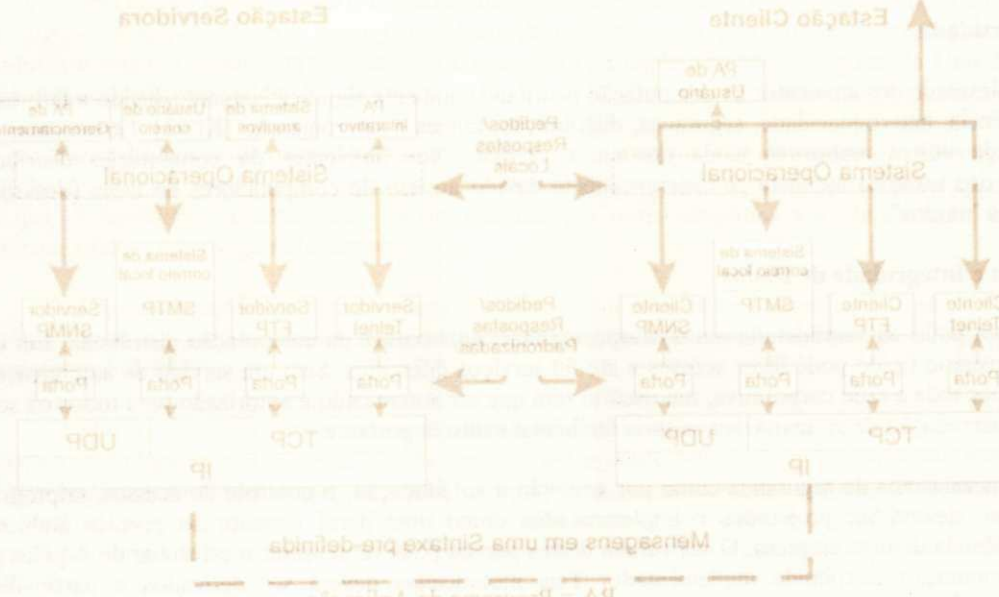
A computação distribuída resulta em economia de custos em certas áreas e em gastos extras em outras áreas [Sta94a]. As empresas precisam deslocar os ganhos de economia no departamento de informática para financiar os departamentos de usuários finais [Gow91]. Como os ambientes da computação distribuída são mais flexíveis do que os ambientes baseados em mainframes, estes são também mais complexos. Várias funções do departamento de informática devem ser reorganizadas para suportar estes novos e complexos ambientes.

Em ambientes da computação distribuída, é muito importante que exista uma equipe altamente qualificada que seja responsável pelo planejamento estratégico da informática na empresa [Far93c]. Em muitos casos, uma transição para um ambiente da computação distribuída é também acompanhado com uma transição para um modelo de suporte distribuído. É um erro muito comum que profissionais de departamento de informática com funções puramente de planejamento sejam muitas vezes eliminados na migração para o novo ambiente. Em ambientes da computação distribuída, é importantíssimo que as atividades computacionais em unidades individuais sejam coordenadas, e que se planeje toda a infra-estrutura [Hur93d].

2.8.5. Treinamento

Apesar das facilidades fornecidas pelo uso de GUIs, surge uma necessidade crescente para treinar usuários finais quando um primeiro conjunto de aplicações começam a ser migradas. As empresas devem se encarregar de garantir para seus usuários que as diferentes aplicações sigam uma interface padronizada.

Requisitos técnicos, tanto centrais quanto a níveis organizacionais menores, aumentam consideravelmente. Para que se possa diagnosticar problemas os administradores de sistemas precisam estar familiarizados com conceitos de redes e de sistemas operacionais. Problemas de desempenho podem ser causados por redes ou servidores saturados, ou por outros problemas nas máquinas clientes. Ferramentas de gerência de sistemas e de redes fornecem amostras de contextos de tráfego e de estatísticas em segmentos particulares da rede, ou em espinhas dorsais (backbones) que suportam as redes corporativas. O mais difícil de tudo é se conseguir acompanhar todos os passos na execução de uma aplicação distribuída por vários servidores. Neste novo ambiente, treinamento torna-se imprescindível ([Sey93] [WSL93] [Col93a] [Mye93a]).



Capítulo 3: Os Serviços Não-Transparentes

3.1. Introdução

Convencionamos chamar de serviços não transparentes, aqueles serviços que estão diretamente disponíveis para os usuários de uma rede de computadores, e servem para facilitar as tarefas mais triviais requisitadas por todas as pessoas envolvidas no uso da rede, como por exemplo, a transferência de arquivos, a emulação de terminais, a troca de mensagens por meio de facilidades de correio-eletrônico e a execução remota de aplicações. Todas estas facilidades são fornecidas a nível da camada de aplicação, tanto da pilha de protocolos OSI quanto da pilha de protocolos TCP/IP. Falamos neste capítulo, inicialmente dos serviços não-transparentes da pilha TCP/IP, e em seguida apresentamos os serviços equivalentes e suas características dentro do contexto da pilha OSI.

3.2. Os Protocolos de Aplicação TCP/IP

Escolhemos alguns dos protocolos de aplicação TCP/IP para apresentarmos neste capítulo. Abordamos os protocolos mais conhecidos, mostrados na figura 3.1, incluindo:

FTP [War93]: que um usuário (ou um processo de aplicação) acesse e interaja com um sistema de arquivos remoto.

Telnet ([Der92c] [Bar93b]): que permite que um usuário (ou processo de aplicação) em determinada máquina na rede se comunique interativamente com outro processo de aplicação, como por exemplo um editor de textos, rodando em uma máquina remota, e dando a impressão de o usuário estar diretamente ligado a esta máquina remota.

SMTP [FA94]: fornece um serviço de correio-eletrônico global entre sistemas de correio-eletrônico disponíveis em diferentes máquinas em uma rede.

SNMP ([Wal94] [Sta93b]): é o protocolo de gerenciamento dos ambientes Internet. Ele permite que um usuário (o administrador da rede) acumule dados de desempenho ou controle a operação de um elemento da rede (como por exemplo, uma ponte ou um roteador). Citamos o mesmo neste capítulo somente com o intuito didático de apresentá-lo no mesmo nível dos outros protocolos de aplicação disponíveis na pilha TCP/IP. No capítulo 9 tratamos do SNMP exaustivamente, inclusive analisando-o do ponto de vista de uso prático e integrado a outras facilidades.

DNS ([Goe93a], Goe93b]): é um bom exemplo de um protocolo para nomes, que fornece uma tradução do tipo nomes-para-endereços-IP. Citado neste capítulo também somente por questões didáticas. Será visto com maior riqueza de detalhes no capítulo 5.

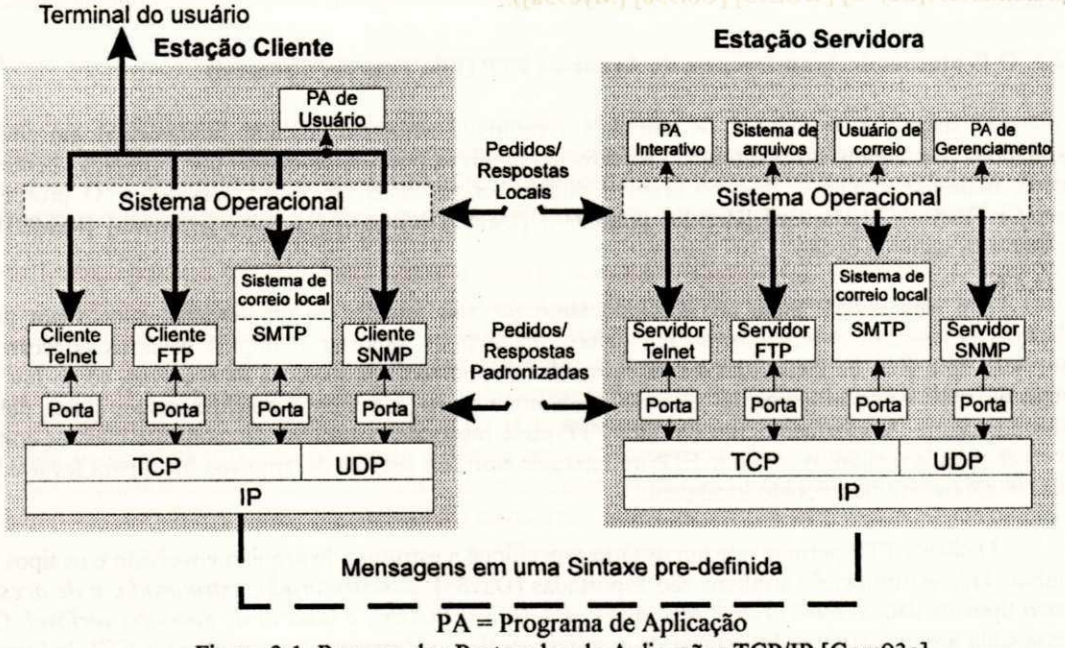


Figura 3.1. Resumo dos Protocolos de Aplicações TCP/IP [Com93a]

3.2.1. O Estabelecimento da Conexão

Uma exigência comum para todas as interações é o estabelecimento de um caminho de comunicação entre os dois protocolos/processos de aplicação envolvidos. Antes de falarmos sobre os diferentes protocolos de aplicação da pilha TCP/IP, faremos uma breve apresentação da maneira de se estabelecer este caminho de comunicação.

Todos os processos de aplicação servidores possuem um nome associado que é traduzido para um endereço correspondente na rede. Este procedimento de tradução é efetuado por um protocolo de aplicação chamado DNS (Domain Name Server) ([Goe93a] [Goe93b]). A operação do DNS será apresentada no capítulo 5 quando falamos a respeito de serviços de nomes e diretórios.

O DNS é um bom exemplo de um protocolo para nomes, que fornece uma tradução do tipo nomes-para-endereços-IP. Permite também a administração descentralizada de nomes de recursos e especifica redundância de servidores como um meio de fornecer serviços de consultas eficientes e confiáveis para os usuários. Originalmente o DNS era utilizado para permitir que computadores em rede que enviassem mensagens de correio-eletrônico pudessem descobrir o endereço IP dos computadores destinatários destas mensagens, ou de computadores que pudessem recebê-las e passá-las adiante pelos nós intermediários.

O endereço de rede gerado pelo DNS para um processo servidor consiste de duas partes: o endereço IP da rede para o computador hospedeiro no qual o processo está rodando, e um número de porta local. O endereço IP é usado pelo protocolo IP em cada gateway internet para rotear os datagramas através das sub-redes até um computador hospedeiro destinatário. O número da porta é então usado pelo protocolo TCP (ou pelo UDP) no computador hospedeiro para identificar o processo específico neste computador, e entregar a este processo a mensagem recebida.

Um sistema aberto inclui múltiplos clientes e servidores [HM91]. Todos os servidores do mesmo tipo (por exemplo, servidores de arquivos, servidores de nomes, etc.), no entanto, têm assinalado para eles o mesmo ***número de porta*** [Blo92] por toda a rede (incluindo todas as sub-redes que a compõem). Assim, um servidor específico é identificado pelo endereço IP da máquina na qual ele estiver rodando. Os números de portas associados com diferentes tipos de servidores são conhecidas como ***portas bem conhecidas*** [Blo92], como por exemplo: a *porta 21* para servidores FTP, a *porta 23* para servidores Telnet, ou a *porta 25* para servidores SMTP [Com91a].

Assim, quando um processo cliente inicia a chamada a um processo servidor correspondente, ele utiliza como endereço destinatário o endereço IP da máquina na qual o servidor estiver rodando, acoplado à porta bem conhecida apropriada para este tipo de servidor. Como endereço fonte, o cliente utiliza o endereço IP de sua própria máquina juntamente com o próximo número de porta disponível nesta máquina. Se o TCP estiver sendo usado, o módulo do protocolo TCP local (ou entidade TCP local) irá então estabelecer uma conexão de transporte entre os processos cliente e servidor - pelo uso destes endereços - através do qual as mensagens apropriadas poderão ser trocadas.

3.2.2. O Protocolo de Transferência de Arquivos FTP (File Transfer Protocol)

O acesso a servidores de arquivos remotos é uma necessidade fundamental em muitas aplicações distribuídas. Em alguns casos um único servidor de arquivos pode ser acessado por múltiplos clientes, enquanto em outros, múltiplas cópias do mesmo arquivo podem ser mantidas em vários servidores. O protocolo **FTP** (*File Transfer Protocol*) ([Bar93b] [Bri94b] [Com93a] [Dav89] [Gun94] [Hal93b] [Hum92a] [Keh92]) é usado para facilitar esta segunda opção.

Um cliente FTP pode ser acessado tanto por um usuário em um terminal quanto por um processo de aplicação de usuário. Normalmente, um único cliente pode suportar múltiplos usuários concorrentemente. São fornecidas uma série de facilidades similares àquelas disponíveis nos sistemas de arquivos, como por exemplo: listar diretórios, criar novos arquivos, ler o conteúdo de um arquivo já existente, atualizar este arquivo, deletar arquivos e assim por diante. Similarmente, um servidor FTP pode responder a pedidos de múltiplos clientes concorrentemente. Ao receber cada pedido, o servidor FTP irá interagir com seu sistema de arquivos local para levar adiante o pedido como se ele tivesse sido gerado localmente.

O cliente FTP permite que um usuário especifique a estrutura do arquivo envolvido e os tipos de dados neste arquivo. Três estruturas de arquivos são suportadas [Dav89]: *desestruturada, estruturada, e de acesso aleatório*, e quatro tipos de dados: *binário de 8 bits, texto: ASCII ou EBCDIC, e binário de tamanho variável*. O servidor FTP acessa cada arquivo através do sistema de arquivos local, e os transfere para os clientes FTP de forma apropriada e de acordo com a estrutura definida.

Arquivos desestruturados podem conter qualquer tipo de dados. *Arquivos estruturados* possuem registros de tamanho definido, e podem ser transferidos de forma compactada. Os *arquivos de acesso aleatório* possuem registros de tamanho variável. Existe a possibilidade de na transferência de grandes arquivos poder-se utilizar mecanismos de "checkpointing", de maneira similar aos serviços oferecidos pela camada de sessão na pilha OSI.

O FTP controla a transferência de arquivos entre computadores hospedeiros em uma rede por meio de duas conexões TCP: uma para a troca de comandos e respostas, e outra para a troca de arquivos de fato. Vários serviços importantes são fornecidos, incluindo a conexão de controle FTP e a conexão de dados FTP [Hal93b].

A Conexão de Controle FTP

O FTP emprega o modelo usuário/servidor [War93] que contém dois componentes: o *FTP usuário*, e o *FTP servidor*. O FTP usuário ativa o processo de transferência e o FTP servidor responde. O FTP usuário geralmente fornece uma interface interativa para os usuários, de tal forma que eles possam requisitar a transferência de arquivos de/para os computadores na rede.

Quando um FTP usuário inicia uma transferência de arquivos, é aberta uma *conexão de controle* [Dav89] com o FTP servidor. São então trocados comandos e respostas com o FTP servidor para propósito de identificação de usuários, e para que se determine a natureza da operação desejada (armazenar, recuperar, remover, adicionar).

A Conexão de dados FTP

Quando a transferência do arquivo está pronta para ser iniciada, o FTP usuário e o FTP servidor estabelecem uma segunda conexão TCP para suportá-la. Esta conexão de dados é usada tanto para enviar quanto para receber dados.

O FTP Trivial (TFTP)

O protocolo FTP é relativamente complexo, pois ele contém todas as estruturas necessárias para serem usadas por uma grande variedade de tipos de arquivos e, caso seja necessário, podem ser usados algoritmos de compressão. Apesar destes serviços serem comumente usados em inter-redes, o nível de funcionalidade oferecida não é normalmente requerido em aplicações locais. Um exemplo sugestivo é a transferência de arquivos entre servidores de arquivos e uma série de estações sem disco em um segmento de rede local.

Nestes casos mais simples, um protocolo para arquivos adicionais está disponível, e é justamente o **TFTP** *(Trivial FTP)* [Hal93b]. TFTP utiliza UDP, e não o TCP, para a troca de mensagens. O uso do TFTP só é recomendado para uso em segmentos de redes locais isolados, devido à baixa taxa de erros fornecida nestes ambientes. No entanto, o seu uso é bastante recomendado neste ambientes, pois reduz-se consideravelmente o overhead adicional gerado pelas exigências impostas às conexões TCP.

3.2.3. Telnet

Telnet [Der92c] fornece uma facilidade de comunicação orientada a byte com oito bits, bidirecional, e de propósito geral, voltada para o interfaceamento de terminais. Uma conexão Telnet é uma conexão TCP [Hal93b]. Na figura 3.2 podemos observar o protocolo/processo cliente Telnet sendo acessado pelo sistema operacional local, seja através de um processo de aplicação de usuário, ou, como é mais comum, por um usuário em um terminal.

Telnet fornece serviços que permitem que um usuário se conecte (login) ao sistema operacional de uma máquina remota, para que ele possa iniciar um programa nesta máquina, como por exemplo um editor de textos, e depois interaja com este programa como se este usuário do processo/terminal estivesse conectado/rodando na mesma máquina. Todos os comandos (caracteres de controle) e dados entrados no terminal do usuário, ou submetidos pela aplicação do usuário, são passados pelo sistema operacional local para o processo cliente Telnet, o qual os encaminha através do serviço de fluxo confiável do TCP para o servidor Telnet correspondente. Este servidor emite em seguida os comandos requisitados pelo usuário, através do sistema operacional local, para o processo interativo. Os dados gerados neste processo interativo são retornados da mesma forma para apresentação no terminal cliente, ou então para serem interpretados pela aplicação de usuário.

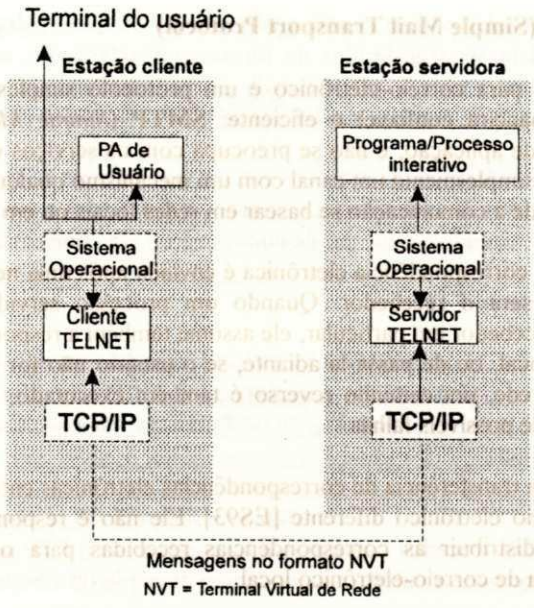


Figura 3.2. Esquema da interação cliente/servidor para o Telnet [Com93a]

Clientes e servidores Telnet comunicam-se entre si por meio de comandos, os quais são codificados em um formato padrão conhecido como **NVT** (*Networked Virtual Terminal*) ([Bri94b] [Com93a] [Der92c] [Yem94). O conjunto de caracteres usado para estes comandos é o ASCII. Todos os dados de entrada/saída relacionados com uma interação são transferidos como cadeias ASCII. Se o conjunto de caracteres local for diferente, o correspondente Telnet irá executar as funções de mapeamento necessárias. Assim, as duas entidades de protocolos Telnet (clientes e servidores) irão executar a função da camada de apresentação da pilha OSI. Dois conceitos importantes são empregados em Telnet: Terminal Virtual em Rede, e opções negociadas [Hal93b].

Terminal Virtual em Rede (NVT - Networked Virtual terminal)

Na maioria das vezes, o Telnet é usado para conectar terminais remotos a computadores centrais. Quando uma conexão Telnet é estabelecida pela primeira vez, ela se origina e termina em um **NVT**. Assim, um NVT é um dispositivo imaginário, que representa um terminal real.

Quando se utiliza Telnet para estabelecer a conexão de um terminal com um computador, um programa chamado "usuário Telnet" traduz os caracteres usados pelo terminal para o conjunto de caracteres disponíveis no NVT, e despacha estes caracteres NVT para o par remoto. No par remoto, um programa "servidor Telnet" traduz os caracteres NVT para o conjunto de caracteres usado pelo computador hospedeiro, que a seguir despacha estes caracteres para o sistema operacional do mesmo, dando a impressão que estes caracteres foram gerados localmente.

O Princípio das Opções negociadas

O Telnet emprega um mecanismo de negociação, chamado de **opções negociadas [Dav89]**, que permite que um cliente e um servidor entrem em concordância com relação ao uso de diferentes conjuntos de convenções para uma conexão Telnet. Desta forma, o emissor informa se deseja ou não executar algum tipo de opção, ou então instrui o outro nó a fazer uma opção, ou a não fazê-la. É um mecanismo considerado elegante e que permite que o dono da opção requisitada responda, na forma negativa, sem precisar entender nada a respeito da opção requisitada.

Estas opções podem incluir uma mudança no conjunto de caracteres, no modo de eco, no comprimento de uma linha, etc. Muitas opções existem pré-definidas, incluindo a transmissão binária; eco de caracteres remotos; desabilitação de linhas lógicas tipo "half-duplex"; reconhecimento de estado; e, a transmissão de rótulos de tempo ("timestamps").

3.2.4. O Protocolo SMTP (Simple Mail Transport Protocol)

O padrão Internet para correio-eletrônico é um protocolo simples, orientado a textos e projetado para transferir mensagens de maneira confiável e eficiente. **SMTP** (*Simple Mail Transfer Protocol*) [FA94] é um protocolo puro da camada de aplicação, e não se preocupa com os serviços de transporte que o suportam. Pode-se usar uma conexão TCP, ou simplesmente um canal com um mecanismo qualquer para a comunicação entre processos (dependendo, logicamente, de a comunicação se basear em redes locais ou em redes de longo alcance).

Cada parte de uma correspondência eletrônica é enviada após uma negociação inicial a respeito de quem é o emissor original, e quem será o recebedor. Quando um processo servidor SMTP concorda em aceitar uma correspondência para um recebedor em particular, ele assume também a responsabilidade de enviar a correspondência para o usuário, se ele for local, ou de passá-la adiante, se o usuário não for local. No caminho percorrido por uma mensagem que trafega na rede, um caminho reverso é também executado, tornando possível que se notifique ao emissor original a respeito de possíveis falhas.

O SMTP gerencia a transferência de correspondências eletrônicas entre um computador na rede, e outro que utilize um sistema de correio eletrônico diferente [ES93]. Ele não é responsável por aceitar correspondências de usuários locais, nem por distribuir as correspondências recebidas para os destinatários adequados. Estas são responsabilidades do sistema de correio-eletrônico local.

Como o SMTP interage com o sistema de correio-eletrônico local e não diretamente com o usuário, ele fica mascarado de todas as transferências que sejam locais àquela máquina onde está rodando o processo cliente. Somente quando uma correspondência é para ser enviada para outras máquinas, ou quando uma mensagem é recebida de uma máquina remota, então o SMTP é escalonado para rodar.

Para enviar uma correspondência eletrônica o cliente SMTP primeiro certifica-se do endereço IP do computador destinatário por meio do serviço de diretórios - o DNS - e em seguida utiliza este endereço, juntamente com a porta bem conhecida do SMTP (25), para poder iniciar o estabelecimento da conexão de transporte com o servidor SMTP no computador destinatário. Depois de estabelecida a conexão, o cliente inicia a transferência da correspondência para o servidor [Com91a].

É interessante observar-se que, na prática, nem sempre os computadores que estão trocando correspondências eletrônicas estão utilizando o protocolo SMTP. Muitos outros protocolos de correio eletrônico são usados pelas redes. Para permitir a troca de correspondências eletrônicas com outros sistemas de correio eletrônico, é necessário que se utilize um "*gateway de correio-eletrônico*" ([Eng94a] [PA92]). Um exemplo é um gateway TCP/IP-para-OSI, onde correspondências recebidas usando uma porta SMTP em uma sub-rede são passadas adiante usando o MOTIS - o protocolo de correio eletrônico da ISO - para uma porta de outra sub-rede.

Voltamos a falar sobre o SMTP com mais detalhes no capítulo 4 adiante, quando abordamos os sistemas de correio-eletrônico.

3.3. Os Protocolos de Aplicação ISO/OSI

Um *conjunto completo de protocolos* específicos *da camada de aplicação* é definido pela ISO [HM91]. Muitos destes protocolos são considerados padrões internacionais completos, enquanto outros encontram-se ainda na forma de rascunhos ("drafts"). Existe muita similaridade entre os serviços fornecidos pelos protocolos de aplicação TCP/IP, porém normalmente uma série de serviços adicionais são fornecidos. Os protocolos que aqui apresentamos são os mais influentes, porém vários outros são fornecidos:

VT [SOIEC89]: é um padrão internacional (de jure), e fornece serviços para emulação de terminais.

FTAM ([HM91] [Cha92] [Mal92b] [Kle88] [Hal93b]): é um padrão internacional (de jure), e fornece serviços para a transferência de arquivos.

MOTIS ([FA94]. [Qua90] [VN90b]): este padrão internacional (de jure) fornece serviços de tratamento de mensagens de correio-eletrônico.

JTM ([Hal93b] [HM91]]: é também um padrão internacional (de jure), e fornece facilidades para que usuários submetam serviços (via programas de aplicação) para que um programa de aplicação remoto possa executá-lo.

Como ocorre com os protocolos de aplicação TCP/IP, o objetivo dos vários protocolos ISO é permitir que dois processos de aplicação que estejam rodando em diferentes computadores comuniquem-se entre sí para executar uma função particular de uma aplicação distribuída. Assim, em FTAM por exemplo, o objetivo é permitir que um processo cliente rodando em um computador interaja com o processo servidor de arquivos rodando em um computador remoto (e possivelmente diferente), e dando a impressão de o processo cliente estar rodando no mesmo computador do servidor.

Para conseguir este objetivo, o enfoque ISO define um ***modelo de recurso virtual*** (*virtual device model*) [Yem93] para cada função de uma aplicação distribuída: um terminal virtual, uma área virtual de armazenamento de arquivos, etc.

3.3.1. A Camada de Aplicação OSI - Conceitos Básicos

Na pilha OSI, os protocolos de aplicação (ou processos de aplicação) interagem por meio de *entidades de protocolos* ([Bri94] [Sta93b] [Yem93]) associadas com as camadas de sessão e apresentação intermediárias. Como mostrado na figura 3.3, a camada de aplicação consiste de dois conjuntos de protocolos, cada um dos quais conhecidos como *elemento de serviço de aplicação* ou **ESA**. Um dos conjuntos executa funções de aplicação específicas, enquanto o outro executa funções de propósito mais geral chamadas de *elementos de serviços de aplicações comuns* ou **ESAC**.

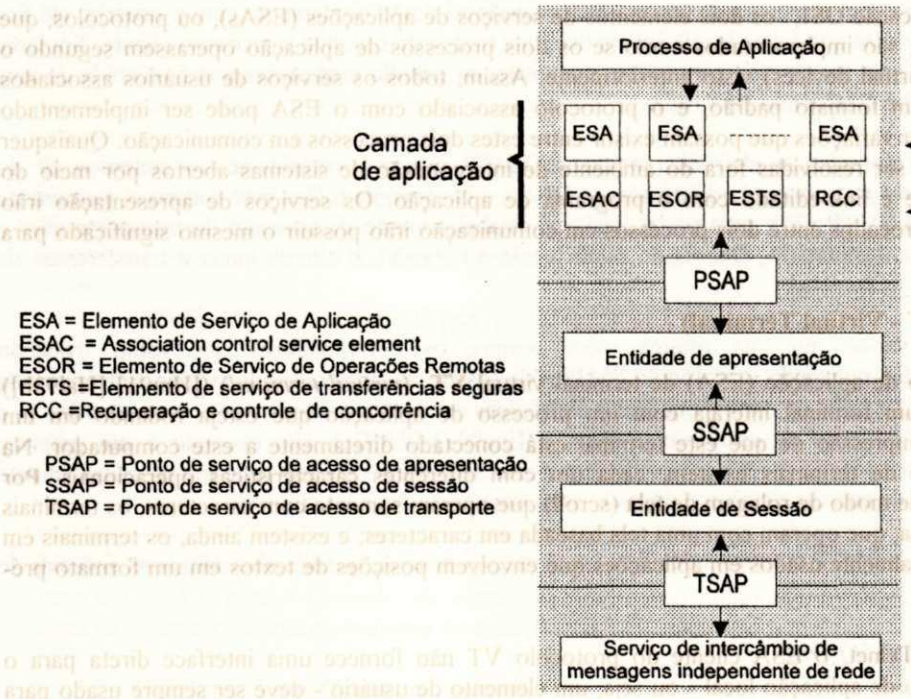


Figura 3.3. Os protocolos de suporte de aplicação no modelo RM/OSI [Kle88]

Para ilustrarmos um princípio muito importante, o da independência de funções no modelo OSI, é importante que se note que durante a transferência de dados entre computadores em uma rede, é essencial que se assegure que a sintaxe dos dados seja conhecida pelo computador destinatário, e se esta for diferente da sintaxe do computador local, deve-se converter os dados para a sintaxe adequada antes do processamento. Este é um procedimento comum em muitas aplicações distribuídas, especialmente naquelas em sistemas abertos que envolvem a interconexão de redes com computadores de diferentes fabricantes. O enfoque adotado pela ISO é de integrar estes tipos de procedimentos na pilha de protocolos. No caso em particular, ou seja, a conversão da sintaxe, é efetuada por uma função especializada da camada de apresentação [Tan89], a qual dentre outras funções cuida ainda de compressão e encriptação de dados.

De maneira similar, várias outras funções são comuns para muitas aplicações. O enfoque da ISO é separar estas funções daquelas funções específicas de protocolos, e implementá-las como um conjunto de **protocolos de suporte** [Hal93b]. Assim, se uma função particular de suporte for requerida em uma aplicação, uma instância do(s) protocolo(s) de suporte adicional(ais) é (são) link-editada(s) juntamente com o protocolo específico daquela aplicação em particular.

Todos os protocolos associados com a camada de aplicação são chamados de *elementos de serviços de aplicação* (**ESAs**), como já havíamos dito anteriormente. Os ESAs que executam funções de propósito geral incluem o *elemento de serviço de controle de associação* (**ESCA**) , o *elemento de serviços de operações remotas* (**ESOR**), o *elemento de serviço de transferência segura* (**ESTS**) e o *elemento de serviço de recuperação e controle de concorrência* (**RCC**).

Dois processos/protocolos de aplicação podem intercomunicar-se de duas maneiras: ou por meio de uma conexão lógica, chamada de ***associação*** ([Sta93b] [HM91]), estabelecida antes de as mensagens começarem a ser transferidas, ou por meio simplesmente do envio de uma mensagem diretamente para o destinatário, e se for o caso, aguardar por uma confirmação. O primeiro enfoque é apropriado para o envio de grandes volumes de dados. O segundo pode ser mais apropriado para uma troca de mensagens entre clientes e servidores. Os serviços fornecidos pelo ESCA foram definidos para atender ao primeiro caso, enquanto que os serviços definidos pelo ESOR são apropriados para o segundo caso. Adicionalmente, o ESTS incluí os serviços tanto do ESCA quanto os serviços da camada de sessão que forem selecionados.

As funções fornecidas pelo RCC serão vistas com mais detalhes quando falamos do protocolo **DTP** ([Cla92b] [Cas93] [Dew93]) (*Distributed Transaction Processing*) no capítulo 11 sobre sistemas para gerenciamento de transações em linha (*sistemas OLTP)*. Em resumo o RCC fornece mecanismos de controle de acesso a recursos partilhados nos ambiente distribuídos.

Nos protocolos de aplicação OSI, os dois elementos de serviços de aplicações (ESAs), ou protocolos, que estiverem intercomunicando-se, são implementados como se os dois processos de aplicação operassem segundo o modelo de recursos virtuais (virtual devices) visto anteriormente. Assim, todos os serviços de usuários associados com cada ESA irão possuir um formato padrão, e o protocolo associado com o ESA pode ser implementado independentemente de quaisquer variações que possam existir entre estes dois processos em comunicação. Quaisquer diferenças que existam podem ser resolvidas fora do ambiente de interconexão de sistemas abertos por meio do elemento de usuário (EU) que é link-editado com o programa de aplicação. Os serviços de apresentação irão assegurar que os dados sendo trocados entre dois processos em comunicação irão possuir o mesmo significado para ambos os processos.

3.3.2. Terminal Virtual (VT - Virtual Terminal)

O elemento de serviço de aplicação (ESA) de terminal virtual **VT** (*virtual terminal*) ([Hm91] [Hal93b]) permite que um usuário em um terminal interaja com um processo de aplicação que esteja rodando em um computador remoto dando a impressão de que este terminal está conectado diretamente a este computador. Na prática, uma grande variedade de terminais existem, cada um com diferentes características operacionais. Por exemplo, existem os terminais de modo de rolagem de tela (scroll) que operam somente com caracteres; os terminais em modo de página ou tela cheia, que operam com uma tela baseada em caracteres; e existem ainda, os terminais em modo formatado, que são normalmente usados em aplicações que envolvem posições de textos em um formato pré-definido.

Em contraste com o Telnet, o ESA cliente do protocolo VT não fornece uma interface direta para o terminal. Ao invés, um processo de aplicação local - ou seja, um elemento de usuário - deve ser sempre usado para gerenciar a interação com o terminal. É este elemento de usuário que interage com o ESA cliente do protocolo VT de acordo com as características definidas para o terminal virtual. Um esquema geral é apresentado na figura 3.4. a seguir.

Devido à grande variedade de terminais, não existe um terminal virtual único. O ESA VT fornece os meios para dois usuários negociarem as características específicas para o terminal virtual requerido poe uma determinada aplicação. Antes de ocorrerem quaisquer interações, ambos usuários devem concordar com o *ambiente do terminal virtual* (**ATV**) [Bri94b] que será usado por esta aplicação.

Os dois usuários do protocolo VT comunicam-se entre sí por meio de uma estrutura de dados partilhada chamada de *área de comunicação conceitual* (**ACC**) . Cada ACC contém várias estruturas de dados que em conjunto descrevem as características do terminal virtual que está sendo usado. Uma cópia separada da ACC é mantida para cada entidade do protocolo VT. As modificações no conteúdo da ACC mantidas por cada entidade são inicializadas pelo processo de usuário local, o qual emite um pedido para uma primitiva VT pré-definida na interface. As mudanças resultantes são então passadas para a ACC local e em seguida propagadas para a entidade VT par no sistema remoto por meio do protocolo VT.

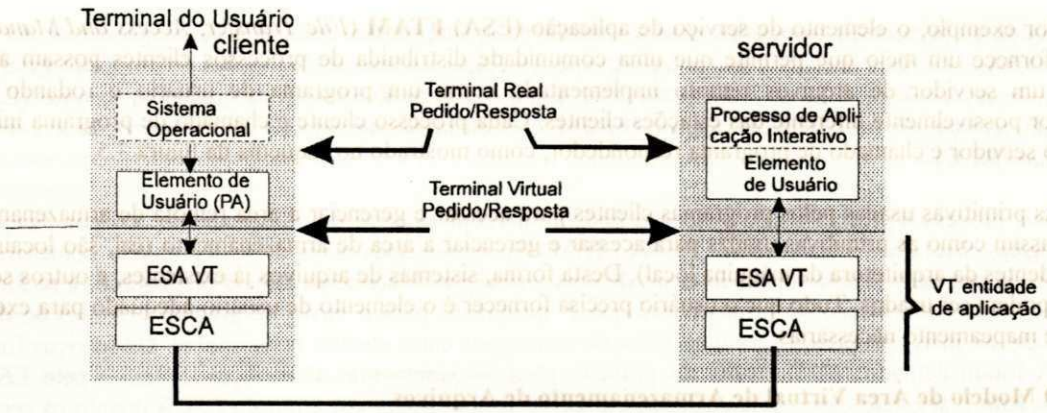


Figura 3.4 O esquema de interação para o protocolo VT [Hal93b]

Associados aos dispositivos de entrada e saída (teclado, mouse, impressora, etc.) em cada terminal existem três objetos, cada um composto por vários parâmetros ([Sta93b] [SOIEC91c]):

- *Objetos de Apresentação*: permitem que eventos relacionados com os dispositivos reais associados sejam apresentados pelos eventos correspondentes associados com o dispositivo virtual.
- *Objetos de Controle*: são usados para modelar características de terminais que não são normalmente disparadas pelos usuários, como por exemplo, uma advertência sonora ("bell").
- *Objetos de Dispositivos*: são usados para modelar as características dos dispositivos reais associados.

Em VT os modos de operação *síncrono* ou *assíncrono* são suportados. No modo síncrono, as funções de entrada e saída em cada lado da associação são combinados. Assim, é necessário que sejam usadas várias *fichas* ("*tokens*") para se controlar a ordem da interação entre os usuários e os processos remotos. No modo assíncrono, as funções de entrada/saída são separadas permitindo que cada lado inicie um evento concorrentemente. Uma área de *armazenamento de controle de acesso* (**ACA**) guarda os assinalamentos correntes das fichas sendo usadas quando seleciona-se o modo síncrono. Na prática, as fichas são transferidas por meio de um serviço de controle de fichas fornecido na camada de sessão.

3.3.3. Gerenciamento e Acesso às Transferências de Arquivos - FTAM

No início da seção 3.3 falamos que os vários elementos de serviços de aplicação (ESAs) não estão envolvidos com o fornecimento de serviços de aplicação específicos, mas sim com o fornecimento dos meios que permitam que um serviço fornecido por um programa de usuário no ambiente do sistema real possa ser acessado e usado de maneira aberta.

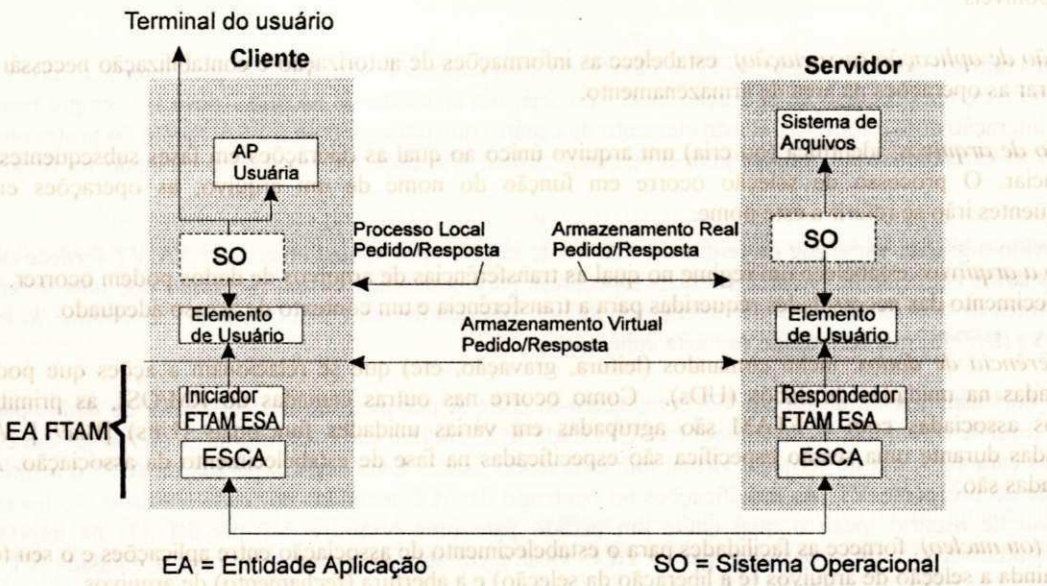


Figura 3.5. As interfaces para dispositivos virtuais do FTAM [Com93a]

Por exemplo, o elemento de serviço de aplicação (ESA) **FTAM** (*File Transfer, Access and Management*) [HM91], fornece um meio que permite que uma comunidade distribuída de processos clientes possam acessar e gerenciar um servidor de arquivos remoto implementado como um programa de usuário e rodando em um computador possivelmente diferente das estações clientes. Cada processo cliente é chamado de programa iniciador e o processo servidor é chamado de programa respondedor, como mostrado no esquema da figura 3.5.

As primitivas usadas pelos programas clientes para acessar e gerenciar a área remota de armazenamento de arquivos, assim como as primitivas usadas para acessar e gerenciar a área de armazenamento real, são locais (isto é, são dependentes da arquitetura da máquina local). Desta forma, sistemas de arquivos já existentes, e outros softwares de acesso podem ser usados. Tudo que o usuário precisa fornecer é o elemento de usuário adequado para executar as funções de mapeamento necessárias.

O Modelo de Área Virtual de Armazenamento de Arquivos

Este é um modelo suficientemente flexível para permitir que quaisquer sistemas de arquivos reais sejam prontamente acessados e gerenciados com um conjunto mínimo de funções de mapeamento.

A ***área virtual de armazenamento de arquivos*** ([Dav89] [Keh92]) é modelada como uma entidade endereçável através da qual um usuário (iniciador) pode se comunicar (estabelecer uma associação). Um número arbitrário de iniciadores podem ter uma associação com a área do arquivo respondedor a qualquer momento. A área de armazenamento de arquivos pode englobar um número arbitrário de arquivos, e cada um destes arquivos pode ter vários atributos. Estes atributos são: um nome; direitos de acesso (lê, grava, etc); tamanho; identidade do dono; data e hora de criação; etc.

Três estruturas de arquivos são possíveis: *desestruturada, "flat" e hierárquica*. Para os *arquivos desestruturados* não existe uma estrutura pré-definida; um bom exemplo é um arquivo de textos. Os *arquivos "flat"* consistem de uma seqüência ordenada de registros (unidades de dados), que podem ou não ser de tamanhos e tipos variáveis. Os *arquivos hierárquicos* além de possuírem registros identificáveis (por chaves), têm uma estrutura em árvore associada, o que permite a recuperação rápida de registros aleatórios.

Os arquivos na área de armazenamento são organizados na forma de unidades de dados (UDs). Cada UD é um objeto de dados com um tipo (escalar, vetor ou conjunto) e contém elementos chamados elementos de dados. A cada elemento de dados está associada uma sintaxe abstrata (caracter, octeto, inteiro, etc); todos os elementos em uma UD estão interrelacionados.

Primitivas de Serviços

As primitivas de serviços de usuários associadas ao FTAM estão agrupadas em "***regimes***" [Bla91]. Ou seja, cada regime possui um conjunto de primitivas de serviços. Abaixo relacionamos algumas das principais primitivas de serviço disponíveis:

- *Conexão de aplicação (associação)*: estabelece as informações de autorização e contabilização necessárias para assegurar as operações na área de armazenamento.

- *Seleção de arquivos*: identifica (ou cria) um arquivo único ao qual as operações em fases subseqüentes irão se referenciar. O processo de seleção ocorre em função do nome de um arquivo; as operações em fases subseqüentes irão se referir a este nome.

- *Acesso a arquivos*: estabelece um regime no qual as transferências de arquivos de dados podem ocorrer. Inclui o estabelecimento das necessidades requeridas para a transferência e um contexto de acesso adequado.

- *Transferência de dados*: inclui comandos (leitura, gravação, etc) que se relacionam a ações que podem ser executadas na unidades de dados (UDs). Como ocorre nas outras camadas do RM/OSI, as primitivas de serviços associadas com o FTAM são agrupadas em várias unidades funcionais (UFs) [Bla91]. As UFs requeridas durante uma sessão específica são especificadas na fase de estabelecimento da associação. As UFs suportadas são:

- *Kernel (ou núcleo)*: fornece as facilidades para o estabelecimento de associação entre aplicações e o seu término. Inclui ainda a seleção de arquivos (e a liberação da seleção) e a abertura (fechamento) de arquivos.

- *Leitura*: fornece facilidades para a leitura de arquivos tratando grandes volumes de dados ("bulk") e a leitura de UDs individualmente.
- *Gravação*: fornece facilidades para gravar arquivos com grandes volumes de dados ("bulk") ou a gravação de UDs individualmente.
- *Acesso a arquivos*: fornece facilidades para localizar e remover arquivos individualmente.
- *Gerenciamento limitado de arquivos*: fornece facilidades para a criação e a remoção de arquivos, e para a leitura de atributos.

3.3.4. Manipulação e Transferência de Serviços (JTM - Job Transfer and Manipulation)

O serviço **JTM** (*Job Transfer and Manipulation*) ([Hal93b] [HM91]) é um conjunto de elementos de serviços de aplicação (ESAs) específico, localizado em diferentes sistemas abertos. Em conjunto, as entidades JTM formam um *fornecedor de serviços* JTM. Assim como ocorre com o protocolo FTAM, o qual não implementa de fato um serviço de arquivos (isto é, ele somente fornece um ambiente através do qual um sistema de arquivos real pode ser acessado e gerenciado de uma maneira aberta), o JTM também fornece somente um ambiente onde documentos relacionados a serviços (jobs), conhecidos como especificações de serviços, podem ser transferidos entre sistemas reais (abertos) e os serviços executados por eles. De fato, o tipo do serviço transferido é transparente para o fornecedor de serviços JTM.

O programa de aplicação que submete uma especificação de um serviço OSI, por meio de um elemento de usuário associado, é conhecido como *agência iniciadora*. Uma especificação de serviço define completamente o serviço a ser efetuado. Por exemplo, em uma aplicação JTM simples pode-se relacionar um serviço à especificação de um programa, juntamente com seus dados, para serem rodados em um computador remoto, ou então, um documento pode ser impresso através de um processo de aplicação em um servidor de impressão remoto. A especificação de serviço inicial pode se espalhar por outras especificações de serviços associadas (são os chamados *sub-serviços* ou "*sub-jobs*")

O programa de aplicação que realmente executa um serviço é conhecido como *agência de execução* [Dav89] enquanto o programa que recebe os requisitos do fornecedor de serviços JTM, para fornecer informações ou dados relacionados a um determinado serviço, é conhecido como *agência fonte* [Dav89], por exemplo, um sistema de arquivos local. É também importante que se note que o tempo entre a submissão de uma especificação de serviço e o término do serviço pode ser muito longo em alguns casos, de tal forma que a agência iniciadora pode especificar que um programa acompanhe o progresso deste serviço enquanto ele estiver rodando. Este programa é conhecido como *monitor de serviços* [Sta93b]. Assim, sempre que um evento significativo ocorrer durante a vida de um serviço, o fornecedor de serviços JTM irá criar um *documento de relato* e o enviará para o monitor de serviços que estiver associado. A agência iniciadora pode em seguida fazer perguntas a respeito do estado deste serviço. Finalmente, após a agência de execução ter completado todo o trabalho associado com a especificação de serviço, ela submete um documento para uma *agência finalizadora* ("sink agency").

Pode-se concluir portanto que o JTM está mais envolvido principalmente com a movimentação de documentos (que dizem respeito aos serviços) entre os programas de aplicação.

3.3.5. MOTIS (Message Oriented Text Interchange Standard)

MOTIS *(Message Oriented Text Interchange Standard)* ([Qua90] [FA94] [VN90b] [Bla91]) corresponde ao sistema de correio-eletrônico da ISO e tem um papel similar ao do protocolo SMTP na pilha de protocolos TCP/IP. Na prática, o MOTIS é um sistema completo de transferência de mensagens tipo de correio e não especificamente um protocolo. Ele é também conhecido como o *sistema de tratamento de mensagens* (**MHS** - *Message Handling System*) da ISO e é baseado no ***serviço público de tratamento de mensagens X.400*** definido pelo ITU-T (International Telecommunications Union - Telecommunications - antigo CCITT) [VN90b].

A recomendação **X.400** do ITU-T foi definida para fornecer um serviço de correspondências eletrônicas internacional que seria similar em princípio ao próprio sistema de correios tradicional (com entrega por carteiros, etc). Ele consiste de um conjunto de protocolos, cada um dos quais executa uma função específica com relação ao sistema de tratamento de mensagens como um todo. As várias entidades (e seus protocolos associados) que formam a recomendação X.400 estão mostradas na figura 3.6 a seguir.

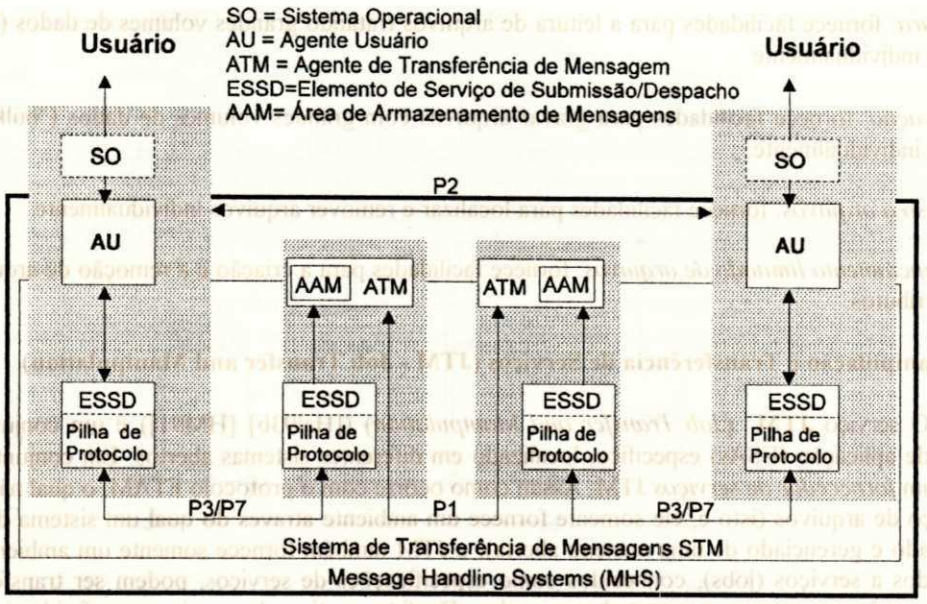


Figura 3.6 O modelo funcional da recomendação X.400 do ITU-T [Bla91]

A interface de usuário para o sistema é um terminal (pode ser um computador pessoal, uma estação de trabalho, etc) que tenha capacidade de processamento suficiente e capacidade de memória para permitir que um usuário crie interativamente uma mensagem (uma correspondência) e possa ler ou passear ("browse") pelas mensagens que ele tenha recebido. Esta última aliás é uma das funções do agente usuário (AU). Adicionalmente, como pressupõe-se que o MHS deverá ser usado por uma grande variedade de tipos de mensagens, o AU deve estar apto a comunicar-se com AUs parceiros em terminais remotos, de tal forma que as mensagens transferidas possuam o mesmo significado em ambos os terminais. Esta função é efetuada pelo *protocolo P2*.

Depois de uma mensagem ter sido devidamente preparada pelo usuário, e o AU ter adicionado suas próprias informações de protocolo, ela é passada adiante para um elemento de serviço adicional chamado de *elemento de serviço de submissão e Despacho* (**ESSD**). A função do ESSD é controlar a submissão e recepção de mensagens de/para o equivalente a uma agência local dos correios. Esta agência local é conhecida como *agente de transferência de mensagens* (ATM). Um protocolo específico é definido para controlar a transferência de mensagens entre os ESSD e a sua ATM local. Este protocolo é conhecido como *protocolo P3*, e cuida dos procedimentos de submissão e entrega de correspondências, assim como de questões como a cobrança pelos serviços.

Os AUs comunicam-se uns com os outros por meio de nomes globais únicos. Assim, ao receber uma mensagem, a ATM local deve primeiramente executar uma tradução do tipo nome-para-endereço. A estrutura dos nomes e endereços são definidas pela recomendação X.500 do ITU-T ([Cas93] [Bla91] [BRI94b]). Associado às ATMs existe um agente de serviços de diretórios que executa a função de tradução.

Depois de a ATM ter obtido os endereços, ela continua com a criação do equivalente a um envelope eletrônico, adicionando os endereços tanto do originador (remetente) quanto do recebedor (destinatário), juntamente com outras informações específicas do *elemento de serviço de transferência de mensagens* (**STM**), que serão adicionadas ao cabeçalho da mensagem recebida de um AU. A ATM em seguida envia a mensagem utilizando o protocolo apropriado, conhecido como *protocolo P1*. A figura 3.7 a seguir mostra o esquema de troca de mensagens entre usuários.

Ao receber uma mensagem endereçada a um de seus AUs (terminais), a ATM irá tentar entregar esta mensagem ao ESSD no terminal que estiver usando o protocolo P3. Como o terminal de usuário estará quase sempre localizado remotamente com relação à ATM, existe a possibilidade de ele estar desligado ou fora de serviço. Para tratar estas situações, a ATM possui uma *área de armazenamento de mensagens* (**AAM**) associada. Assim, no caso de um AU não estar disponível, a mensagem será depositada na AAM para ser entregue mais tarde. Um quarto protocolo, conhecido como protocolo P7, é então usado para controlar a interação entre um usuário, através de seu AU, e a AAM local, permitindo a recuperação de quaisquer mensagens em espera.

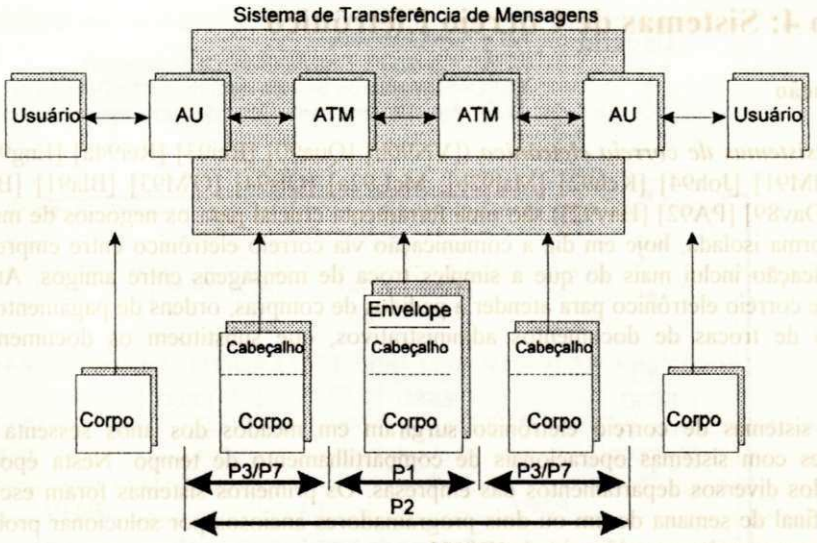


Figura 3.7 O esquema de troca de mensagens entre usuários [Bla91]

Todos estes conceitos dizem respeito à recomendação X.400 para troca de mensagens públicas do ITU-T. No entanto, o MHS ISO é totalmente baseado nesta recomendação, de forma a garantir a máxima compatibilidade entre os sistemas abertos ([VN90b] [Gra91]).

Capítulo 4: Sistemas de Correio Eletrônico

4.1. Introdução

Os ***sistemas de correio eletrônico*** ([VN90b] [Qua90] [Rei93] [Rei94a] [Eng94a] [ES93] [ES94] [FA94] [Hal93b] [HM91] [Joh94] [Keh92] [Mal92b], McL92a] [Ols94] [O'M93] [Bla91] [BRI94b] [Cas93] [Com93a] [Com91a] [Dav89] [PA92] [Hay92]) são uma ferramenta crucial para os negócios de muitas empresas. Inicialmente usados de forma isolada, hoje em dia a comunicação via correio eletrônico entre empresas dispersas é algo trivial. Esta comunicação inclui mais do que a simples troca de mensagens entre amigos. Atualmente podem ser vistas aplicações de correio eletrônico para atender a pedidos de compras, ordens de pagamento, transferências bancárias, e outros tipos de trocas de documentos administrativos, que substituem os documentos equivalentes em papel [Wat94a].

Os sistemas de correio eletrônico surgiram em meados dos anos sessenta baseados em sistemas de computadores com sistemas operacionais de compartilhamento de tempo. Nesta época começaram a proliferar terminais pelos diversos departamentos das empresas. Os primeiros sistemas foram escritos por acaso, a partir de projetos de final de semana de um ou dois programadores anciosos por solucionar problemas locais. Tais sistemas não apresentavam nenhuma uniformidade [ES93].

A substituição de memorandos em papel por uma versão eletrônica dos mesmos foi a idéia usada por sistemas de automação de escritórios como o PROFS (atualmente conhecido como Office Vision) da IBM [Rei93]. No entanto, estas primeiras versões de sistemas de correio eletrônico apresentavam uma desvantagem crucial. Enquanto memorandos escritos em papel podiam ser enviados para qualquer pessoa, as mensagens de correio eletrônico ficavam limitadas aqueles empregados que tivessem um terminal sobre sua mesa e estivessem conectados ao mesmo computador central.

Quando em 1969 o governo americano deu início à ARPANET, pesquisadores em universidades e outros locais passaram a trocar dados eletronicamente uns com os outros [Eng94a]. O sistema de correio eletrônico passou rapidamente a ser usado de diversas formas, permitindo também a transferência de outros tipos de informações, como por exemplo programas executáveis e arquivos de dados, empacotados como mensagens de correio eletrônico.

Com a chegada das estações Unix, e com a proliferação de microcomputadores conectados em redes locais, a possibilidade de que grandes empresas passassem a empregar sistemas de correio eletrônico para facilitar a comunicação entre seus empregados foi sensivelmente incrementada [Ham93]. A figura 4.1 dá uma idéia do quanto os sistemas de correio eletrônico têm proliferado nas empresas americanas, segundo uma pesquisa realizada pelo The Yankee Group e publicado em [Rei93].

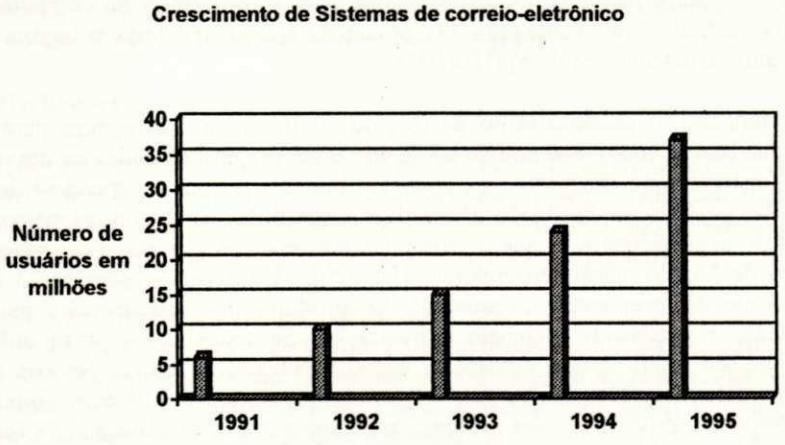


Figura 4.1. Estimativa de uso de sistemas de correio eletrônico nos EUA [Rei93]

Atualmente, os sistemas de correio eletrônico estão disponíveis para milhões de pessoas por todo o mundo, e mensagens são freqüentemente trocadas entre diversas plataformas operacionais de diferentes empresas ([Eng94a] [ES93]).

4.1.1. Correio Eletrônico na Prática

Bons exemplos de uso dos sistemas de correio eletrônico podem ser extraídos da experiência de grandes empresas americanas. Recentemente, a Digital Equipment Corp., que possui filiais em Massachusetts, no Arizona, no Colorado, em Singapura e na Alemanha, executou um projeto para um novo dispositivo de disco. Engenheiros em todas as filiais participaram do projeto, e a maioria deles jamais encontrou-se pessoalmente, ou mesmo trocou conversas por telefone. Ainda assim, a Digital estima que este grupo disperso tenha conseguido terminar o projeto com um ano de antecedência, e com uma quantidade 40 % menor de pessoas envolvidas. O segredo do sucesso: o uso de sistemas de correio eletrônico, que chega a alcançar virtualmente qualquer empregado em uma grande empresa [Hay92].

Os sistemas de correio eletrônico suportam diferentes tipos de serviços. O correio eletrônico privado é estabelecido pessoa a pessoa, e pode envolver arquivos de imagens gerados de projetos CAD (Computer Aided Design), imagens digitalizadas ou textos [HM91]. Os sistemas de correio eletrônico para distribuição em grandes redes originam-se de um indivíduo, mas são enviados para diversas pessoas, muitas vezes substituindo os memorandos interdepartamentais. A conferência eletrônica permite que os usuários identifiquem tópicos, geralmente técnicos, que facilitem seu trabalho diário [Cas93].

As vantagens do uso de sistemas de correio eletrônico são inúmeras, além da vantagem óbvia de permitir a independência do telefone [Der92a]. A flexibilidade fornecida tem como imediata conseqüência uma melhoria de produtividade. As empresas podem operar com pequenas equipes, por exemplo, um especialista pode trabalhar por meio período com duas equipes de projetos separados por muitos quilômetros de distância [Sto93b].

Um dos grandes benefícios fornecidos pelos sistemas de correio eletrônico é a possibilidade de conseguir-se informações detalhadas em uma empresa. Por exemplo, em equipes de projetos, quando surge um problema que atinge a todos no projeto, pode-se emitir uma mensagem de difusão do tipo "alguem sabe a respeito de tal assunto?" para todos que estiverem na rede. Sugestões de soluções comumente retornam em algumas horas [Rad93a].

Na empresa Sun Microsystems organizou-se um projeto de um novo compilador para estações SPARC com equipes de projetistas de supercomputadores nos EUA e em algumas cidades russas, como Moscou, São Petersburgo, e Novosibirski. Os sistemas de telefonia russos são pouco confiáveis, fora os problemas com a linguagem falada, de tal forma que estes engenheiros estão colaborando uns com os outros quase que exclusivamente por meio de correio eletrônico [ES94].

4.1.2. Correio Eletrônico para Produtividade de Grupos de Trabalho

Os tipos de aplicações que fazem melhor uso das funcionalidades oferecidas pelos sistemas de correio eletrônico são inúmeras. Uma é particularmente importante para os ambientes da computação distribuída: são os programas aplicativos centrados em mensagens. Os principais aplicativos desta categoria são os programas de agendas e escalonamento de pessoal ([Sto93b] [Ham93]).

Em muitas empresas, o escalonamento de três ou quatro pessoas ocupadas para uma reunião, exige a preparação de certas facilidades como uma sala de conferência, um projetor de slides ou um retroprojetor. Esta pode ser uma experiência frustrante que certamente irá consumir bastante tempo, e dependerá de uma série de ligações telefônicas. Se uma das pessoas ou facilidade não estiver disponível quando outras pessoas e outras facilidades estiverem, uma série de negociações terá que ocorrer. Os matemáticos chamam este método de tratar diversos fatores desconhecidos de "Aproximação Progressiva" [Der92b]. Estas negociações serão tarefas frustrantes para quem estiver encarregado de fazer todos os contatos. Os produtos de escalonamento para redes locais buscam simplificar estas tarefas, e geralmente eliminam a frustração. Se todos na empresa utilizarem o software de escalonamento, uma pessoa estará apta a acessar as agendas públicas de outras pessoas e as listas de recursos reservados. A tarefa de determinar tempo livre para as pessoas que devam comparecer a uma reunião fica bem mais fácil. Este processo respeita a privacidade das pessoas - a pessoa que estiver planejando a reunião só terá acesso às informações necessárias na agenda alheia para saber se aquela pessoa dispõe de tempo livre e quando.

Os programas de escalonamento variam com relação à forma de apresentação do tempo livre das pessoas. Alguns mostram gráficos de horários conflitantes e horários disponíveis. Outros usam uma página de agenda - outros ainda utilizam texto com explicações para mostrar as opções de escalonamento. Outra variação ocorre na forma como são confirmados os eventos propostos. Os aplicativos mais simples, assumem que se um evento couber em um determinado horário, as pessoas envolvidas irão comparecer. Outros programas buscam a confirmação, enquanto outros vão ainda mais longe, como por exemplo, ligam-se a um sistema de correio eletrônico para criarem mensagens de notificação e confirmação.

Para aliviar este problema, podem ser usados ***backbones de correio-eletrônico*** ([BRI94b] [Com91b] [Der92b] [Hun92a] [Mal92a], Mal92b] [MR88b]). Um backbone fornece a conexão de diferentes sistemas por meio de um protocolo único. Cada sistema possui um gateway para este backbone de protocolo comum. Os gateways mais comuns são exatamente gateways para os protocolos SMTP e para X.400. Os sistemas de backbone com protocolos padrões (X.400 ou SMTP) simplificam a administração e a manutenção de redes com muitos sistemas de correio-eletrônico. Vários destes gateways existem disponíveis no mercado, porém a um custo ainda bastante elevado. Na figura 4.2 ilustramos alguns dos mais conhecidos gateways de correio eletrônico disponíveis no mercado, formando em conjunto, um exemplo para um backbone de correio eletrônico.

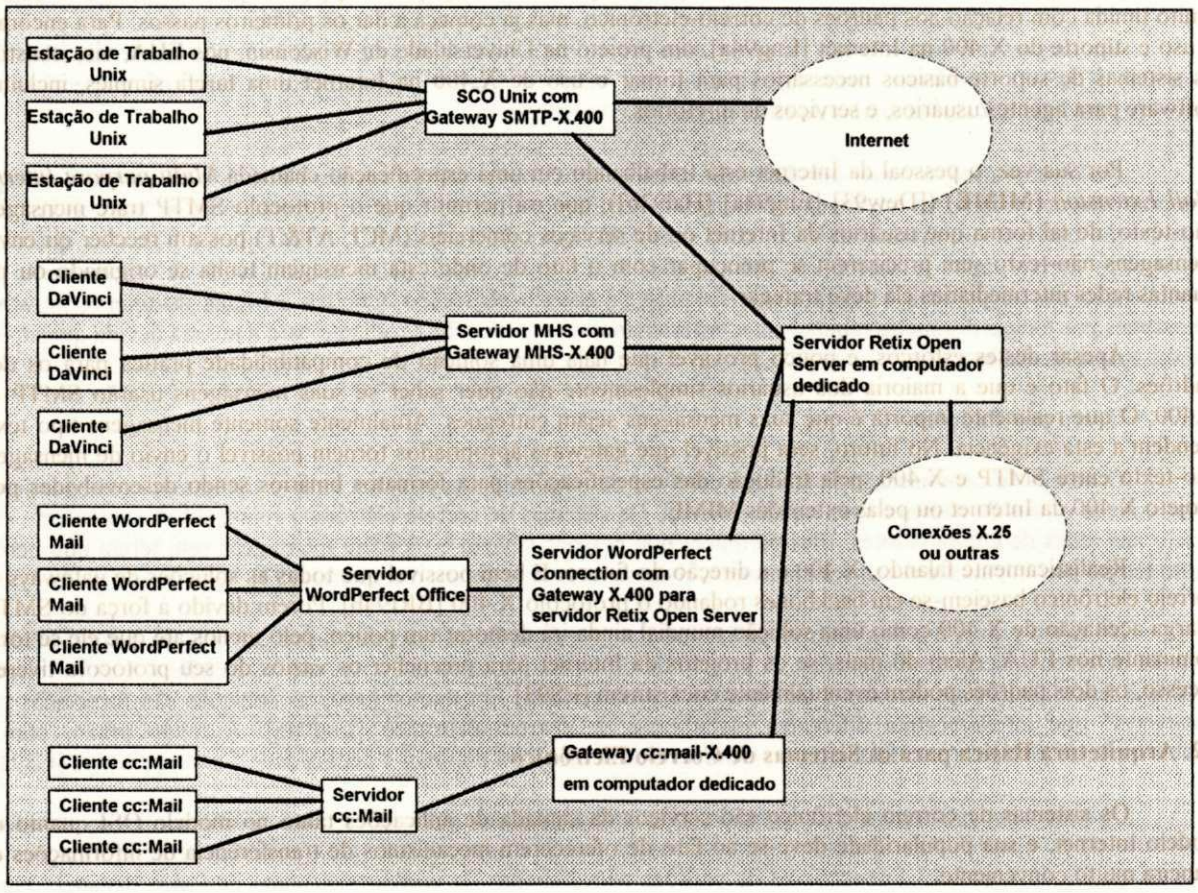


Figura 4.2. Backbones de correio eletrônico [ES93]

4.1.3.2. SMTP da Internet

O poder do **SMTP** (*Simple Mail Transfer Protocol*) ([FA94] [HM91] [Cas93] [Com93a] [Com91a] [Hal93b] [Qua90]) reside na excelente infra-estrutura de suporte da Internet. É um protocolo que, apesar de bastante difundido e já estabelecido, mostra-se fora de seu tempo. Por exemplo, os formatos de mensagens, incluindo seus endereços devem ser representados por bytes de sete bits, na forma de textos ASCII. Mensagens não-texto, como voz, fax, imagens, EDI, código binário, exigem bytes de oito bits, de tal forma ques estas mensagens devem ser codificadas de alguma maneira em bytes de sete bits para poderem trafegar via SMTP.

Apesar de seus problemas, na prática, SMTP é o padrão "de facto" para protocolos de correio-eletrônico, pois graças à infra-estrutura da Internet e seus serviços de suporte (como por exemplo: sistemas de diretórios, auxílio para problemas de endereçamento, e uma vasta gama de sistemas de correio eletrônico disponíveis), o SMTP ainda deve existir por vários anos. Seu uso é mais largamente difundido nos EUA, principalmente devido à falta de uma infra-estrutura Internet confiável na Europa ([Gre94a] [HM91]).

4.1.3.3. X.400 da ISO/ITU-T

O ***protocolo X.400*** ([HM91] [Hal93b] [VN90b] [Bla91]) é tecnicamente superior ao SMTP. É um protocolo especificado por um comitê do ITU-T, sendo posteriormente endossado pela ISO, solucionando a maioria dos problemas encontrados no SMTP. O padrão original foi liberado em 1984. Em 1988 foi expandido para incluir formatos para mensagens binárias não-texto, e em 1992 foi revisto para incluir formatos para EDI (Electronic Data Interchange) [Rad93a].

Apesar de seu uso ser mais largamente difundido na Europa, mesmo nos EUA alguns fornecedores de serviços de correio eletrônico, com as empresas MCI e AT&T, utilizam o X.400 como protocolo de backbone [Gre94a] [ES94]. A versão usada é de 1984, a qual não especifica formatos exatos para diferentes tipos de mensagens binárias, de tal forma que cada fornecedor do serviço teve que especificar seus próprios formatos binários, e conseqüentemente, mensagens trocadas entre estes prestadores de serviços limitam-se a textos.

4.1.3.4. A Cooperação entre Internet e a ISO/ITU-T

A cooperação entre a **IETF** (*Internet Engineering Task Force*) ([Keh92] [Qua90]) e a **ISO/ITU-T** ainda é muito tímida com relação aos padrões de correio-eletrônico, mas já começa a dar os primeiros passos. Para encorajar o uso e suporte do X.400 na Internet [Eng94a], um projeto na Universidade de Wisconsin, nos EUA, visa construir os sistemas de suporte básicos necessários para tornar o uso de X.400 na Internet uma tarefa simples, incluindo software para agentes/usuários, e serviços de diretórios.

Por sua vez, o pessoal da Internet está trabalhando em uma especificação chamada *Multi-purpose Internet Mail Extension* (**MIME**) ([Dew93] [Eng94a] [Hal93b]), que irá permitir que o protocolo SMTP trate mensagens não-texto, de tal forma que usuários da Internet ou de serviços comerciais (MCI, AT&T) possam receber ou enviar mensagens não-texto sem precisarem se preocupar com o fato de onde esta mensagem tenha se originado ou por quantas redes intermediárias ela deve trafegar.

Apesar destes esforços, é pouco provável que haja uma solução de compatibilidade prática para os dois padrões. O fato é que a maioria dos usuários simplesmente não quer saber se suas mensagens usarão SMTP ou X.400. O que realmente importa é que suas mensagens sejam entregues. Atualmente somente mensagens tipo texto atendem a esta exigência. No futuro, será possível que gateways apropriados tornem possível o envio de mensagens não-texto entre SMTP e X.400, pela tradução das especificações para formatos binários sendo desenvolvidas pelo projeto X.400 da Internet ou pelas extensões MIME.

Realisticamente falando, X.400 é a direção do futuro. É bem possível que todas as soluções de gateways de correio eletrônico baseiem-se em backbones rodando o protocolo X.400 [BRI94b]. Porém devido à força do SMTP, a larga aceitação de X.400 como uma solução mundial ainda irá demorar um pouco, pelo menos até que ele se torne dominante nos EUA. Além do mais, se os projetos da Internet para preencher os vazios do seu protocolo tiverem sucesso, os dois padrões podem eventualmente coexistirem [ES93].

4.2. Arquitetura Básica para os Sistemas de Correio Eletrônico

Os sistemas de correio eletrônico são serviços da camada de aplicações tanto no modelo OSI quanto no modelo Internet, e sua popularidade deve-se ao fato de oferecerem mecanismos de transferência de informações de maneira muito conveniente.

Nos protocolos de redes mais simples, os pacotes contendo as mensagens são enviados para um destinatário, usando-se temporização ("time out") e retransmissão para segmentos individuais, se não forem retornados os avisos de recebimento ("acknowledgements"). Com os protocolos para correio-eletrônico, no entanto, o sistema deve fornecer os meios que independam de falhas em conexões da rede ou do fato de estações remotas estarem desligadas ([HM91] [Com91a]). Um remetente não pode esperar que uma estação remota esteja disponível para continuar com o seu trabalho, nem deseja que a aplicação aborte simplesmente porque a comunicação com a máquina remota está temporariamente interrompida.

Para tratar estas entregas de mensagens "postergadas", os sistemas de correio eletrônico utilizam uma técnica de "spooling". Quando um usuário envia uma mensagem de correio-eletrônico, o sistema armazena uma cópia desta mensagem em uma área de armazenamento privada (são justamente as áreas de spool, ou filas de spool) juntamente com a identificação do remetente, destinatário, máquina destinatária, e data e hora em que o envio da mensagem foi requisitado. A seguir o sistema inicia a transferência da mensagem para a máquina remota, por meio de um processo rodando na retaguarda ("background"), de forma a permitir que o remetente continue com suas atividades normais de uso do computador. O recebimento de uma mensagem ocorre da mesma maneira. Assim, tudo que o usuário precisa é executar uma interface para depositar ou receber mensagem da área de spool. Todas as transferências são tratadas por processos submetidos na retaguarda. Observação: as áreas de spool são duas, a área de mensagens de entrada recebidas de remetentes remotos, e a área de mensagens de saída, enviadas para destinatários remotos. A figura 4.3 ilustra o conceito de filas de spool em sistemas de correio-eletrônico.

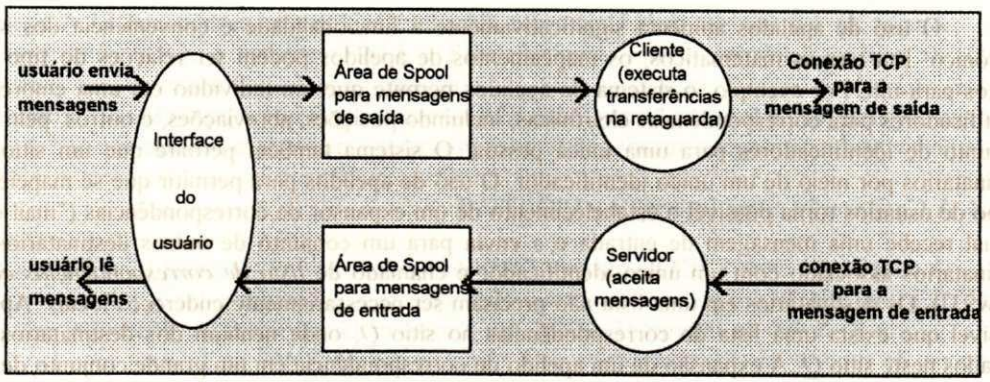


Figura 4.3. Componentes de um sistema de correio eletrônico [Com91a]

O modelo empregado pelos sistemas de correio eletrônico segue o paradigma cliente/servidor para o desenvolvimento de aplicações distribuídas [Cas93]. O processo que cuida da transferência de mensagens na retaguarda é chamado de *processo cliente*. Ele mapeia a máquina destinatária por meio de um endereço de rede, e estabelece uma conexão com o *processo servidor* de mensagens na máquina destinatária. Se a conexão tiver sucesso, o processo de transferência passa uma cópia da mensagem para o servidor remoto, o qual irá armazenar esta cópia na sua área de spool. Quando cliente e servidor concordam com o fato da cópia ter sido transferida e armazenada, o cliente remove a cópia local de sua área de spool.

Caso a conexão não possa ser estabelecida, ou ocorra uma falha em uma conexão que tenha sido previamente estabelecida, o processo de transferência cuida de anotar a hora em que ele tentou a entrega e a seguir termina sua execução. O processo de transferência da retaguarda vasculha temporariamente a área de spool em busca de mensagens não despachadas. Quando a encontra, ou sempre que uma nova mensagem é depositada na área de spool, ele tenta despachá-la para seu destinatário. Se o software detecta que não consegue enviar uma mensagem após um longo período de tentativas (por exemplo, 3 dias), ele então envia uma mensagem de erro para o remetente [Dav89].

4.2.1. Caixas Postais e Apelidos ("Aliases")

Existem três fatos importantes descritos na arquitetura acima que merecem ser destacados: em primeiro lugar, os usuários especificam os destinatários por meio de dois parâmetros que especificam o *nome da máquina destinatária* e um *endereço de caixa postal* nesta máquina [FA94].

Em segundo lugar, os nomes usados nestas especificações são independentes de outros nomes assinalados para outras máquinas na rede. Geralmente, os endereços de caixas postais são os mesmos usados para a identificação de login dos usuários em cada máquina, e os nomes das máquinas destinatárias são os mesmos definidos para os domínios da rede para estas máquinas, porém não de maneira obrigatória. É possível que se assinale o identificador chefe-de-departamento para uma caixa postal, a qual irá referenciar quem quer que seja o chefe do departamento em um determinado momento. É necessário portanto que exista um servidor de nomes (ou de diretórios) que suporte uma base de dados e uma linguagem de consulta para tratar os endereços destinatários, tornando possível que se separem os nomes de destinos para correspondências eletrônicas dos nomes usuais de domínios usados para identificar os nós de uma rede.

E, em terceiro lugar, o modelo apresentado na figura 4.3 anterior é muito simples, e não leva em consideração a forma de tratar mensagens, ou a maneira de como passá-las adiante, o que incluí mensagens enviadas de um mesmo usuário para outro na mesma máquina, e mensagens que chegam em uma máquina e devem ser passadas adiante para outras. Para tratar estes casos existem algoritmos específicos para a expansão de apelidos e para a conexão ("mail forwarding") de mensagens [Com91a], os quais veremos a seguir.

4.2.2. A Expansão de Apelidos e a Conexão de Mensagens

A maioria dos sistemas fornece software para conexão de mensagens que inclui um mecanismo de expansão de apelidos de correspondências. Um software para conexão de mensagens permite que o sítio local mapeie identificadores usados em endereços de mensagens para novos endereços (um ou mais) [Dav89]. Geralmente, após a preparação de uma mensagem, e a identificação do destinatário ser estabelecida, o programa para interface de mensagens consulta os apelidos locais para substituir o nome do destinatário pelo nome que foi mapeado, antes de passar a mensagem para o processo de transferência na retaguarda. Destinatários que não tenham sido mapeados (não possuem apelidos) não são alterados. Similarmente, o sistema de correio eletrônico usado irá utilizar os apelidos para mapear nomes de destinatários locais para mensagens recebidas de remetentes remotos.

O uso de apelidos aumenta significativamente a funcionalidade e conveniência dos sistemas de correio-eletrônico. Em termos matemáticos, os mapeamentos de apelidos podem ser relações do tipo um-para-muitos ou muitos-para-um. Por exemplo, o sistema de apelidos permite que um indivíduo em uma empresa possua múltiplos identificadores para correspondências eletrônicas, incluindo posições, abreviações, e outros, pelo mapeamento de um conjunto de identificadores para uma única pessoa. O sistema também permite que um sítio associe grupos de destinatários por meio de um único identificador. O uso de apelidos para permitir que se mapeie um usuário em um grupo de usuários torna possível o estabelecimento de um expansor de correspondências ("mail exploder") [HM91], o qual recebe uma mensagem de entrada e a envia para um conjunto de outros destinatários. Este conjunto de destinatários associado com um único identificador é chamado de *lista de correspondências eletrônicas* ([HM91] [Dew93]). Os destinatários em uma lista não precisam ser necessariamente endereços locais. Apesar de incomum, é possível que exista uma lista de correspondências no sítio Q, onde nenhum dos destinatários nesta lista estejam alocados neste sítio Q. A expansão de um apelido de correspondência em um grande conjunto de destinatários é uma técnica muito popular e largamente utilizada. A Figura 4.4 ilustra os componentes de um sistema de correio eletrônico que suporta apelidos de correspondência e expansão de listas.

Tanto mensagens de entrada quanto de saída passam pelo mecanismo de expansão de apelidos. Assim, se a base de dados de apelidos especificar que o endereço de correspondência *x* tem uma entrada no mapa correspondente a *y*, o mecanismo de expansão de apelidos irá regravar o endereço destinatário *x*, substituindo-o por *y*. O programa de expansão de apelidos irá em seguida verificar se y especifica um endereço local ou remoto, o que permitirá que se determine em qual das filas de mensagens (fila de entrada ou fila de saída) a correspondência deverá ser armazenada (é claro que se for especificado um endereço remoto, deseja-se enviar esta mensagem para este endereço remoto, e não recebê-la localmente).

A expansão de apelidos pode ser muito perigosa. Suponhamos que dois sítios estabeleçam apelidos conflitantes. Por exemplo, assuma que o sítio *S1* irá mapear o endereço *estação123* como o endereço *estaçãoBD* no sítio *S2*. Assuma também que o sítio *S2* irá mapear o endereço *estaçãoBD* como o endereço *estação123* no sítio *S1*. Uma mensagem enviada para o endereço estação123 no sítio S1 irá ficar sendo enviada de um sítio para o outro em "loop". De maneira similar, se o gerente da rede no sítio S1 mapear acidentalmente a identificação de login de um dos usuários neste sítio, para um endereço em outro sítio, este usuário simplesmente não receberá mensagens de correio-eletrônico. A correspondência será passada para outro usuário, ou, se o apelido especificar um endereço inválido, os remetentes irão receber mensagens de erro.

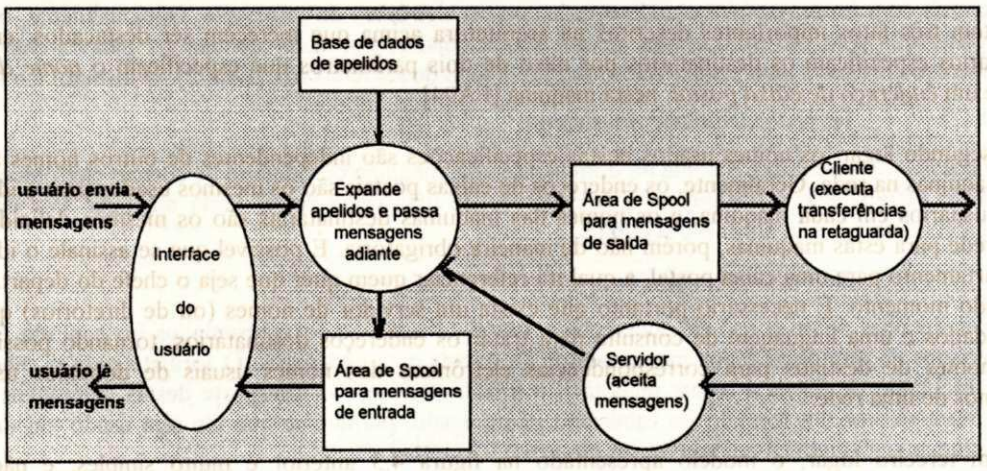


Figura 4.4. Um sistema de correio eletrônico completo [Com91a]

4.3. Serviços de Correio Eletrônico na Internet: SMTP

Já vimos que o principal objetivo do modelo de interconexão de redes de computadores baseado no TCP/IP é a busca pela interoperabilidade [OSF91c] entre os mais diversos sistemas e redes de computadores. Para estender a interoperabilidade de seus serviços de correio-eletrônico, o TCP/IP divide seu protocolo para serviços de correio eletrônico em dois conjuntos de especificações [FA94]. O primeiro destes conjuntos e especificações define o formato para mensagens de correio-eletrônico. Esta definição é conhecida como "RFC822: Standard for the Format of ARPA-Internet Text Messages" [MR88b], e representa um *requisito para comentários* do processo de padronização da IETF (Internet Engineering Task Force), que é o comitê especial que define os padrões do TCP/IP. O segundo conjunto de especificações contém os detalhes para a troca eletrônica de correspondências entre dois sistemas de computadores. A vantagem em dividir-se as especificações é a possibilidade de serem construídos gateways que interconectem as inter-redes TCP/IP com outros sistemas de correio eletrônico de fornecedores de soluções proprietárias, porém, com o uso dos mesmos formatos de mensagens.

4.3.1. O formato de mensagens da RFC822

Uma correspondência eletrônica, em qualquer que seja o sistema usado, é sempre dividida em duas partes: um cabeçalho e um corpo, separados por uma (e somente uma) linha em branco. A RFC822 [FA94] especifica o formato exato para os cabeçalhos de mensagens, assim como a interpretação semântica para cada um de seus campos; o corpo da mensagem é definido pelo remetente. Apresentamos na figura 4.5 um exemplo de mensagem.

```
From: helvecio@dsc.ufpb.br
To: brasileiro@dsc.ncu.uk
Cc: jacques@di.ufal.br
    suzana@dc.uece.br
Subject: Nova Referencia Bibliografica
Date: Mon, 13 August 94 15:13:38

Prezado Brasileiro,

Recebi a bibliografia que você enviou-me a respeito das novas implementações dos algoritmos de
controle de concorrência em ambientes distribuídos. Achei particularmente importante a nova
abordagem dada aos algoritmos de serialização com rótulos de tempo, principalmente o novo enfoque
para o protocolo de commit com duas fase (2PC). Agradeço-lhe a atenção, estamos aguardando o seu
retorno para breve.

Um abraço,      Helvécio
```

Figura 4.5. Exemplo de uma mensagem de correio eletrônico

A especificação define que os cabeçalhos devem conter um texto passível de ser lido, dividido em linhas que consistem de uma palavra-chave seguida de dois-pontos e seguidas de um valor. Algumas palavras-chave são obrigatórias, enquanto que outras são opcionais, e se outras ainda aparecerem (que não sejam aquelas opcionais ou obrigatórias), simplesmente não serão interpretadas. Por exemplo, o cabeçalho deve conter uma linha que especifique um destinatário. A linha que começa com *To:* contém o endereço de correio eletrônico para o destinatário desejado. A linha que começa com *From:* contém o endereço de correio eletrônico do remetente. Opcionalmente, o remetente pode indicar endereços para os quais cópias devem ser mandadas, por meio do uso da palavra-chave *Cc:*. Na Internet, os endereços possuem uma sintaxe muito simples e fácil de ser lembrada [Keh92]. O formato é o seguinte:

parte-local@nome-do-domínio

Onde o *nome-do-domínio* é o nome do domínio de uma máquina destinatária de correio eletrônico para onde a mensagem deve ser enviada, e *parte-local* é o endereço de uma caixa-postal nesta máquina. Por exemplo, na Internet o endereço eletrônico do autor é o seguinte:

Helvecio@dsc.ufpb.br

Maiores detalhes sobre o uso de endereços e seu significado serão apresentados no capítulo 5 , quando falamos dos serviços de diretórios e dos mapeamentos de nomes para endereços. O formato para o cabeçalho das mensagens foi escolhido de forma a tornar mais fácil o processamento e o transporte destas mensagens por redes heterogêneas. Mantendo-se um formato de cabeçalho padronizado, permite-se que ele seja usado em uma grande variedade de sistemas (incluindo os computadores pessoais) [Com91a].

4.3.2. As especificações para a troca de mensagens - SMTP

O **SMTP** (*Simple Mail Transfer Protocol)* ([FA94] [BRI94a] [HM91] [Cas93] [Com93a] [Com91a] [Hal93b] [Qua90]) especifica um padrão para a troca de correspondências eletrônicas de um sistema de correio eletrônico em um computador hospedeiro para outro. O padrão especifica o formato exato para as mensagens que um cliente em uma máquina usa para transferir mensagens para um servidor em outra máquina. O protocolo SMTP enfoca especificamente como o sistema de correio eletrônico que está sendo usado passa as mensagens via uma conexão estabelecida entre dois computadores hospedeiros em uma rede. Ele não especifica detalhes de como o sistema de correio eletrônico efetua o recebimento das mensagens de um usuário, ou de como as mensagens de entrada são apresentadas. Além do mais, o SMTP não se preocupa com a localização onde as mensagens serão armazenadas, nem da frequência das tentativas de transmissão. Em outras palavras, o SMTP não é responsável por aceitar mensagens de usuários locais, nem pela distribuição de mensagens recebidas para seus destinatários. Todas estas são responsabilidades do sistema de correio-eletrônico. O interrelacionamento entre o SMTP e o sistema de correio eletrônico local esta mostrado na figura 4.6 .

O início do conteúdo do corpo da mensagem é então indicado pelo cliente, por meio de um comando DATA. O servidor responde com um comando 354 e o cliente então prossegue o envio do conteúdo de sua correspondência eletrônica como uma seqüência de linhas terminadas por uma linha com um ponto único (<CR><LF>.<CR><LF>). O servidor indica o recebimento da última linha por meio de um comando 250. A fase de transferência é então sinalizada pelo cliente por meio de um comando QUIT enviado para o servidor, que retorna um comando 221, após o qual a conexão é desfeita.

4.4. O Sistema de Tratamento de Mensagens ISO/ITU-T: X.400

A **Recomendação X.400** ([HM91] [Hal93b] [VN90b] [Bla91]) do ITU-T, também conhecida como "*Message Handling System*" (**MHS**) teve sua origem a partir de iniciativas de várias organizações nos Estados Unidos que já haviam trabalhado com sistemas de tratamento de mensagens no ambiente da ARPANET [Bla91] (a atual Internet). O trabalho de padronização inicial foi assumido pelo então CCITT (atualmente ITU-T) que publicou a Série X.400 em 1984. A ISO começou a sua colaboração também em 1984, apesar de seus padrões iniciais serem consideravelmente diferentes daqueles do CCITT. Porém vários compromissos foram assumidos entre CCITT e ISO, de tal forma que nos livros azuis publicados em 1988 as versões ISO e CCITT para serviços de tratamento de mensagens passaram a ser praticamente as mesmas [Bla91].

4.4.1. A Estrutura do MHS

A recomendação MHS fornece serviços de mensagens para usuários finais (remetentes e destinatários) com duas características principais: (1) o *Sistema de Transferência de Mensagens* (**STM**) que suporta sistemas independentes de aplicação. Ele é o responsável pelo "envelope" da mensagem, constituindo o meio pelo qual os usuários trocam mensagens; e (2) o *Sistema de Mensagens Interpessoais* (**MIP**) que suporta a comunicação com os serviços do ITU-T e define as interfaces específicas para o MHS. Ele é o responsável pelo conteúdo dos envelopes, utilizando as funcionalidades do STM para enviar e receber mensagens. O MHS considera os serviços do MIP como uma aplicação padronizada.

O MHS utiliza o termo *objetos funcionais* [SOIEC91c] para descrever os componentes do sistema. Por exemplo, os usuários do sistema de tratamento de mensagens e as listas de distribuição de mensagens são classificadas como objetos funcionais primários. Sistemas de transferência de mensagens, agentes de usuários, áreas de armazenamento de mensagens, e unidades de acesso são classificadas como objetos funcionais secundários.

4.4.2. A Arquitetura do MHS

As entidades abaixo são os principais objetos funcionais definidos pelo MHS ([Hal93b] [Bla91] [HM91]):

- *Agentes de usuários* (**AU**)
- *Agentes de transferência de mensagens* (**ATM**)
- *Sistema de transferência de mensagem* (**STM**)
- *Área de armazenamento de mensagens* (**AAM**)
- *Unidades de acesso* (**UA**)

Na figura 4.9 está representado esquematicamente o relacionamento entre os objetos. O *agente de usuário (AU)* é responsável pelo interfaceamento direto com o usuário final. O MHS não define as interações com o usuário final, nem como são processadas ações isoladas, pois o AU é na realidade um processo de aplicação. De fato, o AU cuida de preparar, submeter e receber mensagens para o usuário. Fornece também edição de textos e serviços de apresentação para o usuário final, além de outras atividades, incluindo interações via GUIs, segurança, controle de prioridades, notificação de entrega de mensagens, e distribuição de subconjuntos de documentos. Adicionalmente, podem existir diferentes tipos de AUs fornecendo serviços para usuários MHS. Veremos adiante que todas as mensagens trocadas entre AUs são padronizadas por um único tipo de mensagem interpessoal definida a nível do MIP, chamadas de mensagens P2.

O MHS define também como são as interações dos AUs com o *agente de transferência de mensagens (ATM)*. Os AUs são agrupados em classes pelo MHS, considerando-se os tipos de mensagens que cada grupo pode processar. Os AUs de um grupo são denominados AUs cooperantes.

P1: especifica a comunicação entre os ATMs.
P3: especifica operações de submissão e entrega de mensagens entre ATMs e AUs.
P7: especifica a submissão e recuperação de/para AAM.
P2: especifica o protocolo de mensagens interpessoais entre o MIP e os AUs.

4.4.2.2. Conteúdo e Envelope das Mensagens de Correio Eletrônico

Como mostrado na figura 4.11 , o MHS utiliza o termo *conteúdo* [FA94] para descrever a informação contida na mensagem. É análogo a uma carta convencional. O termo *envelope* [FA94] descreve as informações de controle usadas para efetivar a entrega do conteúdo. É análogo a um envelope usado pelos correios, exceto pelo fato do envelope MHS conter as informações que o ATM precisa para requisitar muitos elementos de serviços que os sistemas de correio manuais não disponibilizam.

O MHS suporta também o conceito de *listas de distribuição* (**LDs**) [Com91a]. Uma LD é similar a uma lista de distribuição em um escritório. Por exemplo, um remetente MHS pode fornecer ao MHS o nome da LD. É tarefa do MHS expandir esta lista para os nomes destinatários, e distribuir uma cópia desta mensagem para cada um dos usuários destinatários.

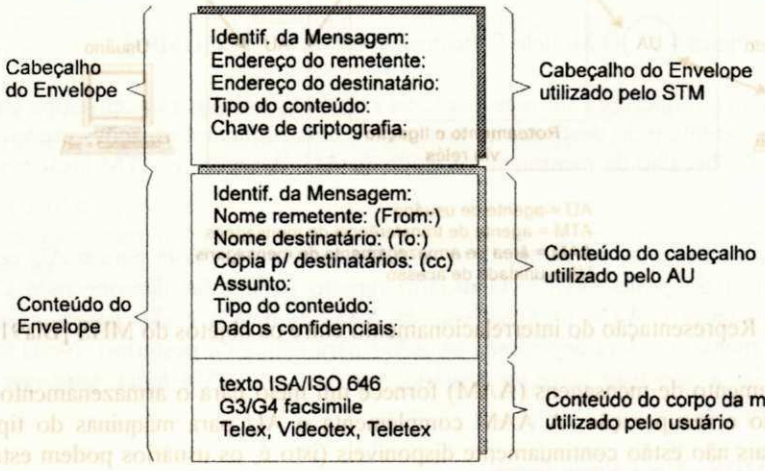


Figura 4.11. A mensagem MHS e seu envelope [Bla91]

4.4.3. Os Detalhes do Modelo Funcional MHS X.400

Depois de vistos os conceitos básicos do MHS e seus componentes mais importantes, vejamos alguns detalhes da transferência de uma mensagem de correio eletrônico de um usuário para outro, passando por todos os objetos funcionais do modelo de arquitetura visto na seção 4.4.2. anterior. Nosso exemplo será dado com relação à figura 4.12 [Hal93b].

Vamos considerar que a interface para o sistema seja uma estação de trabalho com capacidade de processamento e memória suficientes para permitir que o usuário possa criar interativamente uma mensagem de correio eletrônico e possa ler e pesquisar as mensagens que forem recebidas. O serviço de elemento que irá cuidar destas tarefas é o *AU*. O *AU* irá permitir as comunicações entre *AU*s pares via o protocolo *P2*.

Depois de devidamente aprontada, uma mensagem será tratada pelo AU que irá acrescentar a ela informações do próprio protocolo. O AU então irá despachar esta mensagem para um elemento de serviço chamado de *elemento de serviço de submissão e despacho* (**ESSD**), o qual tem a seu encargo a tarefa de submeter/receber mensagem de/para o serviço de correio local chamado de ATM local. Existe um protocolo particular para transferir mensagens entre o ESSD e o seu ATM local, e este é o protocolo P3.

Os AU's comunicam-se uns com os outros por meio de nome globais únicos. Assim, quando uma mensagem é recebida, o ATM local deve primeiramente executar as traduções de nomes para endereços cabíveis (geralmente utilizam-se as convenções estabelecidas pela recomendação X.500 [Qua90] do ITU-T, a qual veremos em detalhes no próximo capítulo. Associado ao ATM local existe um agente de serviços de diretórios que cuida destas funções de traduções.

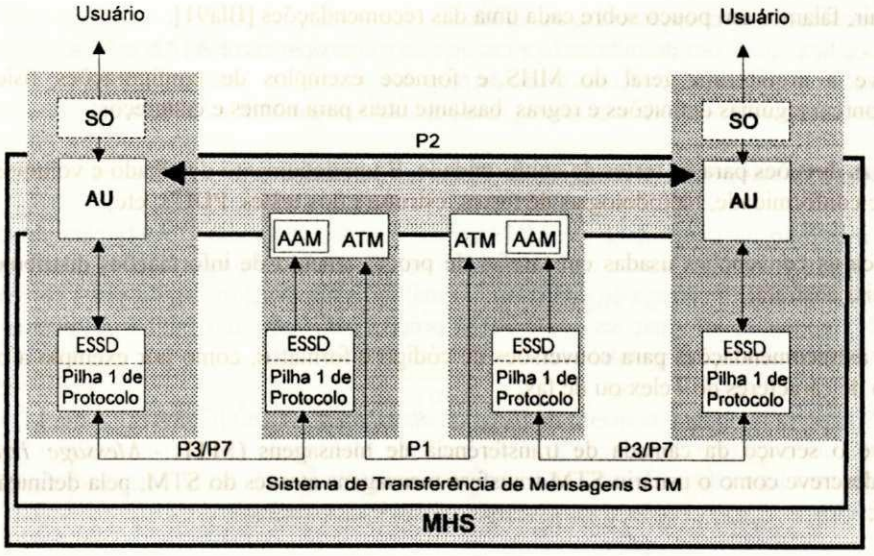


Figura 4.12. O Modelo Funcional X.400 revisitado [HM91]

Quando o ATM tem os endereços em suas mãos, ele cria o equivalente a um envelope eletrônico, e inclui no mesmo os endereços do remetente e do destinatário, juntamente com outras informações específicas do STM: estas informações irão compor o cabeçalho da mensagem recebida do AU. A seguir, o ATM local envia a mensagem por meio do protocolo *P1*.

Quando a mensagem chega ao nó destinatário, o ATM tenta despachá-la para o AU correspondente via o ESSD naquele nó, por meio do protocolo P3. O usuário remoto pode estar desconectado ("logged off") neste momento, e para estas situações o ATM irá fazer uso da AAM associada a ele. Assim, caso o AU não esteja prontamente disponível, a mensagem será depositada na AAM para entrega a posteriori. Nesta situação, utiliza-se o protocolo P7 para controlar a interação entre um usuário, via AU, e a AAM local, para que sejam recuperadas quaisquer mensagens em espera.

4.4.4. As Especificações MHS - Uma Visão Geral

X.400 descreve o modelo MHS básico de acordo com o modelo de referência OSI [VN90b]. X.400 descreve em termos muito gerais como um originador (remetente) interage com o sistema agente de usuário (AU) para preparar, editar, e receber mensagens. Descreve como a entidade AU interage com a rede, e descreve as convenções para nomes e endereços. A recomendação X.400 consiste das seguintes especificações mostradas na tabela 4.1.

ISO	ITU-T	Nome do Documento
10021-1	X.400	Message Handling Systems and Service Overview
10021-2	X.402	Message Handling Systems: Overall architecture
--------	X.403	Message Handling Systems: Conformance testing
10021-3	X.407	Message Handling Systems: Abstract service definition conventions
--------	X.408	Message Handling Systems: Encoded information type conversion rules
10021-4	X.411	Message Handling Systems: Message Transfer system: Abstract service definition and procedures
10021-5	X.413	Message Handling Systems: Message store: Abstract service definition
10021-6	X.419	Message Handling Systems: Protocol specifications
10021-7	X.420	Message Handling Systems: Interpersonal messaging system

Tabela 4.1. Especificações da Série X.400 [Bla91]

A seguir, falamos um pouco sobre cada uma das recomendações [Bla91]:

X.402: descreve a arquitetura geral do MHS e fornece exemplos de configurações físicas possíveis. Esta especificação contém algumas definições e regras bastante úteis para nomes e endereços;

X.403: fornece as direções para os testes de conformidade. É um documento detalhado e volumoso, que define todas as exigências de conformidade, metodologias de testes, estrutura dos testes, PDU's, etc;

X.407: especifica as convenções usadas em tarefas de processamento de informações distribuídas. Descreve estas tarefas de maneira abstrata;

X.408: fornece as recomendações para conversões de código e formatos, como por exemplo a conversão de ASCII para o conjunto de caracteres de Telex ou de fax;

X.411: descreve o serviço da camada de transferência de mensagens (**MTL** - *Message Transfer Layer)*. Esta recomendação descreve como o usuário STM transfere mensagens através do STM, pela definição de serviços, via a sintaxe abstrata;

X.413: contém as provisões para a área de armazenamento de mensagens (AAM). Descreve como a AAM age como intermediária entre os AU's e o STM;

X.419: define os procedimentos para acessar o STM e a AAM e também a troca de mensagens entre ATM's. Descreve os contextos de aplicação com os protocolos MHS *P1, P3 e P7*; e

X.420: descreve o MIP e o protocolo P2. Este serviço define a semântica e a sintaxe envolvidas no recebimento e na transmissão de mensagens para o tráfego interpessoal. Adicionalmente, recomenda as operações para a transferência de PDU's pelo sistema.

4.4.5. Considerações Finais

MOTIS [VN90b] compõe o sistema de correio eletrônico da ISO/ITU-T, e atua de maneira similar ao protocolo SMTP da pilha TCP/IP. Na prática, o MOTIS é um sistema de transferência de mensagens de correio eletrônico completo, e não um simples protocolo. Ele é também conhecido por MHS e é baseado no serviço público de tratamento de mensagens X.400 definido originalmente pelo então CCITT (atualmente ITU-T). A recomendação X.400 foi definida para fornecer um serviço de troca de mensagens de correio eletrônico internacional que fosse similar aos sistemas de correio eletrônico manuais. Consiste de um conjunto de protocolos que executa funções específicas em relação a todo o sistema de tratamento de mensagens.

O MHS define também como são as interações dos AUs com o *agente de transferência de mensagens (ATM)*. Os AUs são agrupados em classes pelo MHS, considerando-se os tipos de mensagens que cada grupo pode processar. Os AUs de um grupo são denominados AUs cooperantes.

Capítulo 5: Serviços de Nomes e Diretórios Distribuídos

5.1. Introdução

As novas tecnologias que compõem a computação distribuída representam mudanças fabulosas nos ambientes de computação atuais. Os novos ambientes são completamente diferentes dos ambientes de computação de alguns anos atrás. A disponibilidade de estações de trabalho pessoais, redes locais de computadores, e redes de longo alcance de alto desempenho, disponibilizam as facilidades necessárias para permitir o partilhamento de informações por todo o mundo. Existe agora um enorme potencial para permitir que qualquer computador possa comunicar-se com milhões de outros - em qualquer parte do mundo.

Com estas facilidades de comunicações avançadas surgem inúmeras complicações - para os usuários finais, para os administradores de sistemas, para o projetista de aplicações. Exige-se um ambiente de computação que permita que se explore este valioso potencial sem a necessidade de se conhecer em detalhes a complexidade da tecnologia que o suporta. É necessário um meio que permita que fiquem escondidas as esquisitices da rede, e ao mesmo tempo forneça facilidades de comunicações ótimas.

O ***Serviço de Diretórios*** (ou de ***Nomes***) ([Bil94] [Cha94] [Hal93b] [Com93a] [Tan89] [BRI94] [OSF91b], CTR92b] [Cha92] [OSF90b] [OSF92a]) é o componente da computação distribuída que permite que usuários e programas possam localizar e descrever pessoas, lugares, aplicações, e serviços em geral que fazem parte dos ambientes distribuídos. Ou seja, o Serviço de Diretórios é o componente que permite que aplicações e serviços localizem e façam referência a informações em um ambiente de computação distribuída. Os Nomes são uma parte crítica da computação distribuída, pois ajudam na administração dos recursos disponíveis na rede [Hun92a].

Preservar a consistência dos objetos em um ambiente de computação distribuída significa que cada modificação em um destes objetos deve ser propagada para cada sítio na rede. Em uma pequena rede, esta é uma tarefa simples, porém, em ambientes com centenas ou milhares de nós dispersos, pequenas tarefas administrativas podem vir a tornar-se uma grande dor de cabeça. Sem ferramentas que automatizem estas tarefas, a probabilidade de que venham a ocorrer erros aumenta com o tamanho da rede e com o número de locais onde as mudanças devem ser feitas [Ste91].

5.1.1. A Necessidade de Nomes em Ambientes Distribuídos

As pessoas dependem de nomes para identificar e descrever os objetos. Os nomes são compostos por uma série de informações que formam os seus *atributos* [Keh92]. Os nomes e as informações a respeito dos objetos que eles descrevem podem ser agrupados em listas e disponibilizados publicamente, de tal forma que se possa referenciar esta informação. Por exemplo, os nomes das pessoas e empresas em uma cidade são compiladas em listas telefônicas (que são na realidade *diretórios de telefones*) que fornecem informações como números de telefones e endereços. Em nossa sociedade moderna, as listas telefônicas fornecem um meio fácil para se localizar as pessoas. Lembre-se que todos os anos são emitidas novas listas telefônicas, de tal forma que se uma pessoa mudar de endereço, esta informação será atualizada na nova lista.

De maneira similar, as redes de computadores precisam de nomes e diretórios para descrever e guardar as características dos diversos serviços e informações que elas fornecem. Em um ambiente de computação distribuída, qualquer objeto que possa ser referenciado individualmente pode receber um nome. Exemplos de tais objetos são os serviços da rede, caixas postais de correio-eletrônico, e computadores. Cada objeto possui um registro correspondente no serviço de diretórios, chamado de *entrada*, que contém as informações, ou *atributos*, que descrevem o objeto. Nomes de entradas são agrupadas em *listas de entradas*, chamadas de *diretórios* ([FA94], [Com93a]).

Os atributos podem ser qualquer tipo de informação que descreva um objeto, como por exemplo a sua localização, cor, ou tamanho. O serviço de nomes permite que os diretórios sejam organizados em hierarquias, nas quais um diretório pode conter outros diretórios [Tan92]. Por exemplo, um diretório de telefones internacional contém os diretórios de telefones em cada país. Os diretórios de cada país contêm, por sua vez, as listas telefônicas estaduais, as quais contêm as listas telefônicas por cidades. Finalmente, as listas telefônicas de cidades contêm as entradas reais com informações de assinantes.

O serviço de nomes é de primordial importância para os ambientes da computação distribuída porque os objetos são definidos através de seus nomes. Aplicações e serviços ganham acesso a um objeto ao acessar a entrada correspondente ao nome deste objeto no diretório correspondente, e ao recuperar seus atributos. A separação de um objeto da sua localização e de suas características de acesso leva-nos ao princípio da ***independência de localização*** ([OV91] [TVR85] [LS90]). Este princípio permite que aplicações e serviços possam acessar um determinado objeto mesmo quando este objeto muda de local ou muda outras de suas características.

5.1.2. Nomes e Endereços

Endereços são usados nos ambientes da computação distribuída para identificar os processos remetentes e destinatários envolvidos em uma sessão na rede. Um endereço consiste de duas partes: a primeira parte é usada pela rede (ou inter-rede) para rotear as mensagens para os destinatários (ou sistemas finais) assinalados, e a segunda parte é usada dentro do próprio sistema final para que se possa rotear uma mensagem recebida para o devido processo de aplicação [Com91a].

Para tornar as coisas mais claras, fazemos uma analogia com os sistemas de telefonia convencionais. Imagine que uma chamada envolva um pabx de uma empresa. Neste caso, os números de ramais são relativamente pequenos pois só precisam identificar o endereço do telefone dentro de um ambiente limitado. No caso de uma rede de computadores, esta situação é similar aos endereços associados aos nós de um segmento de rede local. No entanto, se considerarmos uma ligação para fora do limite do pabx de nosso exemplo, então certamente os números de telefones serão maiores, de forma a permitir que as chamadas sejam identificadas e roteadas através das linhas nacionais e até mesmo internacionais. Lembre-se porém, que um endereço é sempre único em toda a rede (local ou de longa distância).

A segunda parte do endereço, usada para identificar um processo de aplicação específico em um computador hospedeiro, pode assumir duas formas, dependendo da pilha de protocolos que se esteja utilizando. Em TCP/IP, esta é a função dos números de portas [Mal92a]. No mundo OSI, esta é a função dos pontos de acesso de serviço (SAPs - Service Access Points) de seleção de endereços .

Os endereços usados em ambas as pilhas são compostos por muitos dígitos. Em TCP/IP, o endereço IP é um inteiro de 32 bits, o qual em uma representação mais próxima dos usuários finais poderia representar até 12 dígitos decimais. Estes 32 bits representam a primeira parte do endereço. A segunda parte é representada pelos números de portas (falamos maiores detalhes sobre portas na seção 5.1.2.1), e requer três dígitos extras (ou 8 bits). Na pilha OSI, o endereço relacionado com a rede pode ter até 40 dígitos decimais, mais os dígitos adicionais para a segunda parte do endereço. Portanto, não é interessante que sejam usados endereços (mesmo em formato decimal) pelos usuários finais.

No mundo real, os usuários - pessoas ou processos de aplicações - são referenciados através de *nomes simbólicos* ([FA94] [Tan89] [BRI94]), e não por endereços. Assim, da mesma maneira que os usuários dos sistemas de telefonia utilizam as listas telefônicas (ou diretórios de telefones) para encontrar o número de uma pessoa conhecida, os serviços de diretórios associados com as pilhas de protocolos são usados para encontrar o endereço de um destinatário nomeado (usuário ou processo). Ous seja, os serviços de tradução de nomes para endereços, ou vice-versa, são primordiais para facilitar a vida dos usuários finais. Este **serviço de resolução de endereços** (*address resolution service*) [Com91a] é um dos serviços de diretórios mais freqüentemente usados.

Podemos concluir que a grande funcionalidade no uso de nomes fica por conta do mapeamento de endereços muito longos. Adicionalmente, outros motivos tornam o uso de nomes particularmente interessante em um ambiente de computação distribuída. Por exemplo, o uso de nomes isola os usuários (ou processos de aplicações) de qualquer conhecimento aprofundado de detalhes sobre a configuração da rede, o qual evidentemente pode mudar. Por exemplo, um segmento de rede local pode ser adicionado ou removido de uma inter-rede, ou pode-se contratar outro tipo de serviço de comunicação de dados para a interconexão de segmentos isolados de redes. Tudo isto deve ser transparente para os usuários/processos de aplicações [OV91].

Devemos considerar ainda as situações onde um usuário/processo de aplicação deve estar apto a poder migrar de uma localização na rede para outra, sem que outros usuários/processos de aplicação tenham que tomar conhecimento deste fato (idealmente). Para permitir estas eventualidades, o serviço de diretórios deve fornecer além do serviço de resolução de endereços, serviços que permitam que o conteúdo do diretório, ou seja, suas entradas com a lista dos nomes simbólicos e seus endereços correspondentes, sejam modificadas e atualizadas de maneira controlada. A esta ligação de um nome simbólico com o seu endereço na rede para um usuário/processo de aplicação chamamos de "*binding*" [Hun92a].

O sistema completo de diretórios para a pilha TCP/IP é chamado de **Domain Name System - DNS** ([Com91a] [Ste91] [Hal93b] [Mal92a] [Mal92b] [OT92] [Ros91] [Dav89] [Cha92]). Na pilha OSI, ele é chamado de **Diretório X.500** ([Hal93b] [Gra91] [CTR92b] [Mal92b] [Bla91] [Cha92]). Consideramos adiante cada um destes sistemas. Examinamos também o serviço de nomes mais convencional dos ambientes Unix: o **NIS** - *Network Information Service* ([Bil94] [Ste91] [Mal92a] [Mal92b] [Blo92] [Der92b]). Antes, contudo, vejamos alguns detalhes relativos a números de portas.

5.1.2.1. Números de Portas

Como vimos no capítulo 3, na pilha TCP/IP um computador na rede pode ter muitas conexões TCP ou UDP a qualquer momento. Cada uma destas conexões é diferenciada das demais por um *número de porta* ([Hal93b] [Com91a] [Com93a]), que serve como uma espécie de número de caixa postal para os datagramas que chegam a um computador hospedeiro. Podem existir vários processos de aplicações utilizando conexões TCP ou UDP em uma única máquina, e os números de portas distinguem estes processos para os pacotes que chegam. Quando um programa de usuário abre uma conexão TCP ou UDP, ele é conectado a uma porta no computador local. A aplicação pode especificar uma porta, geralmente ao tentar alcançar algum serviço via um *número de porta bem-conhecida* ([Com91a] [Mal92a]), ou pode permitir que o próprio sistema operacional escolha o número de porta a partir do próximo número de porta disponível. Quando um pacote é recebido e passado para o módulo TCP ou UDP, ele é direcionado para o processo de aplicação pertinente, com base na definição do número de porta no pacote. A seqüência: ***(endereço-IP-de-origem, porta-de-origem, endereço-IP-de-destino, porta-de-destino)*** identifica de maneira única cada conexão entre sistemas na rede.

5.2. Network Information Service - NIS

O serviço de nomes da Sun Microsystem chama-se **Network Information Service - NIS** ([Bil94] [Ste91] [Mal92a] [Mal92b] [Blo92] [Der92b]). Nosso intuito ao falarmos do NIS é principalmente didático, pois o NIS é largamente usado em ambientes Unix em todo o mundo. O NIS é um serviço de nomes voltado para facilitar a administração de arquivos de configuração em redes do mundo Unix, principalmente em "*clusters*" da própria Sun. Atualmente, é comum que o encontremos instalado em ambientes Internet que também utilizam o DNS, e que também continuam utilizando o NIS devido ao elevado nível de funcionalidade oferecido por ele para a administração dos recursos em segmentos de redes locais. Adiante veremos como é feita a integração do NIS com o DNS nestes ambientes.

O NIS enfoca os problemas de manutenção de arquivos comuns de configuração em uma rede Unix [Ste91]. Arquivos de senhas, de grupos de usuários, e de nós na rede são mantidos e propagados para todos os nós na rede. O NIS é um sistema de banco de dados distribuído que substitui as cópias de arquivos de configuração comumente replicados por uma facilidade de administração centralizada [Mal92a]. Ao invés de se manter cada arquivo de configuração (como */etc/hosts*, */etc/passwd*, */etc/group*, */etc/ethers*, etc.) mantém-se uma base de dados para cada um destes arquivos em um servidor centralizado. As estações que estiverem usando o NIS podem recuperar informações nestas bases de dados. Se um novo segmento de rede for adicionado à rede, pode-se modificar um arquivo no servidor central e propagar-se esta modificação para outros servidores; ao invés de se sair mudando os arquivos **hosts** em cada estação de trabalho.

Assim, o NIS é voltado para fornecer a consistência necessária para a administração com sucesso de sistemas de arquivos distribuídos em ambientes Unix. É comum o seu uso em ambientes que também utilizam o **NFS** (*Network File System*) [SUN89], um dos sistemas de arquivos distribuídos mais usados em todo o mundo, como veremos no capítulo 6.

5.2.1. A Arquitetura do NIS

O NIS adequa-se ao modelo cliente/servidor. Um servidor NIS é um servidor na rede que contém arquivos de dados, chamados de *mapas* [Bil94]. As estações clientes requisitam informações destes mapas. Os servidores são divididos por sua vez em servidor mestre e servidores escravos: o servidor mestre é o verdadeiro dono dos mapas ([Bil94] [Blo92]). Os servidores NIS escravos tratam os requisitos de usuários, porém não podem modificar os mapas. O servidor mestre é responsável por toda a manutenção dos mapas, e por sua distribuição através dos servidores escravos. A figura 5.1 mostra a relação entre mestres, escravos e clientes.

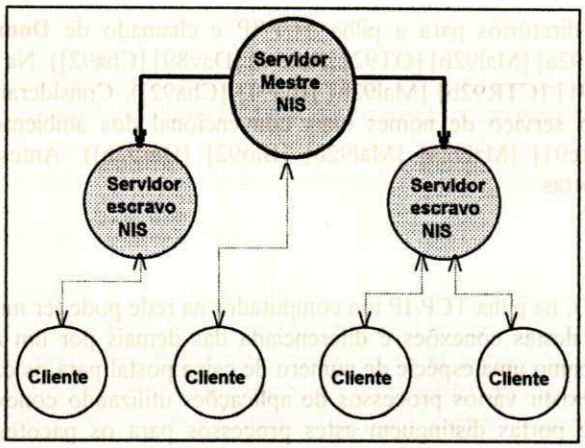


Figura 5.1. Servidores mestre, escravos e os clientes NIS [Ste91]

Atualizações são sempre efetuadas nos arquivos *hosts* do servidor mestre. O servidor mestre distribui a seguir as novas versões dos mapas NIS para os servidores escravos, que podem a partir deste momento fornecer informações atualizadas para os clientes NIS. Assim, as estações em uma rede serão enquadradas da seguinte maneira com relação ao esquema do NIS: ou somente servidores escravos; ou servidor mestre; ou clientes.

5.2.2. Domínios no NIS

Para permitir que o administrador de sistemas estabeleça diferentes políticas para diferentes ambientes, o NIS fornece o conceito de domínios. Um domínio é um conjunto de mapas NIS. Um cliente pode referir-se a um mapa (por exemplo, o mapa *hosts*) a partir de um de vários domínios diferentes. Na maioria dos casos, no entanto, uma estação só irá procurar dados em um conjunto específico de mapas NIS. Portanto, é comum (apesar de não ser correto) que se use o termo "***domínio***" [Mal92a] para significar "o grupo de estações que partilham o mesmo conjunto de mapas NIS".

Todas as estações que precisem partilhar informações de configuração comuns são colocadas em um domínio NIS específico. Apesar de cada estação poder potencialmente acessar informações em qualquer domínio NIS, cada uma delas tem o seu domínio *default* assinalado pelo administrador de sistemas. É o administrador de sistemas que decide quantos domínios diferentes são necessários [Mal92a].

Todas as estações em um domínio NIS partilham os mesmo arquivos de senhas, grupos e hosts. Tecnicamente, os mapas são eles mesmos agrupados em conjunto para formar um domínio, e as estações são ligadas a um ou mais destes domínios. Para todos os propósitos práticos, no entanto, um domínio NIS inclui tanto um conjunto de mapas quanto as máquinas que utilizam as informações nestes mapas. Podemos considerar que um domínio é um *nome* que se aplica a um conjunto de mapas NIS [Bil94].

5.2.3. O Projeto de uma Rede com os Serviços do NIS

No projeto de uma rede que deseje implementar os serviços do NIS, devem ser decididos o número de domínios, o número de servidores para cada domínio, e o nome destes domínios. Depois de pronto o modelo conceitual, a instalação e a manutenção dos servidores NIS é feita de maneira muito simples [Ste91].

5.2.3.1. A Divisão da Rede em Domínios

O número de domínios necessários em uma rede depende da segmentação e divisão dos recursos. Podemos considerar duas opções para a divisão da rede em domínios [Ste91]:

- Usar um domínio NIS separado para cada grupo de estações em seu próprio segmento de rede. Por exemplo, em uma grande empresa pode existir um segmento de rede Ethernet no departamento de planejamento, interligada por meio de uma ponte a um outro segmento Token Ring na Engenharia. Poderíamos considerar então os dois domínios: *planejamento* e *engenharia*.

- A outra opção é usar um domínio NIS separado para cada grupo de estações que tenham o seu próprio administrador de sistemas.

5.2.3.2. Os Nomes dos Domínios

No NIS qualquer convenção de nomes pode ser usada desde que os nomes de domínios sejam únicos. Pode-se escolher de nomear múltiplos domínios baseando-se em divisões hierárquicas, como por exemplo:

lab-engenharia
proj-engenharia
usuario-planejamento
prog-planejamento

Assim, em uma determinada empresa, os dois primeiros domínios representariam as estações no laboratório e as estações usadas pelos projetistas, ambas no departamento de Engenharia. Nos outros dois domínios, estão representadas as máquinas de usuários convencionais e de programadores do departamento de Planejamento. No entanto, no NIS a estrutura de domínios é do tipo "flat", isto é, não fornece hierarquias de fato.

5.2.3.3. Número de Servidores NIS por Domínio

O número de servidores em um domínio NIS é determinado pelo tamanho do domínio, pelo volume de serviços exigidos dele, pelo nível de tolerância a falhas requerido, e por quaisquer restrições físicas da rede que venham a afetar as características de ligação ("*binding*") [Hun92a] das estações clientes.

Como uma regra geral, devem existir pelo menos dois servidores por domínio: um mestre e um escravo. Este modelo de servidores dual oferece uma proteção básica no caso de queda de um dos servidores, pois os clientes em um destes servidores poderão ser religados ("*rebind*") para um segundo servidor. Com um servidor solitário, a operacionalidade de toda a rede irá depender da saúde deste servidor NIS, criando-se um ponto de gargalo e um ponto de falha único na rede. Aumentando-se o número de servidores NIS por domínio reduz-se o impacto da queda de um dos servidores [Ges93a].

5.2.4. Limitações do NIS

Podemos considerar o NIS como um serviço efetivo de manutenção de arquivos comuns a serem usados em pequenas redes, porém faltam-lhe características de serviços mais modernos. Podemos considerar três limitações principais no NIS [Mal92a]:

- Espaço de nomes "flat" (sem hierarquias): isto significa que um mapa fica disponível para todo um domínio. Também significa que os servidores não têm como obter informações de outros domínios.

- Controle de acesso e autenticação mínimos: o NIS não possui controle de acessos. O serviço básico do NIS é um acesso de somente-leitura aos dados, sem provisões de segurança: qualquer usuário pode ler tudo. Não existe o conceito de listas de controle de acesso para os mapas.

- Não permite atualizações interativas: este problema dificulta particularmente a administração dos domínios. O serviço interativo fornecido é de simples busca, e não há a possibilidade de se modificar os dados. Isto significa que as modificações devem ser feitas manualmente: conecta-se a um servidor mestre, edita-se um arquivo texto com uma determinada sintaxe e submete-se este arquivo para se efetivar a atualização.

5.2.4. O NIS+

O sucessor natural do NIS é o NIS+. O **NIS+** ([Bil94] [Mil91] [Cha92] [Cha94], é um serviço de nomes "leve". Ele é na realidade um acessório complementar para os serviços oferecidos pelo padrão de diretórios ISO/ITU-T **X.500** ([Hal93b] [Bla91]), este sim é um sistema de nomes completo, baseado em atributos.

O NIS+ fornece dois serviços básicos [Cha94]: um serviço de nomes distribuídos; um sistema de gerenciamento de banco de dados.

Como provedor de espaços de nomes, o NIS+ permite que um usuário adicione, remova, e procure por nomes. As bases de dados manipuladas pelo NIS+ seguem uma hierarquia de nomes (ao invés de uma organização "flat" como no NIS). Como provedor de banco de dados, o NIS+ fornece uma série de primitivas para a manipulação de registros em tabelas: adicionar, remover, pesquisar.

Os serviços NIS+ são fornecidos por um conjunto de servidores de nomes. Cada servidor trata uma porção do espaço de nomes. O conteúdo da base de dados em cada servidor de nomes pode ser replicado em um ou mais *pontos de serviço*: que são os pontos de contato para os *principais* que desejem alterar o espaço de nomes. *Principal* é a denominação que se dá a um cliente NIS+. Cada principal possui uma chave pública e um par de chaves privadas, usadas para autenticação. O acesso às informações sobre um principal requer o nome deste principal e sua chave pública. É neste ponto que se checam os direitos de acesso para um determinado principal.

5.2.5. Domínios Internet Versus Domínios NIS

O termo "domínio" é empregado de diferentes maneiras em diferentes ambientes. Na comunidade Internet, um domínio refere-se a um grupo de estações que são gerenciadas por um serviço de nomes de inter-redes. Estes domínios são definidos estritamente em termos de grupos de estações que são gerenciadas de maneira comum, e estão ligadas a uma empresa através de suas hierarquias. Estes domínios incluem organizações inteiras, ou divisões, e podem abranger várias redes TCP/IP. Por exemplo o domínio Internet ***rnp.br*** *e*spalha-se por todos os estados brasileiros (é o domínio da Rede Nacional de Pesquisas, o braço brasileiro da Internet).

Domínios do NIS diferem dos domínios DNS na Internet de várias maneiras [Ste91]. Os domínios NIS existem somente no esquema local de gerenciameno de redes, e são geralmente limitados por barreiras físicas, ou então por questões ligadas aos fornecedores de estações de trabalho (lembre-se que o NIS é autêntico somente nos ambientes Unix da Sun). Devido às constantes difusões de pacotes pela rede (causadas por atualizações de domínios), um domínio NIS irá freqüentemente cobrir somente um único segmento de rede. Podem existir, é claro, vários domínios em uma rede, todos administrados pelos mesmo administrador de sistemas.

É bom lembrar o que já dissemos anteriormente: no NIS é o conjunto de estações que usam estes mapas que definem os domínios, e não simplesmente os segmentos que particionam a rede. Em geral, pode-se encontrar um domínio NIS em um domínio DNS da Internet: a integração entre o NIS e o DNS será vista na seção 5.4 adiante.

5.3. Domain Name System - DNS

Antes de falarmos do DNS em si, abordamos alguns pontos freqüentemente questionados a respeito dos serviços de nomes. Estes são pontos práticos, e que devem ser apresentados ao leitor para que ele tenha um conceito factível do que lhe espera quando estiver fazendo uso dos serviços oferecidos em ambientes da computação distribuída, particularmente com relação à possibilidade de se acessar e partilhar recursos dispersos, por meio dos serviços de diretórios.

5.3.1. Como Funcionam os Serviços de Nomes

Do ponto de vista histórico, os usuários de um sistema de correio eletrónico eram endereçados simplesmente pelos seus nomes [OT92]. Um usuário na rede poderia mandar uma mensagem para outro usuário Suzana, na Universidade Estadual do Ceará, por meio do comando:

% mail suzana

e digitando a mensagem a seguir. Se Suzana é um usuário em um sistema, poder-se-ia adicionar simplesmente o nome deste sistema remoto (UECE) como um prefixo, de tal forma que o comando passaria para:

% mail uece!suzana

O ponto de exclamação é um separador usado no UUCP (Unix-to-Unix-CoPy), um conjunto de programas para permitir que sistemas Unix comuniquem-se entre si. Este tipo de nomeação é adequado para pequenos ambientes. Em redes maiores, é comum nos depararmos com comandos como:

% mail mec!cnpq!rnp!uece!suzana

que indica um caminho por onde a mensagem deve passar para ser despachada para o destinatário final Suzana, e que certamente representa uma forma muito inconveniente de se localizar recursos ou usuários na rede. Por conseqüinte, foram desenvolvidos módulos de software para registrar os sistemas de computação e os sítios em uma rede, utilizando-se o conceito de nomes de domínios (domain names). Assim, um endereço equivalente ao anterior, utilizando o sistema de domínios, e não uma caminho de roteamento explícito, poderia ser simplesmente:

% mail suzana@dc.uece.br

No sistema de domínios, todos os sistemas possíveis são agrupados em sub-grupos chamados de domínios. Cada domínio é dividido em sub-domínios. Idealmente, visto de fora, um sistema de roteamento de mensagens de correio eletrônico só precisa saber como alcançar o domínio mais alto na hierarquia, pois este domínio irá indicar a localização do usuário de interesse. A partir daí, fica por conta do domínio mais alto na hierarquia descobrir como alcançar o sub-domínio, e por conta deste sub-domínio alcançar cada um de seus próprios sub-domínios, e assim por diante.

Por exemplo, o endereço acima especifica que Suzana pode ser encontrada na máquina *dc*, no domínio *uece.br* o qual está assinalado para uma universidade, a Universidade Estadual do Ceará. Poderíamos simplesmente ter usado também a seguinte forma de endereçamento em nosso comando do exemplo:

% mail suzana@uece.br

e neste caso, estamos simplesmente considerando que o domínio *uece.br* sabe não somente como encontrar cada um de seus sub-domínios, mas também cada um de seus usuários. Esta afirmação é verdadeira somente para uma pequena Universidade como a UECE, porém não o seria certamente para uma grande universidade como a USP (Universidade de São Paulo).

Para esclarecermos o assunto, vamos mais uma vez fazer uma analogia, e desta vez com o serviço dos Correios e Telégrafos. Imagine uma pequena cidade de, digamos, 900 habitantes, chamada de Brejo da Cruz, no interior da Paraíba. Se alguem enviar uma carta para D. Laís Rodrigues, e endereçar somente "Para: D.Laís Rodrigues, Brejo da Cruz, Pb" no envelope, ainda assim haverá uma grande chance desta carta chegar ao seu destinatário. Porém algo deste tipo jamais aconteceria em cidade maiores como Recife ou Fortaleza.

De qualquer forma, para que um sistema de domínios funcione a contento, deve existir um sistema que permita o registro de domínios, e deve existir também algum tipo de banco de dados que explique como se acessar pelo menos os domínios mais altos na hierarquia. A sugestão óbvia para os sistemas de domínio, é que seja seguida a mesma referência de domínios geográficos usada pelos Correios. Por meio deste enfoque, poderíamos encontrar Suzana no seguinte endereço: ***suzana@uece.fortaleza.ce.br.*** Este tipo de endereçamento é usado na Europa, onde os padrões ISO são mais difundidos, o que realmente simplifica o enfoque. Infelizmente, o modelo de endereçamento da Internet não funciona desta maneira, devido a fatores históricos dos quais falamos a seguir.

A Internet utilizava inicialmente um serviço de nomes não hierárquico ("flat"), o qual foi rapidamente substituído pelo **DNS** (*Domain Name System*) ([Goe93a] [Goe93b] [Com91a] [Ste91] [Hal93b] [Mal92a] [Mal92b] [OT92] [Ros91] [Dav89] [Cha92]). Inicialmente foram criados nos Estados Unidos seis domínios topos na hierarquia: *.com* para sítios comerciais; *.edu* para instituições educacionais; *.gov* para instituições governamentais americanas; *.mil* para agências militares; *.net* para a administração de outras redes. Atualmente, a maioria dos domínios faz parte ou do tipo *.com* ou *.edu*. [255c] Hoje em dia existem outros domínios topos, como os domínios nacionais como **us** para Estados Unidos, **es** para Espanha, **br** para Brasil, ou **ca** para Canadá (são nomes padronizados pela norma ISO 3166) [Goe93a].

Nome dos Domínios	*Significado*
COM	Organizações Comerciais
EDU	Instituições Educacionais
GOV	Instituições Governamentais
MIL	Grupos Militares
NET	(Internet) Centros de Suporte da Rede
ORG	Outras Organizações
INT	Organizações Nacionais (internas)
USA	
BR	
|	
|	
|	Códigos de países para atribuições geográficas

Tabela 5.1. Nomes de domínios na Internet [Goe93a]

Cada um destes domínios por sua vez usa uma hierarquia de nomes apropriada. Por exemplo, no domínio *.edu* o próximo nível na hierarquia são os nomes das diferentes instituições educacionais que o compõem, enquanto no domínio *.com* estão as organizações comerciais. Assim, cada um destes domínios topo são subdivididos em sub-domínios, que por sua vez podem ter até 127 outros níveis de sub-domínios. O esquema geral e as convenções de nomes estão mostrados na figura 5.2.

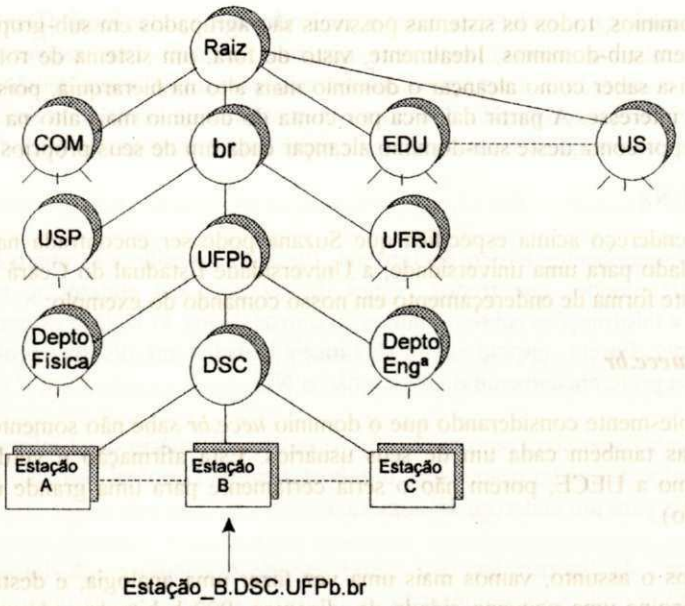


Figura 5.2. Hierarquia de nomes em domínios Internet [Com93a]

Na prática, o endereçamento de domínios da Internet é o mais largamente usado em todo o mundo (pelo menos até que o padrão internacional X.500 esteja largamente implementado e disponibilizado) [HM91].

5.3.2. Como Funciona o Endereçamento de Domínios na Internet

Cada nó na Internet recebe um endereço numérico único - o endereço ***IP***. Por exemplo, ***monica.dsc.ufpb.br*** corresponde ao endereço numérico 150.165.1.1. Este endereço numérico é usado tanto para redes locais que usam o TCP/IP quanto para a própria Internet. Os números da Internet, assim como os nomes de domínios, são assinalados por uma organização conhecida como Network Information Center (NIC) [Qua90], a qual é administrada pela empresa Government Systems, em Virgínia, nos Estados Unidos (anteriormente o NIC era administrado pela empresa SRI International, da California).

Quando um sítio Internet deseja conectar-se a outro, o sítio remetente tem que fazer a tradução do endereço simbólico (por exemplo, *monica.dsc.ufpb.br*) para o endereço IP correspondente (150.165.1.1). Esta tradução é feita por um programa chamado **named** (*"name daemon"*) [Hun92a]. O programa *named* no sítio de origem requisita dos *servidores de nomes raiz* (**"root name servers"**) da Internet o nome do servidor que mantém o domínio *ufpb.br*. O sítio de origem requisita a seguir a este servidor a tradução do nome *monica.dsc.ufpb.br* e recebe como resposta o endereço *IP* correspondente, o qual será usado para estabelecer a conexão. Na prática, nomes de estações e endereços numéricos podem ser usados em comandos de utilitários da pilha TCP/IP [Hun92a]. Por exemplo, um usuário que deseja emular uma estação no endereço IP 150.165.1.1 pode digitar:

% telnet 150.165.1.1

ou então usar o nome da estação associado a este endereço e usar o comando equivalente:

% telnet monica.dsc.ufpb.br

Seja o comando fornecido com um nome de estação ou com um endereço IP, a conexão da rede será sempre estabelecida levando-se em consideração o endereço IP. O DNS converte o nome da estação para um endereço antes que a conexão da rede seja estabelecida. O administrador da rede é o responsável pelo estabelecimento de nomes e endereços e por armazená-los no banco de dados [Ste91].

Em uma rede tão abrangente quanto a Internet, onde existem muito poucas pessoas para administrarem tantos detalhes que envolvem toda a rede (de alguns milhões de usuários), faz sentido que se utilize o enfoque hierárquico para nomes no DNS ([Goe93a] [Goe93b]). Cada organização torna-se responsável pela manutenção de seus próprios servidores de nomes com seus próprios domínios, através do administrador local da rede. Quando surge a necessidade de se usar um endereço de estação em outros domínios, os *servidores raiz* passam adiante os requisitos para os domínios corretos.

A tradução de nomes para endereços não é uma simples tarefa local [Dav89]. O comando ***telnet monica.dsc.ufpb.br*** deve trabalhar corretamente em qualquer estação que estiver conectada à rede. Se ***monica.dsc.ufpb.br*** estiver conectado à Internet, todos os nós no mundo estão habilitados a traduzir o nome *monica.dsc.ufpb.br* para o endereço apropriado. Portanto é necessário que exista uma facilidade para disseminar as informações sobre nomes de estações para todos os sítios da rede. Esta é uma das funções do DNS.

5.3.3. A Configuração do DNS

O DNS é um dos componentes básico da Internet. Podemos considerá-lo como um sistema de banco de dados distribuído a nível de mundo, que traduz nomes de estações em endereços, e vice-versa, fornecendo informações de roteamento, e informações sobre domínios ou outros dados. O DNS é essencial para cada sítio que se conecta diretamente à Internet. Porém, mesmo para redes locais isoladas que utilizem protocolos da pilha TCP/IP, sempre faz sentido que se use preferencialmente o DNS, e não o NIS.

Sempre que são usados serviços como telnet, ftp, e outros, o usuário deve informar um nome de estação (um nome de um computador hospedeiro, ou seja um *host name*). Funções resolvedoras (*resolver functions*) [Hun92a] traduzem este nome para um endereço ***IP*** numérico.

O rápido crescimento da Internet levou à introdução de domínios e sub-domínios DNS. Os domínios compõem a estrutura do DNS e podem ser comparados a diretórios em um sistema de arquivos, com cada domínio aparecendo como sub-árvores de informações no espaço de nomes de domínios. Um nome de estação (*host*) em um domínio deve ser único, porém, este mesmo nome pode existir em outros domínios. Os domínios são implementados como hierarquias, e podem ser vistos como uma árvore invertida começando no domínio *raiz* (cujo nome real é " ", ou seja, um *label* nulo) [Goe93a].

O domínio *raiz* é servido por um grupo de servidores de nomes chamados de ***servidores raiz*** [Blo92]. Os servidores raiz só possuem informações completas a respeito dos domínios de topo, que são aqueles domínios que vêem imediatamente abaixo do domínio raiz na árvore de hierarquia de domínios. Estes domínios de topo por sua vez têm apontadores para os servidores em domínios de níveis inferiores. Nenhum servidor, nem mesmo os servidores raizes, possuem informações completas sobre todos os domínios, mas os apontadores podem indicar-lhes que servidores fornecem a informação desejada. Assim, os servidores raiz podem até não saber a resposta para uma pergunta, mas certamente saberão a quem perguntá-la. A figura 5.3 mostra uma parte da árvore de domínios da Internet.

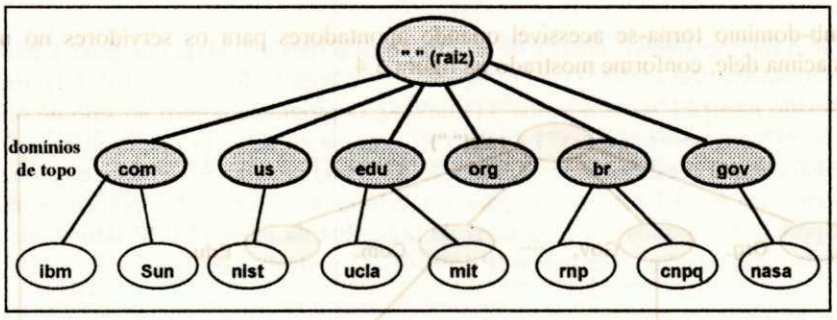


Figura 5.3. Uma parte da árvore de domínios da Internet [Ros91]

5.3.3.1. Características do DNS

O DNS é um sistema perfeitamente escalável. Ele não se baseia em uma única tabela; é na realidade um sistema de banco de dados distribuído que não se degrada à medida que as bases de dados crescem. Em 1993, o DNS fornecia informações para aproximadamente 700.000 sítios em todo o mundo [Goe93a]. O DNS garante que, a disseminação das novas informações a respeito de novos sítios seja efetuada para o restante da rede à proporção que estas forem sendo requisitadas.

Além de propagar as informações automaticamente, estas somente são disseminadas para quem estiver interessado nelas. Vejamos como funciona o DNS: se um servidor DNS recebe um pedido sobre informações a respeito de uma estação, este servidor passa adiante o pedido para um ***servidor autorizado*** ([Ros91] [OT92]). Um servidor autorizado é qualquer servidor que seja responsável pela manutenção de informações precisas e atuais sobre o domínio que estiver sendo pesquisado. Quando um servidor autorizado emite uma resposta, o servidor local salva (em *cache*) esta resposta para uso futuro. A próxima vez que o servidor local receber um pedido por esta informação, ele mesmo fornecerá a resposta.

Esta habilidade de controlar as informações a respeito de estações a partir de uma fonte autorizada, e também a possibilidade de disseminar informações precisas de maneira automática, tornam o DNS um serviço de alto nível, mesmo para pequenas redes que não estejam conectadas à Internet.

5.3.3.2. A Criação de Domínios e Sub-domínios DNS

A autoridade para alocar domínios é de responsabilidade do **NIC** - *Network Information Center* ([Goe93a] [Goe93b] [Qua90]). Para conseguir-se um domínio, deve-se candidatar ao NIC para que ele possa designar um domínio abaixo de um dos domínios de topo. Quando se consegue a autorização, uma organização pode então criar domínios adicionais, chamados de sub-domínios, abaixo daquele domínio fornecido pelo NIC.

Vamos supor que a empresa Connect, em Campina Grande, na Paraíba, tenha se candidatado a um domínio junto ao NIC. A Connect é um instituição comercial com fins lucrativos. Ela se encaixa claramente no domínio *com*. Foram encaminhados os documentos para o NIC na tentativa de se conseguir criar um domínio chamado *connect* abaixo do domínio *com*. Os documentos para este novo domínio contêm os nomes de pelo menos duas estações (seus endereços serão fornecidos pelo NIC) que irão fornecer serviços de nomes para o novo domínio. Quando o NIC aprova o requisito por este novo domínio, ele adiciona apontadores no domínio *com* apontando para os novos servidores de nomes no novo domínio. A partir deste momento, quando consultas são recebidas pelos servidores raiz para o domínio *connect.com*, estas consultas são direcionadas para os novos servidores de nomes.

A aprovação por parte do NIC garante completa autoridade à Connect sobre os novos domínios que forem criados abaixo de *connect.com*. Qualquer domínio registrado tem autoridade para dividir seu domínio em sub-domínios. No caso da Connect, podemos criar domínios separados para o departamento de planejamento (*pnjto.connect.com*), e para o departamento de engenharia (*enga.connect.com*). Isto pode ser feito sem se consultar o NIC. A decisão de se adicionar sub-domínios fica completamente a cargo do administrador do domínio local.

O assinalamento de nomes é similar ao assinalamento de endereços. O NIC assinala um endereço IP para uma organização, e a organização cuida de assinalar endereços de sub-redes e de estações dentro da abrangência do endereço original fornecido pelo NIC. Similarmente, o NIC assinala um domínio para uma organização, e a própria organização cuida de assinalar sub-domínios e nomes de estações naquele domínio. O NIC é a autoridade central, e é quem delega autoridade e distribui o controle sobre nomes e endereços para as organizações individualmente. Desde que esta autorização tenha sido delegada, as organizações por si são responsáveis pelo gerenciamento de nomes e endereços que lhes sejam assinalados.

Um novo sub-domínio torna-se acessível quando apontadores para os servidores no novo domínio são criados nos domínios acima dele, conforme mostrado na figura 5.4. .

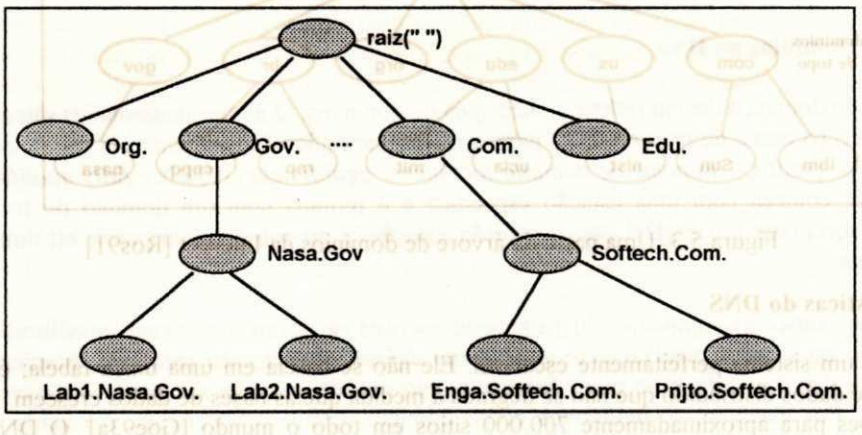


Figura 5.4. A hierarquia de domínios [Goe93a]

Servidores remotos não estão habilitados para acessar o domínio *connect.com* de nosso exemplo, até que um apontador para seus servidores seja criado no domínio *com*. Da mesma forma, os sub-domínios *enga* e *pnjto* não podem ser acessados até que apontadores para eles sejam criados em *connect.com*. Os registros de banco de dados do DNS que apontam para os servidores de nomes de um domínio são chamados de registros **NS** (*name server*) [Hun92a]. Este registro contém o nome do domínio e o nome da estação que é servidora para este domínio. A figura 5.5. ilustra como os registros NS são usados como apontadores.

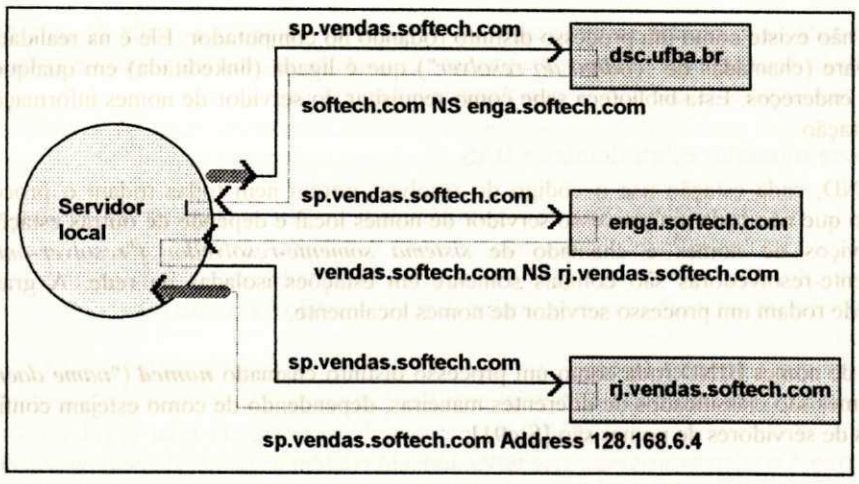


Figura 5.5. Um busca não-recursiva (adaptado de [Hun92a])

Um servidor local faz um pedido para traduzir (resolver) o nome *sp.vendas.connect.com* em um endereço IP. O servidor local (de um domínio, digamos, em uma empresa de Salvador) não possui informações sobre *connect.com* em sua cache, e portanto, ele consulta um servidor raiz *dsc.ufba.br* (o servidor de nomes na Universidade Federal da Bahia) para descobrir o endereço procurado. Este servidor raiz responde com um registro NS (um apontador) que aponta para *enga.connect.com* como a fonte de informações sobre *connect.com*. O servidor local então consulta o servidor *enga.connect.com*, que por sua vez aponta para rj.*vendas.connect.com* como o servidor para *sp.vendas.connect.com*. O servidor local consulta agora rj.*vendas.connect.com* e finalmente recebe o endereço IP desejado.

O servidor local guarda em cache o registro de endereço e cada um dos registros NS. A próxima vez que ele for consultado a respeito de *sp.vendas.connect.com*, ele mesmo cuidará de responder à consulta. E todas as vezes que o servidor venha a receber consultas sobre outras informações no domínio *connect.com*, ele irá diretamente para enga.connect.com, sem ter que passar por outros servidores raiz.

A figura 5.5. anterior ilustra o exemplo de uma busca não-recursiva. Em uma consulta não-recursiva, o servidor remoto indica ao servidor local quem deve ser o próximo servidor a ser consultado. O servidor local deve então seguir os apontadores por sua própria conta. Em uma busca recursiva, o servidor remoto segue ele mesmo os ponteiros, e retorna o resultado final para o servidor local. Os servidores raiz geralmente só executam buscas não-recursivas ([Com91a] [Com93a]).

5.3.3.3. Nomes de Domínios no DNS

Os nomes de domínios devem refletir a hierarquia de domínios. O nomes de domínios são escritos a partir do nome mais específico (o nome de uma estação) para o menos específico (um domínio de topo), com cada parte do domínio separada por pontos. Um *nome de domínio totalmente qualificado* (**FQDN** - *fully qualified domain name*) ([Dav89] [Hal93b]), começa com uma estação específica e e termina com um domínio de topo. Por exemplo, *estação22.pnjto.connect.com* é o FQDN para a estação *estação22* no sub-domínio *pnjto*, no domínio *connect*, do domínio de topo *com*.

Na prática, nomes de domínios são raramente escritos na forma totalmente qualificada. Geralmente os nomes de domínios são escritos relativamente ao *domínio default* estabelecido para uma estação. O próprio DNS cuida de adicionar o domínio default ao nome fornecido pelo usuário, quando são construídos os requisitos para o servidor de nomes. Por exemplo, se definirmos como domínio default *connect.com*, o usuário pode omitir esta extensão para quaisquer referências a nomes neste domínio. Assim, *pnjto.connect.com* poderia ser referenciado simplesmente como *pnjto*, o DNS cuidaria de adicionar o domínio *connect.com* como sufixo.

5.3.3.4. BIND, Resolver e Named

A implementação do DNS na maioria dos sistemas Unix é baseada no software **BIND** (*Berkeley Internet Name Domain*). O software de serviços de nomes do DNS é dividido conceitualmente em dois componentes: o ***resolver***, que é o software responsável por montar as consultas; é ele que faz as perguntas. O ***servidor de nomes*** é o processo que responde às perguntas; é ele que fornece as respostas (Hal93b] [Hun92a]).

O resolver não existe como um processo distinto rodando no computador. Ele é na realidade uma biblioteca de rotinas de software (chamadas de "*código do resolver*") que é ligada (linkeditada) em qualquer programa que deseje procurar por endereços. Esta biblioteca sabe como requisitar do servidor de nomes informações a respeito de uma determinada estação.

Com o BIND, cada estação usa o código do resolver, porém nem todas rodam o processo servidor de nomes. Uma estação que não roda um processo servidor de nomes local e depende de outras estações para todas as respostas para serviços de nomes é chamado de *sistema somente-resolvedor ("resolver-only system")*. As configurações somente-resolvedoras são comuns somente em estações isoladas da rede. A grande maioria das estações Unix em rede rodam um processo servidor de nomes localmente.

O servidor de nomes BIND roda como um processo distinto chamado ***named*** ("*name daemon*") [Hun92a]. Os servidores de nomes são classificados de diferentes maneiras, dependendo de como estejam configurados. As três principais categorias de servidores de nomes são [Ste91]:

Servidor primário: é o servidor a partir do qual todos os dados sobre um domínio são derivados. O servidor primário carrega as informações sobre o domínio diretamente a partir de um arquivo texto criado pelo administrador de sistemas. Os servidores primários são *autorizados*, isto é, eles possuem informações completas a atualizadas a respeito de seus domínios. Só pode existir um único servidor primário para um domínio.

Servidor secundário: são servidores para os quais a base de dados completa de um domínio é transferida a partir do servidor primário. Uma base de dados particular para um determinado domínio é chamada de *arquivo de zona*; copiar este arquivo para um servidor secundário chama-se de *transferência de arquivo de zona*. Um servidor secundário garante que ele sempre terá informações atuais sobre um domínio por meio de transferências periódicas de arquivos de zona para este domínio. Os servidores secundários são autorizados para seus domínios.

Servidores de caching: estes servidores recebem as respostas para todas as consultas de serviços de nomes que venham de outros servidores de nomes. Quando um servidor de *caching* recebe uma resposta para uma consulta, ele guarda esta informação e a utiliza no futuro para responder ele mesmo a outras consultas que venham a ocorrer. A maioria dos servidores de nomes guardam respostas e as utilizam desta maneira. O que torna os servidores de *caching* únicos é que esta é a única técnica usada na construção de sua base de dados para o domínio correspondente. Servidores de *caching* são não autorizados, isto é, suas informações podem ser incompletas ou desatualizadas, apesar de geralmente estarem coerentes.

No DNS existe somente um servidor primário para cada domínio. Os dados do DNS são fornecidos à base de dados do servidor primário pelo administrador do domínio. Portanto, os administradores têm um controle centralizado sobre as informações de nomes de estações. Este banco de dados controlado de maneira centralizada e distribuído de maneira automática é uma vantagem para redes de qualquer tamanho. Quando um novo segmento de rede é adicionado à inter-rede, só é necessário que se modifique a base de dados do DNS no servidor primário. As informações serão disseminadas automaticamente para os outros servidores, seja por meio de transferências de zona totais, seja pelo *caching* de respostas individuais.

5.4. Integrando Domínios DNS com Domínios NIS

O DNS e o NIS possuem similaridades e diferenças. Assim como o DNS, o NIS resolve o problema de distribuição confiável de nomes, mas diferentemente do DNS, ele só fornece serviços para redes locais. O NIS não é um serviço voltado para a Internet como um todo. O NIS fornece muito mais do que conversões de nomes para endereços. Ele converte vários arquivos padrões do Unix em bases de dados que podem ser acessadas na rede (são os mapas NIS). Os mapas NIS podem ser armazenados em um servidor central onde podem ser mantidos de forma centralizada, e disseminados automaticamente para os usuários ([Ste91] [Mal92a]).

O NIS não pode ser considerado uma alternativa para o DNS, pois ele cuida de somente uma parte das informações disponíveis no DNS [Mal92a]. Por este motivo, é mais comum encontrarmos ambientes onde o NIS e o DNS são usados de maneira integrada.

Para possibilitar a integração do NIS com o DNS, em primeiro lugar deve-se usar uma hierarquia de nomes, mesmo para nomes de domínios NIS. Cada estação cuida de seu próprio nome, de tal forma que elas são os nós folha da árvore que representa a hierarquia. As estações são agrupadas em domínios NIS e DNS, criando-se uma árvore de dois níveis. Os domínios DNS podem ser agrupados em níveis mais altos na hierarquia, que podem representar departamentos, empresas, ou mesmo localizações físicas (***eu, br, jp***,...). O NIS se encaixa no esquema de gerenciamento do DNS no nível mais baixo da hierarquia, conforme mostrado na figura 5.6 [Mal92a].

Em cada domínio DNS podem existir vários segmentos de redes com vários administradores. O NIS fornece um sistema que permite o gerenciamento independente destes segmentos; o mapa *hosts* do NIS pode ser combinado para formar o arquivo *host* do DNS. Podemos considerar três formas de se integrar o NIS com o DNS [Ste91]:

- Rodar o NIS sem o DNS, o que é o default.
- Usar os mapas NIS em primeiro lugar, depois procurar no DNS nomes que não sejam gerenciados pelo NIS. Isto é possível pelo uso de um marcador ("flag") especial no mapa *hosts* do NIS.
- Ignorar o NIS, e usar somente o DNS. O maior argumento a favor do uso do DNS sozinho é que fica consolidado o gerenciamento de nomes de estações em um único serviço distribuído, ao invés de se dividir este serviço por dois sistemas.

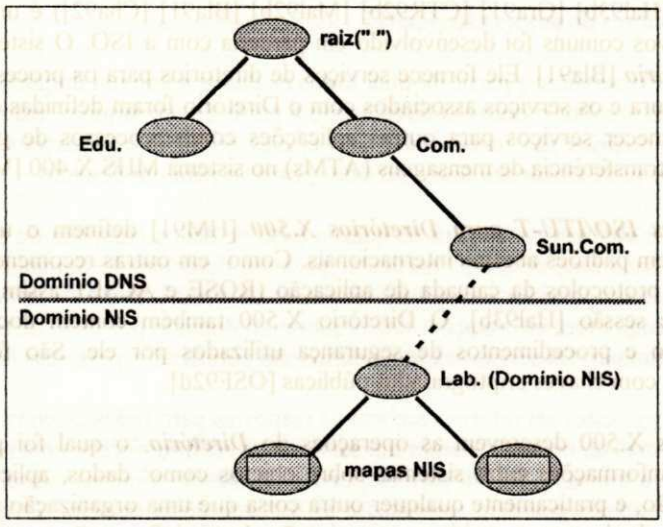


Figura 5.6. A hierarquia de nomes NIS/DNS [Mal92a]

Em algumas versões do Unix, como no sistema operacional Ultrix da DEC, pode-se especificar a ordem de busca no arquivo /etc/hosts, no mapa NIS, ou nos servidores DNS [Gli92].

Quando se opta pela segunda alternativa, os servidores NIS localizam os servidores DNS pela interface ***resolver***, que se baseia em informações no arquivo de configuração */etc/resolv.conf*. O arquivo de configuração do *resolver* deve apontar para o servidor mestre do NIS. Com esta alternativa é interessante que no projeto de nomes, os nome em domínios NIS estejam ligados aos nomes em domínios DNS. A opção é derivar os nomes em domínios NIS a partir de nomes em domínios DNS, de tal forma que a parte do nome em um domínio NIS que foi derivada de um domínio DNS indique aonde o domínio NIS deve ir buscar suas informações sobre nomes de estações.

Por exemplo, digamos que a empresa Connect Tecnologia Ltda, em Campina Grande, na Paraíba, utiliza o nome de domínio DNS ***connect.com***. Suponhamos que existam três departamentos na empresa, representados por domínios NIS e subordinados ao domínio DNS *connect.com*. Os nomes de domínios NIS utilizam o nome do domínio DNS como seu sufixo:

enga.connect.com
pnjto.connect.com
adm.connect.com

Em alguns sistemas operacionais, como no SunOs da Sun Microsystems [Mal92a], alguns comandos assumem que nomes de domínios NIS com três componentes ou mais são sempre derivados de um nome de domínio DNS, e de tal forma que ao se extrair do nome no domínio NIS seu primeiro componente, se possa formar o nome do domínio DNS. A tabela 5.2 mostra exemplos de nomes de domínios NIS e DNS.

Domínio NIS	Domíno DNS
enga.connect.com	.connect.com
pnjto.connect.com	.connect.com
adm.connect.com	.connect.com
cg.vendas.connect.com	.vendas.connect.com
sp.vendas.connect.com	.vendas.connect.com

Tabela 5.2. A subdivisão de domínios DNS em domínios NIS para o exemplo dado

5.5. Serviços de Diretório X.500 ISO/ITU-T

O padrão **X.500** ([Hal93b] [Gra91] [CTR92b] [Mal92b] [Bla91] [Cha92]) é uma das recomendações do ITU-T, e devido aos objetivos comuns foi desenvolvido em parceria com a ISO. O sistema completo é conhecido simplesmente como o ***Diretório*** [Bla91]. Ele fornece serviços de diretórios para os processos de aplicações na pilha de protocolos OSI. A estrutura e os serviços associados com o Diretório foram definidas de tal forma que ele possa ser usado também para fornecer serviços para outras aplicações como processos de gerência em aplicações de gerenciamento e agentes de transferência de mensagens (ATMs) no sistema MHS X.400 [VN90b].

As ***Recomendações ISO/ITU-T para Diretórios X.500*** [HM91] definem o uso de um Diretório para suportar sistemas baseados em padrões abertos internacionais. Como em outras recomendações ITU-T, o Diretório X.500 baseia-se no uso de protocolos da camada de aplicação (ROSE e ACSE), assim como em protocolos das camadas de apresentação e sessão [Hal93b]. O Diretório X.500 também contém documentação extensa sobre mecanismos de autenticação e procedimentos de segurança utilizados por ele. São fornecidos mecanismos de autenticação simples e forte, com chaves criptografadas públicas [OSF92d].

As Recomendações X.500 descrevem as operações do ***Diretório***, o qual foi projetado para suportar e facilitar a comunicação de informações entre sistemas sobre objetos como: dados, aplicações, hardware, pessoas, arquivos, listas de distribuição, e praticamente qualquer outra coisa que uma organização precise para propósitos de administração. O X.500 é voltado para permitir a comunicação destas informações entre diferentes sistemas, que podem incluir aplicações OSI, entidades de camadas OSI, entidades de gerenciamento OSI e redes de comunicações.

O X.500 abrange oito recomendações, chamadas em conjunto de Recomendações X.500. A tabela 5.3 abaixo lista as especificações ITU-T e suas correspondentes na ISO [Bla91].

ITU-T	ISO
X.500	9594-1
X.501	9594-2
X.509	9594-8
X.511	9594-3
X.518	9594-4
X.519	9594-5
X.520	9594-6
X.521	9594-7

Tabela 5.3. Especificações ITU-T/ISO para Diretórios [Bla91]

A seguir listamos o nome dado a cada uma das especificações:

X.500: O Diretório - Visão geral, conceitos, modelos e serviços
X.501: Modelos do Diretório
X.509: O Diretório - infra-estrutura de autenticação
X.511: O Diretório - definição de serviços abstratos
X.518: O Diretório - procedimentos para operações distribuídas
X.519: O Diretório - especificações de protocolos
X.521: O Diretório - tipos selecionados de atributos
X.521: O Diretório - tipos selecionados de classes

5.5.1 O Modelo Funcional do Diretório X.500

A informação armazenada no Diretório é conhecida por *Base de Informações do Diretório* (**DIB** - *Directory Information Base*) ([Cha92] [CTR92b] [Bla91]). A DIB contém informações sobre objetos, e é composta de *entradas*. Cada entrada consiste de um conjunto de informações sobre um determinado objeto. Cada entrada é formada por *atributos*, e cada atributo possui um *tipo* definido e um ou mais *valores*. A DIB é acessada pelo usuário do Diretório por meio de um *agente de usuário para diretório* (**DUA** - *Directory User Agent*), que é um processo de aplicação. A figura 5.7 ilustra de maneira simples os conceitos apresentados.

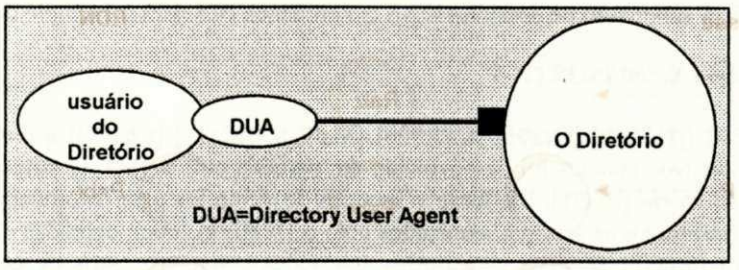


Figura 5.7. O modelo funcional do Diretório X.500 [SOIEC91b]

As entradas na DIB são arranjadas em uma estrutura de árvore chamada de Árvore de Informações do Diretório (DIT - Directory Information Tree) [Hal93b]. Os vértices na árvore representam as entradas na DIB. Estas entradas formam uma coleção de informações sobre um objeto (como por exemplo, uma pessoa, um elemento de dados, um periférico, um programa, e outros). A figura 5.8 ilustra os conceitos associados com a DIT.

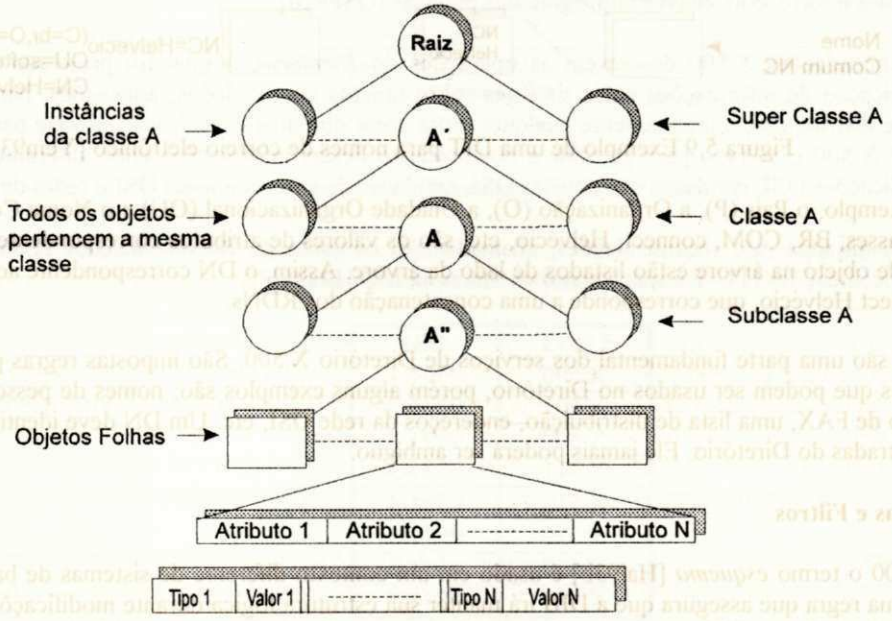


Figura 5.8. A estrutura da DIT [Sta93b]

Todas as entradas na DIT são *objetos* que representam *instâncias* de uma *classe de objetos* em particular (se o leitor desejar, pode dirigir-se ao capítulo 12 adiante, onde apresentamos uma rápida introdução ao paradigma de orientação a objetos). Todos os objetos em um determinado nível na hierarquia da DIT pertencem à mesma classe de objetos. Objetos na classe acima na hierarquia são membros da *superclasse* destes objetos, enquanto os objetos na classe de baixo são membros de sua *subclasse* [KA90]. Associado com cada classe de objeto existe um conjunto de *atributos*, e cada um destes atributos possui um *tipo* e um ou mais *valores*.

5.5.2. A Convenção Para Nomes X.500

A convenção de nomes para o Diretório X.500 é similar em princípio àquela usada no DNS. No entanto, em X.500, cada rótulo (*"label"*) é referenciado como um *nome relativo distinto* (**RDN** - *relative distinguished name*), e a lista completa de rótulos associados com um objeto na árvore é chamado de *nome distinto* (**DN** - *distinguished name*) [Yem93]. Os RDNs devem ser únicos em qualquer nível da hierarquia. Uma parte simplificada de uma DIT está mostrada na figura 5.9. Os nomes do exemplo podem referir-se, por exemplo, a nomes no sistema de correio eletrônico X.400.

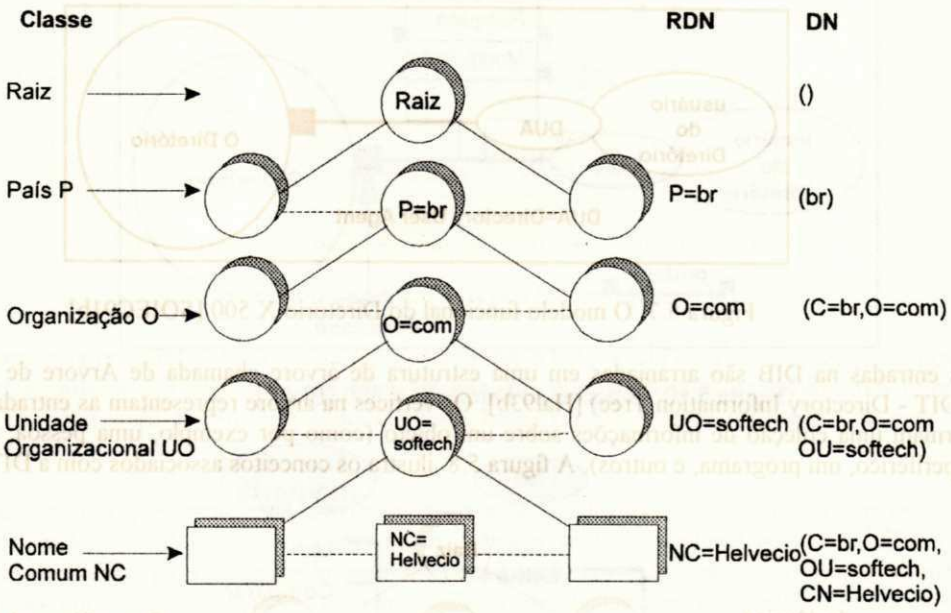


Figura 5.9 Exemplo de uma DIT para nomes de correio eletrônico [Yem93]

Neste exemplo, o País (P), a Organização (O), a Unidade Organizacional (OU) e o Nome Comum (NC) são todos tipos de classes; BR, COM, connect, Helvécio, etc, são os valores de atributos correspondentes. RDNs e DNs para cada nome de objeto na árvore estão listados de lado da árvore. Assim, o DN correspondente ao RDN Helvécio é BR.COM.Connect.Helvécio, que corresponde a uma concatenação dos RDNs.

Os DNs são uma parte fundamental dos serviços de Diretório X.500. São impostas regras para a formação de tipos de nomes que podem ser usados no Diretório, porém alguns exemplos são: nomes de pessoas, números de telefones, número de FAX, uma lista de distribuição, endereços da rede OSI, etc. Um DN deve identificar de maneira única uma das entradas do Diretório. Ele jamais poderá ser ambíguo.

5.5.2.1. Esquemas e Filtros

Em X.500 o termo *esquema* [Hal93b] é usado em um contexto diferente de sistemas de bancos de dados. Um esquema é uma regra que assegura que a DIB irá manter sua estrutura lógica durante modificações. Desta forma previnem-se inconsistências na DIB, como entradas em classes subordinadas incorretas, atributos inválidos, etc. É de responsabilidade do Diretório cuidar para que todas as modificações na DIB estejam de acordo com o esquema do Diretório. Adicionalmente, controles são fornecidos para evitar que um usuário exceda os limites (*"thresholds"*) como por exemplo o escopo de uma consulta, o tempo gasto por uma transação, o tamanho do conjunto de resultados de uma pesquisa, etc.

A DIB pode retornar uma resposta de um pedido utilizando mais de uma entrada no Diretório. Para fazer isto, emprega-se o conceito de *filtro*. Um filtro é uma regra ou um conjunto de regras sobre os atributos de um objeto. É comum o uso de filtros em sistemas de gerenciamento de redes. Em sua forma mais simples, um filtro é uma forma de selecionar testes para a escolha de operações. As operações de filtragem podem ser codificadas em C, ASN.1, etc., para permitir que sejam selecionados parâmetros específicos do Diretório. São usadas operações booleanas simples (OU, E, NÃO) para a elaboração de filtros.

5.5.3. Serviços e Portas no Diretório X.500

O Diretório é organizado utilizando-se o conceito de portas ([Bla91] [HM91]). Uma porta representa os serviços enxergados por usuários de um protocolo ISO/ITU-T. As portas X.500 são organizadas como mostrado na figura 5.10. Três portas são definidas: leitura (*read*), pesquisa (*search*) e modificação (*modify*). As operações para cada porta estão mostradas na figura 5.11.

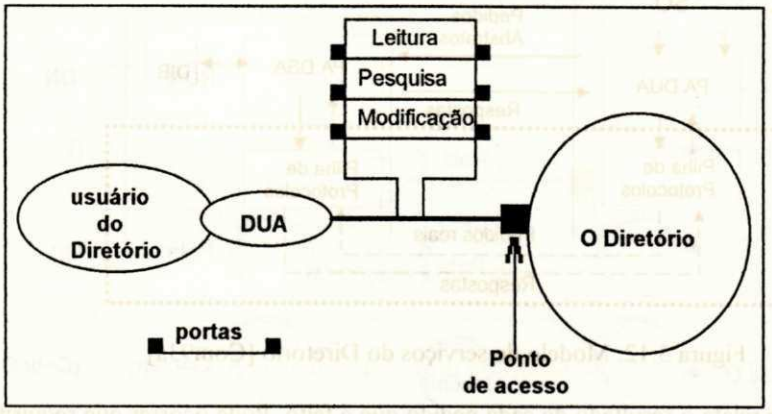


Figura 5.10 As portas no Diretório X.500 [Bla91]

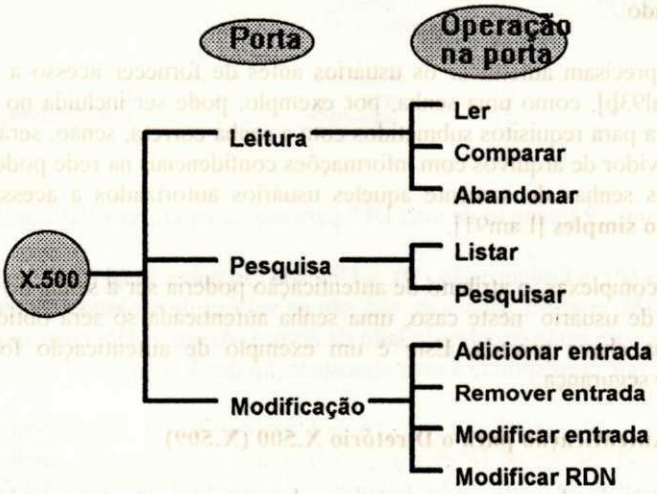


Figura 5.11 As portas X.500 e suas operações [Bla91]

A porta de **leitura** (*read*) consiste de três operações: (1) *ler*: que é usada para extrair informações sobre um objeto no Diretório; (2) *comparar*: usada para comparar um valor de argumento com um valor de atributo no Diretório; (3) *abandonar*: usada para abandonar uma operação anterior.

A porta de **pesquisa** (*search*) consiste de duas operações: (1) *listar*: para obter listas do Diretório; (2) *pesquisar*: é usada para pesquisas detalhadas ("browse") nas informações da DIT, retornando informações sobre suas entradas.

A porta de **modificação** (*modify*) consiste de quatro operações: (1) *adicionar entrada*: permite a adição de novas entradas *folhas* na DIT; (2) *remover entrada*: remove entradas folha na DIT; (3) *modificar entrada*: opera em entradas já existentes na DIT, executando operações como: remoção de atributos, adicionar valor de atributo, substituir valores de atributos, etc.; (4) *modificar RDN*: é usada para alterar o RDN de uma entrada folha na DIT.

O modelo para os serviços de diretório [Cha92] está mostrado na figura 5.12. No início desta seção citamos que todos os usuários acessam o diretório por meio de um processo de aplicação chamado *agente de usuário de diretório* (*DUA - Directory user agent*). Para resolver cada pedido de usuários, um DUA interage com o Diretório por meio um processo de aplicação chamado de *agente de serviços de diretório* (**DSA** - *directory service agent*). O DUA desempenha um papel similar ao *resolver* em serviços DNS.

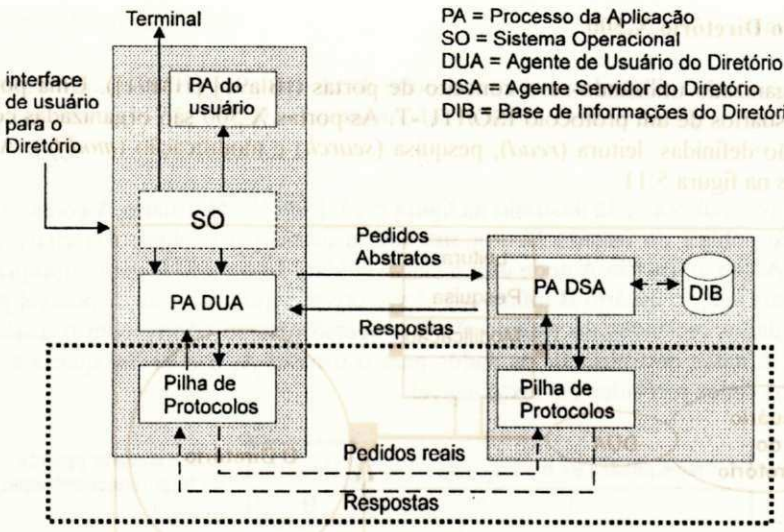


Figura 5.12. Modelo de serviços do Diretório [Com93a]

O diretório sempre relata o resultado de cada pedido que é feito. Pode ocorrer que requisitos possam falhar, por razões como a violação de segurança em uma entrada, ou problemas com os parâmetros fornecidos (por exemplo, nomes ou valores de atributos inválidos). Nestes casos retorna-se uma mensagem de erro com o tipo apropriado de erro discriminado.

Em aplicações que precisam autenticar os usuários antes de fornecer acesso a entradas no diretório, um *atributo de autenticação* [Hal93b], como uma senha, por exemplo, pode ser incluída no pedido. O diretório só irá retornar uma resposta positiva para requisitos submetidos com a senha correta, senão, será retornada uma mensagem de erro. Por exemplo, um servidor de arquivos com informações confidenciais na rede poderia checar um conjunto de atributos que contivessem as senhas de somente aqueles usuários autorizados a acessar o servidor. Este é um exemplo de uma **autenticação simples** [Lam91].

Em aplicações mais complexas, o atributo de autenticação poderia ser a senha do usuário, criptografada por meio de uma *chave pública* de usuário. neste caso, uma senha autenticada só será obtida se o usuário fornecer a senha criptografada por uma chave privada. Este é um exemplo de autenticação forte, que requer do DSA processamentos adicionais de segurança.

5.5.3.1. Procedimentos de Autenticação para o Diretório X.500 (X.509)

O serviço de diretório implementa características de segurança para prevenir acessos não autorizados [Bla91]. Ele deve conter nomes para autenticação e senhas para suportar outras aplicações. A recomendação ITU-T X.509 inclui este tipo de suporte ao Diretório, onde são definidos dois tipos de autenticação: autenticação simples e forte.

Autenticação simples

A autenticação simples [WL92] é voltada para suportar domínios simples, restritos ao uso de um único DUA com um ou dois DSAs. O procedimento opera como mostrado na figura 5.13 : (1) o usuário A envia para um usuário B seu DN e sua senha; (2) esta informação é passada para o Diretório e a senha é checada; (3) o Diretório informa ao usuário B se a senha do usuário A é válida ou inválida; (4) finalmente, o usuário B informa ao usuário A os resultados da operação.

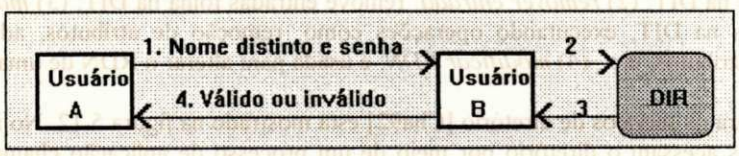


Figura 5.13. O mecanismo de autenticação simples [OSF90c]

Autenticação forte

Na autenticação forte [WL92] faz-se uso do conceito de chaves públicas, que é um conceito relativamente novo. A vantagem das chaves públicas com relação às chaves privadas é que elas são mais fáceis de serem administradas.

O conceito de chaves públicas está ilustrado na figura 5.14 [Sch94]. O usuário A conhece sua própria chave privada e também a chave pública do usuário B. Por sua vez, o usuário B conhece sua chave privada e a chave pública para o usuário A. Assim, o usuário A pode usar a chave pública do usuário B para criptografar dados a serem transmitidos para B. Por sua vez, o usuário B utiliza sua chave privada (que não está disponível para nenhum outro usuário) para decifrar os dados recebidos do usuário A. O processo reverso ocorre com o usuário B ao utilizar a chave pública do usuário A antes de transmitir os dados para o usuário A, o qual irá aplicar a sua própria chave privada para transformar os dados recebidos em texto legível.

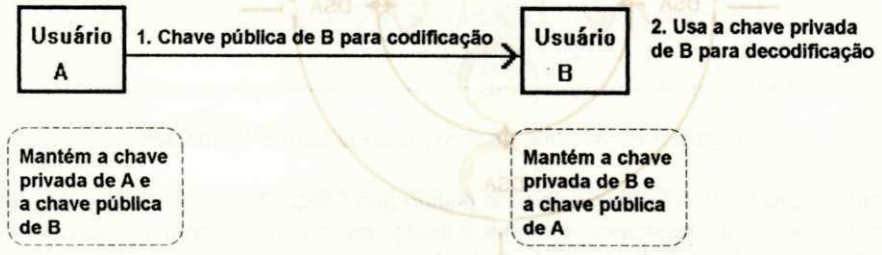


Figura 5.14. Chaves públicas [Sch94]

Assinaturas Digitais

No caso anterior, o processo se passa com o uso de chaves públicas para se codificar dados antes deles serem transmitidos, e uma chave secreta complementar, a chave privada, conhecida somente por seu dono, usada para decodificar os dados. Podemos inverter este processo para fornecer um outro poderoso mecanismo de autenticação chamado de *assinatura digital* [ACM92b].

A operação de uma assinatura digital está ilustrada na figura 5.15 . o usuário A emprega sua chave privada para criptografar uma assinatura digital (por exemplo, uma senha ou outra forma de identificação). Esta senha é transmitida para o usuário B, que possui a chave pública para o usuário A. Qualquer usuário que conheça a chave pública do usuário A poderá decodificar os dados, porém, somente o usuário A poderá executar a codificação complementar.

O usuário A informa em primeiro lugar ao usuário B que ele realmente é o usuário A. A seguir, ele envia os dados codificados. Depois, o usuário B usa a chave pública do usuário A para decifrar os dados enviados. Se o usuário A afirmar ser ele um outro usuário, ele não terá a chave privada apropriada para criar a assinatura digital. Desta maneira, um usuário pode ser autenticado e a fonte de informações pode ser verificada.

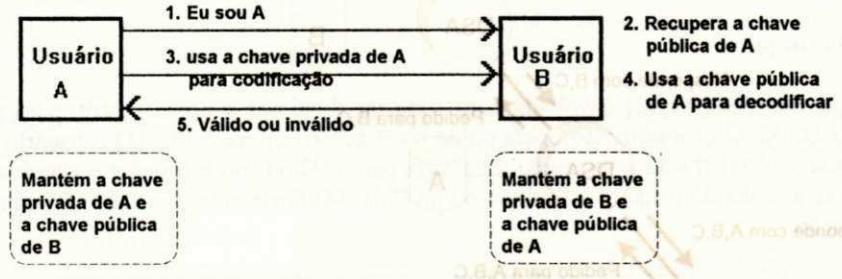


Figura 5.15. Assinaturas Digitais [Lam91]

5.5.4. A Estrutura do Diretório X.500

A distribuição da estrutura do diretório é uma necessidade. A distribuição física normalmente reflete a distribuição lógica da DIT. A DIB completa é subdividida em várias partições que se relacionam com a estrutura da DIT. Um DSA é associado com cada partição, de forma a fornecer acesso a ela. O esquema geral está apresentado na figura 5.16 .

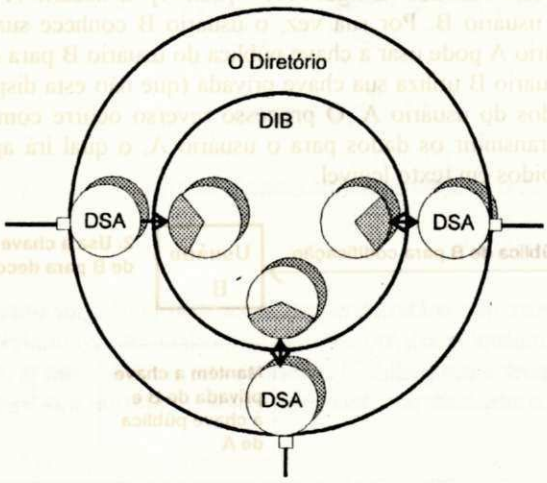


Figura 5.16. A estrutura do Diretório [Hal93b]

Quando é recebido um pedido de um de seus DUAs locais, um DSA procura primeiramente em sua partição da DIT para buscar as informações pedidas [Yem93]. Se a informação estiver presente, o DSA responde diretamente para a DUA. Senão, o pedido deve ser passado para outro DSA de nível superior - chamado de referência ("*referral*"). Como se pode ver, o procedimento é o mesmo usado no DNS. Adicionalmente, o DSA suporta dois procedimentos de busca: *encadeamento* e *multicasting*.

No **encadeamento** [Hal93b], se um DSA receber um pedido para o qual ele só tem parte da informação desejada, ele cria um pedido para o restante da informação e o envia para o DSA que ele considera que possa ter a informação. A resposta para um pedido encadeado deve fazer sempre o caminho original em ordem inversa. O DSA recebedor combina as informações recebidas com suas próprias informações, e retorna a resposta final para o DUA no final da cadeia. O esquema geral está mostrado na figura 5.17 .

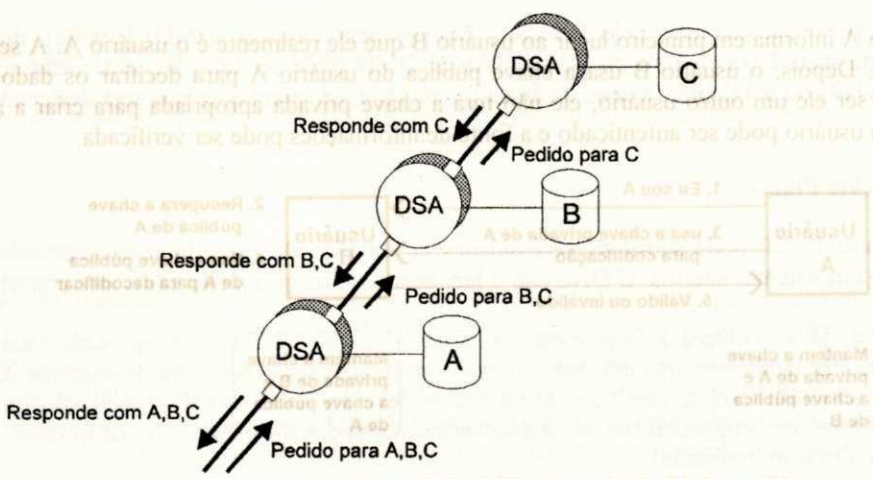


Figura 5.17. A operação de encadeamento [Hal93b]

Em **multicasting** [Hal93b], se o DSA não possuir as informações pedidas relacionadas com a sua partição DIB, ele passa adiante uma cópia do pedido para todos os outros DSAs que ele tenha conhecimento. Da mesma forma como ocorre no DNS, o DSA mantém em cache os endereços de rede para todos os DSAs no sistema. Assim, desde que um destes DSAs possua a resposta desejada, ela é enviada para o DSA original, e dele para o usuário via DUA, como mostrado na figura 5.18 .

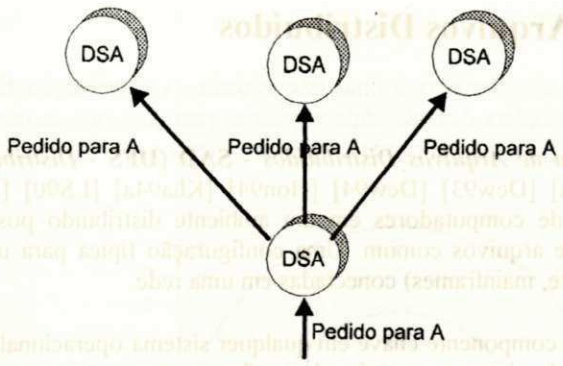


Figura 5.18. A operação de multicasting [Hal93b]

5.5.4.1. A Operação do Diretório Distribuído

A arquitetura do X.500 assegura que o Diretório possa ser distribuído por uma vasta área geográfica. Um DUA pode interagir com um ou com múltiplos DSAs, e os DSAs podem intercomunicar-se com outros DSAs por meio das referências para poderem atender a um pedido. Lembre-se que o DSA é um processo de aplicação OSI. As comunicações entre DUAs e DSAs é controlada por dois protocolos, conforme mostrado na figura 5.19 ([Bla91] [Cha92]).

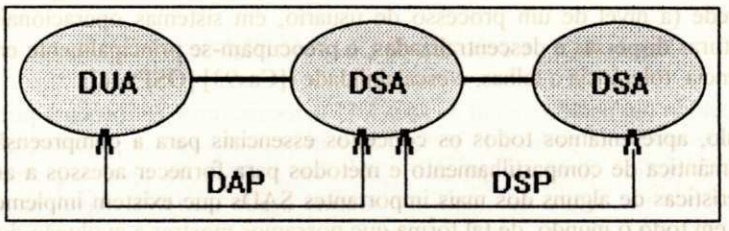


Figura 5.19. Os protocolos de comunicações X.500 [Bla91]

- *Protocolo de Acesso ao Diretório* (**DAP** - *Directory Access Protocol*): especifica as ações entre o DUA e o DSA de forma a permitir que o DUA tenha acesso ao Diretório.
- *Protocolo de Sistema de Diretório* (**DSP** - *Directory System Protocol*): especifica as ações entre DSAs.

O Diretório é administrado pelo Directory Management Domain (DMD) [Cha92], que consiste de um conjunto de um ou mais DSAs e zero ou mais DUAs. O DMD pode ser um país, uma instituição governamental (como a EMBRATEL ou a Telebrás, a nível de Brasil), uma rede, ou quem quer que seja designado para administrar a DIB.

5.5.5. Considerações Finais

O Diretório X.500 pode suprir as necessidades de interligações de alto nível necessárias para permitir a interoperabilidade em sistemas abertos. O DNS não o faz, pois só abrange aqueles sítios interconectados à Internet.

Na prática, já se começa a implementar o Diretório, que irá fornecer aos seus usuários não somente endereços na rede, mas também outras informações como endereços residenciais ou os números de telefones. Assim como no DNS, a administração de partições de diretórios deve ser local a cada organização, porém não se pode assegurar ainda se estes serviços trabalham adequadamente em grandes sistemas (como na Internet, onde com certeza o DNS funciona muito bem, obrigado).

Atualmente, apesar de existirem implementações comerciais do X.500, não existem grupos dedicados ao desenvolvimento de facilidades de endereçamento e nomes, como ocorre no mundo Internet. Por exemplo, na Universidade da California, em Berkeley, existe um grupo chamado *Computer Sciences Research Group* [Goe93a], patrocinado pelo governo americano e responsável pelo desenvolvimento do próprio DNS que hoje é uma facilidade de domínio público. Vamos esperar mais alguns anos para ver o X.500 funcionado a pleno vapor.

Capítulo 6: Sistemas de Arquivos Distribuídos

6.1. Introdução

O propósito de um ***Sistema de Arquivos Distribuídos*** - **SAD (DFS - *Distributed File System*)** ([Bil94] [Blo92] [BRI94] [Cas93] [Com91a] [Dew93] [Dew94] [Hon94] [Kha94a] [LS90] [SUN92a] [Ste91] [Tan92] [TVR85]) é permitir que usuários de computadores em um ambiente distribuído possam compartilhar dados e recursos por meio de um sistema de arquivos comum. Uma configuração típica para um SAD consiste de várias estações de trabalho (e, eventualmente, mainframes) conectadas em uma rede.

O sistema de arquivos é um componente chave em qualquer sistema operacional ([Mil87] [Don92] [Tan92] [Mae87]). Ele é justamente a parte do sistema operacional que fornece armazenamento de longa duração para os dados. É o sistema de arquivos que cuida dos objetos armazenados, de tal forma que eles possam ser referenciados por um nome. O sistema de arquivos deve garantir que um arquivo e seu nome existam desde o momento em que forem criados, até o momento em que forem descartados.

Um sistema de arquivos local é aquele que reside inteiramente em um único computador, e é inacessivel a partir de outros computadores. Normalmente, o meio de armazenamento utilizado são os discos magnéticos, que fornecem armazenamento rápido, barato e confiável. Os sistemas de arquivos em sistema operacionais de compartilhamento de tempo e centralizados são chamados de *sistemas de arquivos convencionais* ([Mil87] [Mae87]).

Por sua vez, os SADs, que são implementados como parte do sistema operacional em cada computador conectado em uma rede (a nível de um processo de usuário, em sistemas operacionais com micro-núcleos), são adequados para estruturas dispersas e descentralizadas, e preocupam-se principalmente em fornecer soluções para as questões de transparência, tolerância a falhas, e escalabilidade ([Cas93] [OSF91c]).

Neste capítulo, apresentamos todos os conceitos essenciais para a compreensão dos SADs, incluindo as alternativas para a semântica de compartilhamento e métodos para fornecer acessos a arquivos remotos, e falamos das principais características de alguns dos mais importantes SADs que existem implementados em universidades e em grandes empresas em todo o mundo, de tal forma que possamos mostrar a evolução dos SADs nos últimos anos.

Os SADs são baseados no modelo cliente/servidor [Ull93]. Um ou mais servidores de arquivos cooperam com clientes do sistema de arquivos, de tal forma que estes clientes possam acessar os arquivos gerenciados pelos servidores. Com as preocupações de controle distribuídas entre clientes e servidores, muitos dos objetivos dos sistemas de arquivos centralizados, que são de fácil manutenção, tornam-se mais complicados. Para começarmos nossa discussão, é importante que se faça a distinção, nos sistemas distribuídos, entre os conceitos de serviço de arquivos e servidor de arquivos. Abordamos este tópico na próxima seção.

6.1.1. Clientes, Serviços e Servidores de Arquivos

O ***serviço de arquivos*** ([UI89b] [Ste91] [Hon94]) é a especificação daquilo que os sistemas de arquivos oferecem aos seus clientes. Descreve as primitivas disponíveis, que parâmetros elas utilizam, e as ações que elas executam. Para os clientes, o serviço de arquivos define exatamente com quais serviços eles podem contar, porém nada é fornecido a respeito de sua implementação. Assim o serviço de arquivos define as interfaces do sistema de arquivos para os clientes.

O ***servidor de arquivos*** [Mil94a] por sua vez, é um processo que executa em alguma máquina, e que cuida da implementação dos serviços de arquivos. Um sistema pode ter um ou vários servidores de arquivos, de tal forma que os clientes não precisem tomar conhecimento a respeito deste fato. Para os clientes, o que lhes interessa é que quando eles executarem uma das primitivas do serviço de arquivos, o trabalho requisitado será efetuado de alguma maneira, e os resultados lhes serão devolvidos. O ideal é que os clientes nem ao menos tomem conhecimento de que o serviço de arquivos é distribuído, e este deve comportar-se da mesma maneira que um sistema de arquivos centralizado.

Como o servidor de arquivos é simplesmente um processo de usuário que fica executando em uma estação, um sistema pode ter múltiplos servidores de arquivos, com cada um oferecendo um serviço de arquivos diferente [Bil94]. Por exemplo, um sistema distribuído pode ter dois servidores que oferecem serviços de arquivos Unix e MS/DOS, respectivamente, com cada processo de usuário usando aquele que for apropriado para ele.

Um ***cliente*** [Hon94] de serviços de arquivos é um processo que pode invocar um serviço através de um conjunto de operações que formam a interface do cliente. Estas interfaces são implementadas para serem usadas por aplicações de alto nível ou diretamente por usuários humanos. Uma interface cliente para um serviço de arquivos é implementada por um conjunto de operações de arquivos. As principais primitivas são [Dew93]: *criação de arquivo, remoção de arquivo, leitura de arquivo, gravação em um arquivo.*

A implementação dos SADs pode variar: existem configurações onde servidores rodam em máquinas dedicadas, e existem configurações onde uma máquina tanto pode ser cliente quanto servidora. Um SAD pode ser implementado como parte do sistema operacional, ou alternativamente, como uma camada de software cuja tarefa é de gerenciar a comunicação entre as funções do sistema operacional em si e aquelas dos sistemas de arquivos. Adiante, abordamos as considerações de projeto para sistemas de arquivos distribuídos.

6.2. Tendências e Terminologia

Como dissemos há pouco, um SAD deve parecer para seus clientes como se fosse um sistema de arquivos convencional, de forma que a multiplicidade e a dispersão de servidores possa ser transparente para eles. Portanto, uma propriedade fundamental para os SADs é chamada de *transparência de rede* [LS90], e implica que os clientes devem estar aptos a acessarem arquivos remotos usando o mesmo conjunto de operações aplicáveis aos seus arquivos locais. Esta propriedade exige a disponibilização da *mobilidade dos usuários* [LS90] onde clientes podem se conectar a qualquer máquina no sistema.

Uma das grandes vantagens dos SADs com relação aos sistemas de arquivos convencionais é o potencial para suportar mecanismos de tolerância a falhas [Nel90] e escalabilidade [How88], graças à multiplicidade de recursos disponíveis. Estas duas características exigem definições de projeto onde se dê atenção à distribuição de controle e dos dados. Qualquer entidade centralizada, seja um controlador central, seja um repositório de dados central, irá introduzir pontos de falhas severas, e possíveis gargalos de desempenho [Tan92]. Portanto, um SAD escalável e tolerante a falhas deve ter servidores múltiplos e independentes [Ric94].

6.3. O Projeto de Sistemas de Arquivos Distribuídos

Os principais componentes de um SAD são o próprio serviço de arquivos e o serviço de diretórios. O serviço de arquivos, como já vimos, está relacionado com as operações e arquivos individuais, como a leitura, gravação, ou remoção. O serviço de diretórios cuida da criação e manutenção de diretórios, e da adição e remoção de entradas no diretório, dentre outras.

6.3.1. A Interface de Serviço de Arquivos

Podemos dividir os serviços de arquivos em dois tipos ([UI89b] [Ste91] [Hon94]), dependendo do tipo de modelo suportado: modelo carga-ascendente/carga-descendente ("download/upload") ou modelo de acesso remoto.

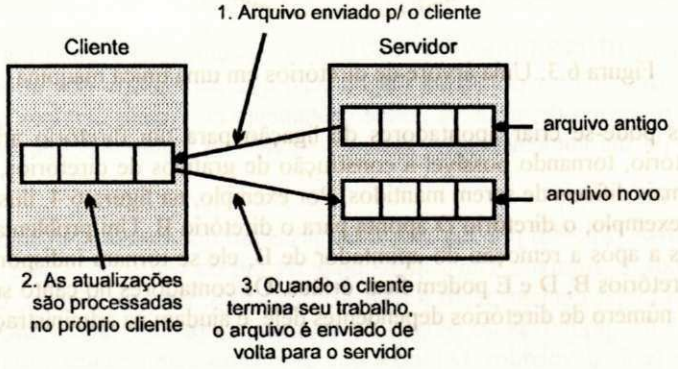


Figura 6.1. O modelo de carga-ascendente/carga-descendente [Dew93]

No ***modelo de carga-ascendente/carga-descendente***, mostrado na figura 6.1, o serviço de arquivos fornece somente dois tipos de operações: ler o arquivo ou gravar o arquivo. Este modelo conceitual prevê a transferência de arquivos inteiros em ambas as direções, ou seja, do servidor para o cliente, e vice-versa. Os arquivos podem ser armazenados em memória ou em discos locais, dependendo do uso. É um tipo de tratamento geralmente empregado por servidores de arquivos em redes locais de microcomputadores [Sal93], e sua principal vantagem é sua simplicidade. A principal desvantagem fica por conta da necessidade de capacidade de armazenamento adicional no cliente, e outra é que como mesmo quando somente parte do arquivo é necessário no cliente, todo o arquivo é transferido, o que causa problemas de congestionamento na rede local.

No ***modelo de acesso remoto*** [Dew93], ilustrado na figura 6.2, o serviço de arquivos fornece uma variedade de operações para abrir e fechar arquivos, ler e gravar partes de arquivos, modificar atributos de arquivos, dentre várias outras. Nesta configuração, os servidores rodam somente em estações dedicadas, e não em clientes, com a vantagem clara de não haver necessidade de espaço adicional nas estações clientes, e eliminando o tráfego de dados desnecessário na rede.

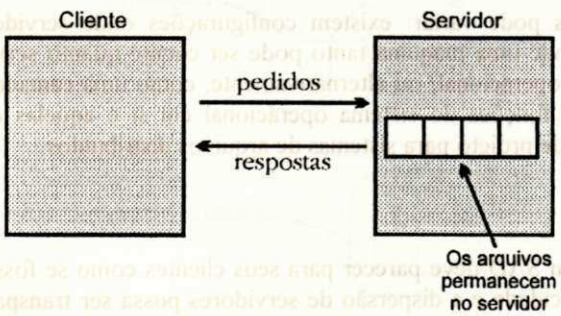


Figura 6.2. O modelo de acesso remoto [Dew93]

6.3.2. A Interface do Servidor de Diretórios

O ***serviço de diretórios*** ([Hon94] [LS90]) fornece operações para a criar e remover diretórios, nomear e renomear arquivos, e mover arquivos de um diretório para outro. Diretórios são compostos de sub-diretórios, para tornar possível o agrupamento de arquivos relacionados. Os sub-diretórios por sua vez podem conter seus próprios sub-diretórios, formando uma estrutura conhecida como ***árvore de diretório***, que compõe um *sistema de arquivos hierárquico* [Ste91]. A figura 6.3 ilustra uma árvore com cinco diretórios.

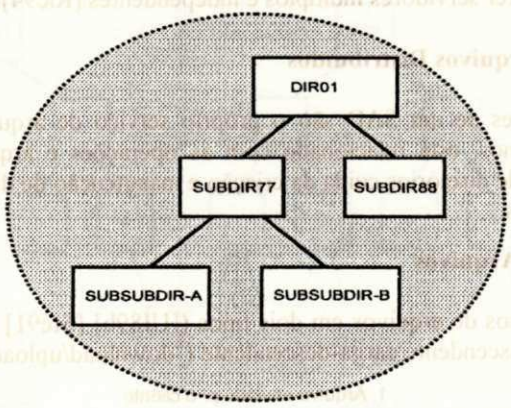


Figura 6.3. Uma árvore de diretórios em uma única máquina

Em alguns sistemas pode-se criar apontadores de ligação para um diretório arbitrário. Eles podem ser colocados em qualquer diretório, tornando possível a construção de gráficos de diretórios, que são mais poderosos que as árvores, porém bem mais difíceis de serem mantidos. Por exemplo, na figura 6.4 ilustramos o conceito de um gráfico de diretórios. Neste exemplo, o diretório D aponta para o diretório B. Um problema pode surgir na remoção da ligação de A para B, pois a após a remoção do apontador de B, ele se tornará indisponível a partir do diretório raiz, o que implica que os diretórios B, D e E podem ficar órfãos. Os contadores no canto superior esquerdo de cada diretório na figura indicam o número de diretórios dependentes dele, e ajudam na administração dos apontadores.

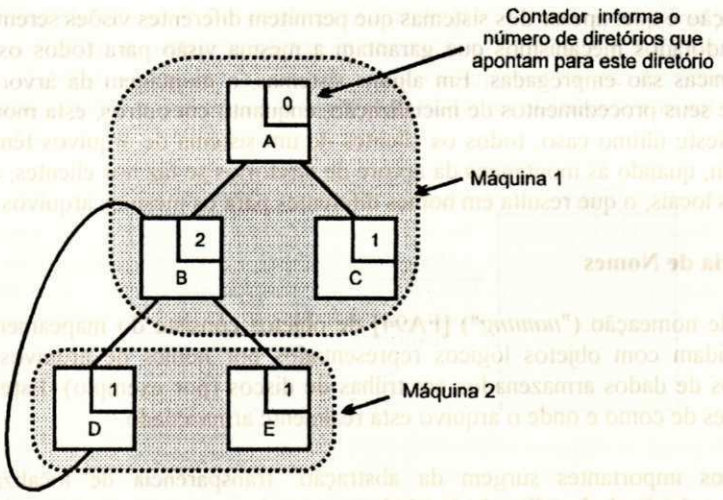


Figura 6.4. Um gráfico de diretórios em duas máquinas [Tan92]

Uma consideração de projeto básica para qualquer sistema de arquivos distribuídos é se todas as máquinas devem ou não ter a mesma visão da hierarquia de diretórios. Por exemplo, digamos que em uma instalação tenhamos dois servidores de arquivos, cada um deles suportando três diretórios e alguns arquivos, como mostrado na figura 6.5.(a) . Na figura 6.5 (b) temos um sistema onde todos os clientes têm a mesma visão do sistema de arquivos distribuído. Em contraste, na figura 6.5 (c), diferentes máquinas podem ter diferentes visões do sistema de arquivos. Este último caso é mais comum em sistemas que tratam múltiplos servidores de arquivos por meio de operações de montagem ("mount") remotas.

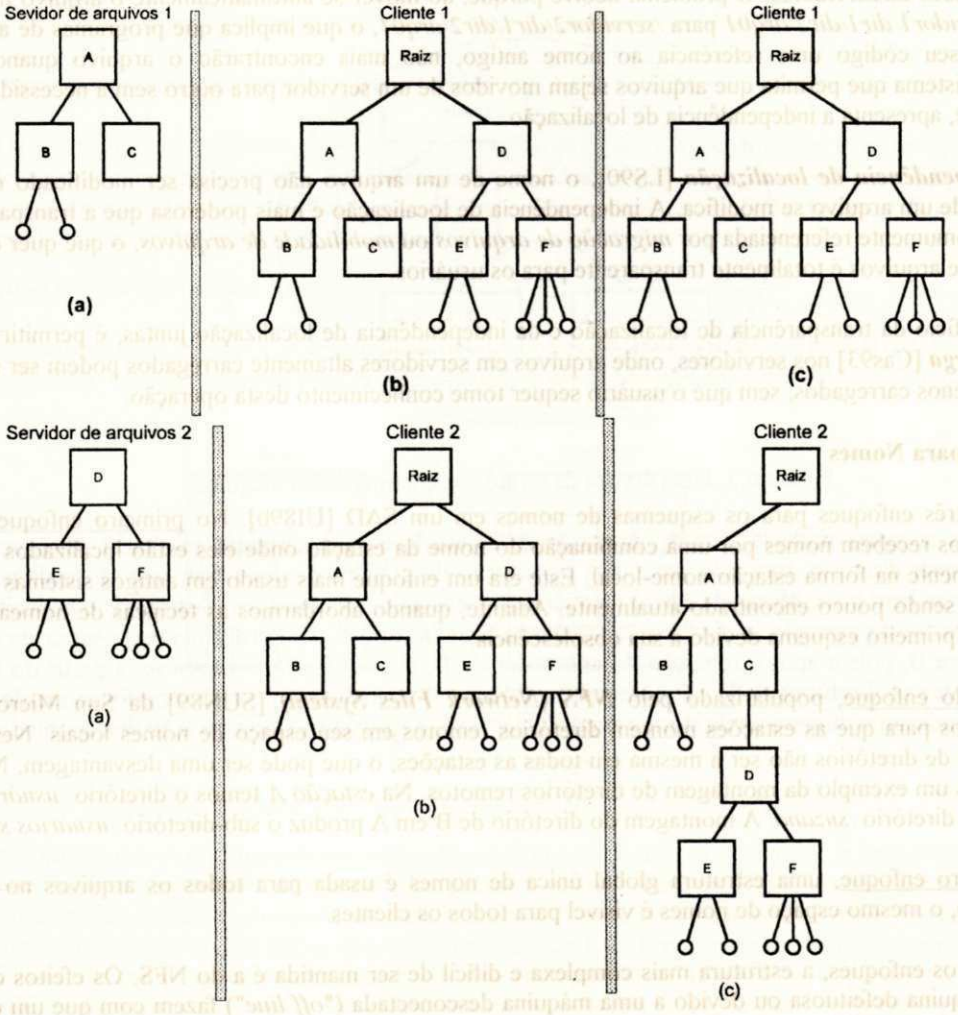


Figura 6.5 (a) Dois servidores de arquivos; (b) clientes com a mesma visão do SAD
(c) clientes com diferentes visões do mesmo SAD [Han94]

A recomendação é que, apesar dos sistemas que permitem diferentes visões serem mais flexíveis e naturais, o melhor é que sejam adotados mecanismos que garantam a mesma visão para todos os usuários e processos. Na prática, ambas as técnicas são empregadas. Em alguns sistemas, a montagem da árvore de diretórios é feita nos clientes como parte de seus procedimentos de inicialização, enquanto em outros, esta montagem é feita nos próprios servidores [Han94]. Neste último caso, todos os clientes de um sistema de arquivos têm uma visão consistente de todos os nomes. Porém, quando as montagens da árvore de diretórios se faz nos clientes, um mesmo objeto pode ser montado em diferentes locais, o que resulta em nomes diferentes para os mesmos arquivos.

6.3.3. A Transparência de Nomes

O processo de nomeação ("*naming*") [FA94] de objetos consiste do mapeamento entre objetos lógicos e físicos. Os usuários lidam com objetos lógicos representados por nomes de arquivos, enquanto que o sistema manipula blocos físicos de dados armazenados em trilhas de discos (por exemplo). Este mapeamento fornece uma abstração sobre detalhes de como e onde o arquivo está realmente armazenado.

Dois conceitos importantes surgem da abstração: transparência de localização e independência de localização. Na ***transparência de localização*** [LS90], o nome de um arquivo não revela qualquer pista que possa indicar a sua localização física de armazenamento. Por exemplo, um nome do tipo */servidor1/dir1/dir2/arq01* informa explicitamente que o arquivo *arq01* está localizado no servidor *servidor1*, porém não informa onde este servidor está localizado, de tal forma que este servidor pode ser transferido para qualquer estação na rede, sem a necessidade de ter-se que modificar o seu caminho de acesso. Portanto, este sistema apresenta transparência de localização.

Consideremos agora um caso onde o arquivo *arq01* do exemplo tenha que migrar para outro servidor, digamos o *servidor2*. Infelizmente, quando a primeira parte de um nome é o próprio nome do servidor, o sistema não terá a habilidade para mover o arquivo *arq01* para outro servidor automaticamente, mesmo que existam os diretórios *dir1* e *dir2* em ambos os servidores. O problema ocorre porque, ao mover-se automaticamente o arquivo muda-se o seu nome de */servidor1/dir1/dir2/arq01* para */servidor2/dir1/dir2/arq01*, o que implica que programas de aplicações que tenham em seu código uma referência ao nome antigo, não mais encontrarão o arquivo quando forem executados. Um sistema que permita que arquivos sejam movidos de um servidor para outro sem a necessidade de se alterar o seu nome, apresenta a independência de localização.

Na ***independência de localização*** [LS90], o nome de um arquivo não precisa ser modificado quando a localização física de um arquivo se modifica. A independência de localização é mais poderosa que a transparência de localização, e é comumente referenciada por *migração de arquivos* ou *mobilidade de arquivos*, o que quer dizer que a movimentação de arquivos é totalmente transparente para os usuários.

Um benefício da transparência de localização e da independência de localização juntas, é permitir o ***ajuste automático de carga*** [Cas93] nos servidores, onde arquivos em servidores altamente carregados podem ser migrados para servidores menos carregados, sem que o usuário sequer tome conhecimento desta operação.

6.3.4. Esquemas para Nomes

Existem três enfoques para os esquemas de nomes em um SAD [UI89b]. No primeiro enfoque, o mais simples, os arquivos recebem nomes por uma combinação do nome da estação onde eles estão localizados mais um nome local, geralmente na forma estação:nome-local. Este era um enfoque mais usado em antigos sistemas de redes de computadores, sendo pouco encontrado atualmente. Adiante, quando abordarmos as técnicas de nomeação, não consideramos este primeiro esquema devido à sua obsolescência.

O segundo enfoque, popularizado pelo ***NFS (Network Files System)*** [SUN89] da Sun Microsystems, fornece mecanismos para que as estações montem diretórios remotos em seu espaço de nomes locais. Neste caso, ocorre de a árvore de diretórios não ser a mesma em todas as estações, o que pode ser uma desvantagem. Na figura 6.6. apresentamos um exemplo da montagem de diretórios remotos. Na *estação A* temos o diretório */usuários*, e na *estação B* temos o diretório */suzana*. A montagem do diretório de B em A produz o sub-diretório */usuários/suzana*.

No terceiro enfoque, uma estrutura global única de nomes é usada para todos os arquivos no sistema. Conseqüentemente, o mesmo espaço de nomes é visível para todos os clientes.

De todos os enfoques, a estrutura mais complexa e difícil de ser mantida é a do NFS. Os efeitos causados devidos a uma máquina defeituosa ou devido a uma máquina desconectada ("*off line*") fazem com que um conjunto arbitrário de diretórios em diferentes máquinas fiquem indisponíveis. Da mesma forma, a migração de arquivos de uma máquina para outra irá requerer mudanças nos espaços de nomes de todas as máquinas afetadas.

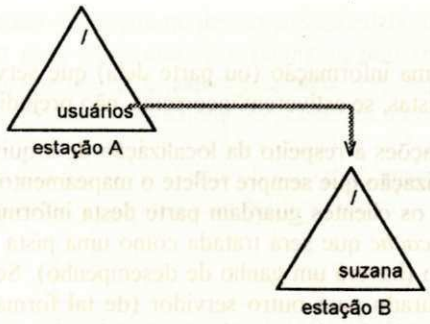


Figura 6.6. A montagem de diretórios [Hon94]

A seguir, apresentamos as principais técnicas empregadas no estabelecimento de nomes em um SAD.

6.3.4.1. A Tradução de Caminhos para Nomes

O mapeamento de nomes textuais para identificadores de baixo nível se faz por um procedimento recursivo de pesquisa ("*lookup*") ([Tan92], [LS90], [Hon94]). Vejamos um exemplo de tradução de caminhos para nomes ("*pathnames*") na figura 6.7 , que mostra uma estrutura de nomes parcial distribuída por três servidores, usando o terceiro esquema apresentado anteriormente (*estrutura única de nomes globais*). No nosso exemplo, adaptado de [Tan92], temos um sistema no qual o diretório corrente, no *servidor1*, contém uma entrada ***a*** para um outro diretório, no *servidor2*, o qual de maneira similar possui uma entrada ***b*** para um diretório no *servidor3*. O terceiro diretório contém uma entrada ***c*** para um arquivo, juntamente com o nome binário deste arquivo.

Para pesquisar */a/b/c*, (ou seja, para fazer a tradução do caminho do nome), um cliente envia uma mensagem para o *servidor1*, o qual cuida do diretório corrente deste cliente. O *servidor1* encontra a entrada ***a***, porém verifica que ela na realidade representa um apontador para outro servidor. Neste momento, existem duas possibilidades de implementações para a tradução do caminho de nomes ([LS90] [Hon94]): a pesquisa interativa, ou a pesquisa automática.

Na pesquisa interativa, o *servidor1* informa de volta para o cliente qual dos servidores guarda ***b***, e o próprio cliente continua com a pesquisa do restante do caminho ***b/c***. Este esquema exige que o cliente saiba que servidor armazena qual diretório, e requer um maior número de troca de mensagens, como mostrado na figura 6.7 (a) .

Na pesquisa automática, o servidor *servidor1* pode passar adiante a pesquisa para o *servidor2*, sem responder de imediato ao cliente. Este método é mais eficiente, porém não pode ser implementado simplesmente pelo uso de RPCs [Cha92], pois o processo para o qual o cliente envia a mensagem não é o mesmo de quem virá a resposta (muito provavelmente). A figura 6.7 parte (b) ilustra o conceito. A maioria dos SADs implementam de alguma forma uma destas duas pesquisas apresentadas.

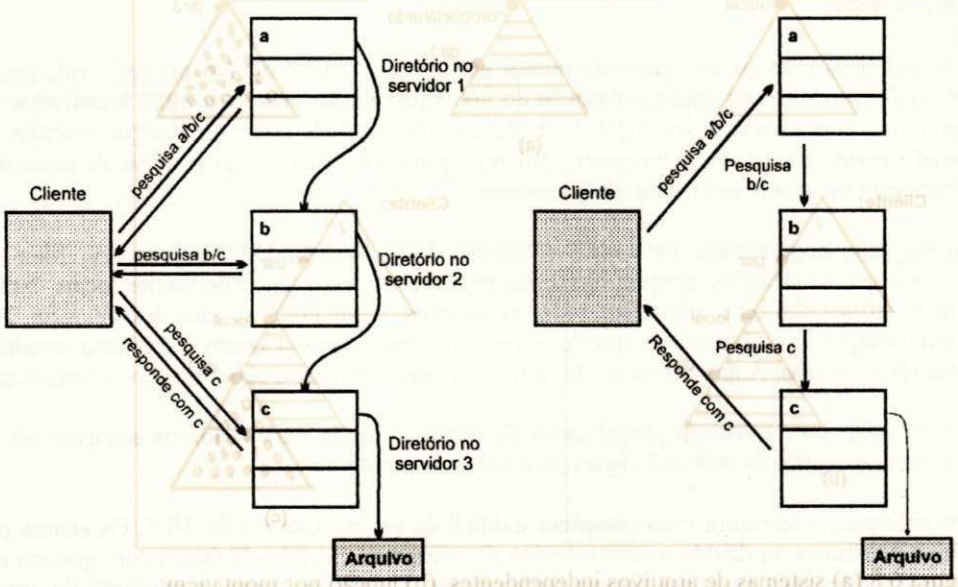


Figura 6.7. (a) Uma pesquisa de nomes interativa. (b) uma pesquisa de nomes automática [Bil94]

6.3.4.2. Pistas

Uma *pista* ("hint") [TVR85] é uma informação (ou parte dela) que serve para acelerar o desempenho do processo de tradução de nomes [LS90]. Pistas, se estiverem incorretas, não prejudicam a semântica da busca.

Para ilustrarmos como as informações a respeito da localização de arquivos são tratadas nas pistas, vamos considerar que exista um servidor de localização que sempre reflete o mapeamento correto e completo da localização de arquivos. Vamos assumir também que os clientes guardam parte desta informação de mapeamento em memória *cache*. É exatamente esta informação em *cache* que será tratada como uma pista. Se em uma pesquisa, um arquivo for encontrado pelo uso desta pista, ótimo (haverá um ganho de desempenho). Senão, se a pista for invalidada, por digamos, uma migração do arquivo procurado para outro servidor (de tal forma que a pista ficou desatualizada), então o cliente deve buscar o arquivo por meio de uma pesquisa convencional.

6.3.4.3. O Mecanismo de Montagem

O ***mecanismo de montagem*** [SUN89] agrupa sistemas de arquivos remotos para criar uma estrutura de nomes global. A operação de montagem liga a raiz de um sistema de arquivos a um diretório em outro sistema de arquivos. O diretório que serve para juntar os dois sistemas de arquivos é chamado de *ponto de montagem*. Todos os pontos de montagem são gravados pelo núcleo do sistema operacional em uma *tabela de montagem*. Depois de terminada a operação de montagem, os arquivos no sistema de arquivos remoto podem ser acessados localmente como se fossem descendentes do ponto de montagem no diretório.

Vejamos um exemplo na figura 6.8 . Na figura 6.8 (a), são mostrados os sistemas de arquivos independentes que pertencem às máquinas cliente, servidor1, e servidor2. Neste estágio, somente os arquivos locais podem ser acessados em cada máquina. Os triângulos representam sub-árvores de diretórios.

Na figura 6.8 (b), mostramos o efeito da montagem de ***servidor1:/usr/compartilhado*** sobre ***cliente:/usr/local***. A partir de agora, qualquer arquivo no diretório ***dir1*** (que é remoto) poderá ser acessado pelo prefixo /usr/local/dir1 na máquina ***cliente***. O diretório /usr/local original nesta máquina não é mais visivel (pois foi sobreposto).

Montagens cascateadas também são permitidas [LS90]. Desta forma, um sistema de arquivos pode ser montado sobre outro sistema de arquivos que já tenha, ele mesmo, sido montado anteriormente. Na figura 6.8 (c), ilustramos a montagem cascateada, dando continuidade ao nosso exemplo. A figura mostra o resultado da montagem de ***servidor2:/dir2/dir3*** sobre ***cliente:/usr/local/dir1***, o qual já foi montado previamente a partir do ***servidor1***. Arquivos no diretório ***dir3*** podem ser acessados na máquina cliente pelo uso do prefixo ***/usr/local/dir1***.

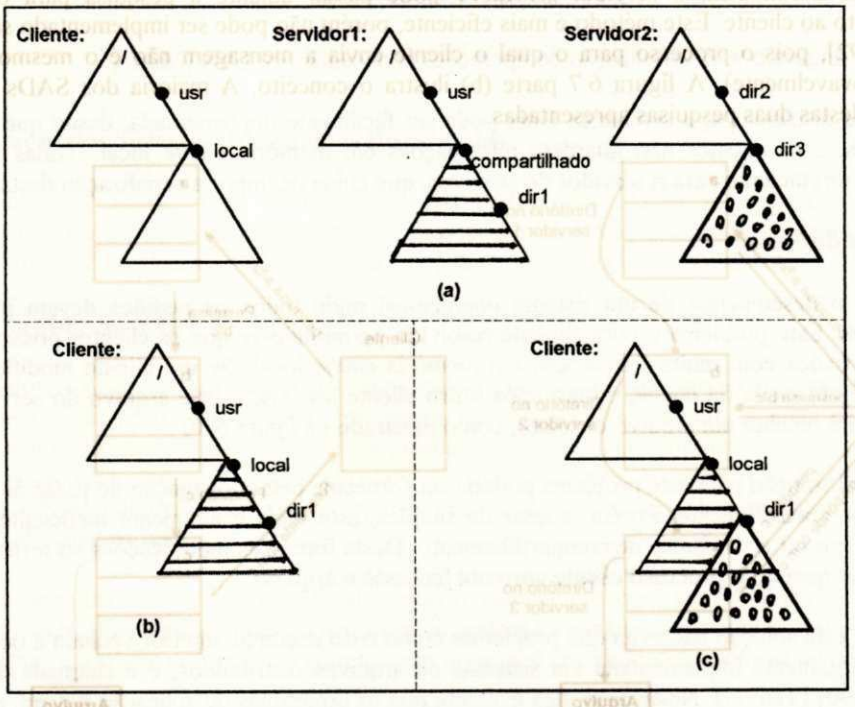


Figura 6.8 (a) sistemas de arquivos independentes; (b) junção por montagem; (c) montagem por cascateamento [LS90]

6.3.5. A Semântica de Compartilhamento de Arquivos

A ***semântica de compartilhamento de arquivos*** ([Bil94] [Com91a] [OV91] [CTR92a] [CTR92b] [Tan92] [LS90]) é uma característica do sistema que especifica os efeitos causados por acessos simultâneos, de múltiplos clientes, a um mesmo arquivo compartilhado. Em particular, esta semântica especifica quando as modificações de dados efetuadas por um determinado cliente, devem tornar-se visíveis para os demais clientes remotos.

Vamos assumir que uma série de acessos a arquivos (isto é, leituras e gravações) feitas por um cliente no mesmo arquivo, estarão sempre limitadas pelas operações de abrir arquivo ("open") e fechar arquivo ("close"). Chamamos esta série de acessos de uma ***sessão de arquivo*** [Bla90].

É interessante observarmos que as aplicações que utilizam o sistema de arquivos para armazenar dados e impor restrições aos acessos concorrentes, de forma a garantir a consistência semântica destes dados, devem elas mesmas fazer uso de mecanismos especiais (por exemplo, usar operações de bloqueio [Mil87]) para este propósito, e não devem confiar na semântica de compartilhamento fornecida pelo próprio sistema de arquivos.

6.3.5.1. A Semântica Unix

Em sistemas operacionais centralizados que permitem que vários processos compartilhem o mesmo arquivo, a semântica de compartilhamento empregada exige, normalmente, que quando uma operação de leitura ocorrer após uma operação de gravação, esta operação de leitura deva receber o valor recém gravado. Na figura 6.9 ilustramos o conceito. Da mesma maneira, se duas operações de gravação ocorrem em uma sucessão muito rápida, e são seguidas por uma operação de leitura, então o valor a ser lido deve ser o último valor gravado pela operação de gravação que ocorreu por último. Assim, o sistema força uma ordenação de tempo para todas as operações, de tal forma que o valor mais recente da informação seja sempre retornado. Este processo de ordenação é chamado de "serialização" [OV91] e este modelo é conhecido como ***semântica Unix*** [OSF91b] (por ser comumente encontrado nos sistemas operacionais Unix).

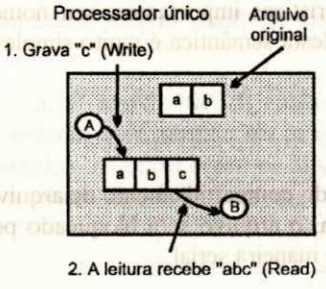


Figura 6.9. A serialização de operações com semântica Unix [Mil87]

Em sistemas distribuídos, a semântica Unix pode ser facilmente implementada, desde que exista um único servidor de arquivos, e os clientes não guardem informações em memória *cache* local. Todas as operações de leitura/gravação vão diretamente para o servidor de arquivos, que cuida de impor a serialização destas operações.

6.3.5.2 A Semântica de Sessão

Na prática, o desempenho de um sistema operacional onde todos os pedidos devam ir para um único servidor é muito ruim. Este problema é normalmente resolvido permitindo-se que os clientes possam manter cópias locais, de arquivos usados com muita freqüência, em memória *cache* local. Se um cliente modificar localmente o conteúdo do arquivo em memória *cache*, e logo após outro cliente for buscar este arquivo do servidor, certamente este segundo cliente irá receber um arquivo obsoleto, como mostrado na figura 6.10 .

Uma possível solução para este problema poderia ser fornecida pela propagação de todas as modificações de volta para o servidor imediatamente. Porém, apesar de simples, esta idéia é altamente ineficiente. Portanto, uma solução alternativa é relaxar a semântica de compartilhamento. Desta forma, as modificações só tornar-se-iam visíveis para os outros clientes quando o processo cliente corrente fechasse o arquivo.

Certamente, esta solução não evita que problemas como o do exemplo anterior venham a ocorrer. Apesar de tudo, esta regra é largamente implementada em sistemas de arquivos distribuídos, e é chamada de ***semântica de sessão*** ([Hon94] [LS90] [Tan92]). Neste caso, fica evidente que os programas de aplicação devem, eles mesmos, ser os responsáveis pela serialização dos acessos, de forma a coordenarem seus acessos explicitamente, sem precisarem confiar na semântica fornecida.

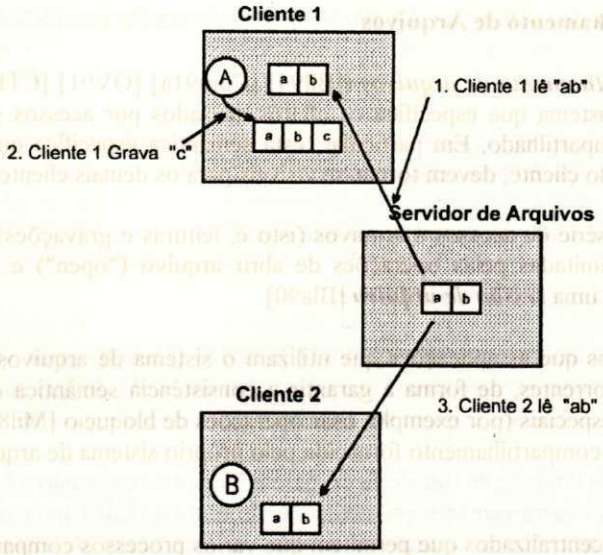


Figura 6.10. Em sistemas distribuídos com *caching*, valores obsoletos podem ser retornados para o cliente [Mil87]

6.3.5.3 A Semântica para Arquivos Compartilhados Imutáveis

A ***semântica de arquivos compartilhados imutáveis*** ([LS90], Hon94]) é um enfoque completamente diferente. Neste enfoque, desde que um arquivo tenha sido criado e declarado compartilhado por seu dono, ele não poderá mais ser modificado.

Um arquivo imutável possui duas características importantes: seu nome não pode ser reutilizado, e seu conteúdo não pode ser alterado. A implementação desta semântica é muito simples, pois o compartilhamento só será permitido para o modo de somente-leitura.

6.3.5.4 A Semântica de Transações

Uma outra possibilidade para o tratamento do compartilhamento de arquivos em um SAD é por meio do uso de transações atômicas ([Cla92b] [OV91]). Ou seja, o arquivo será bloqueado pela duração de uma sessão, de tal forma que as transações serão sempre executadas de maneira serial.

Na prática, as semânticas mais comumente implementadas em SADs são a semântica Unix e a semântica de sessão [Cas93].

6.4. A Implementação dos Sistemas de Arquivos Distribuídos

Descrevemos até agora várias características dos SADs com relação à perspectiva de como elas são apresentadas para os usuários. Nesta seção, examinamos como os SADs são implementados. Apresentamos os conceitos de "*caching*" e de replicação [Ges93a], que compõem a estrutura básica do sistema.

6.4.1. O Uso de *Caching* em SADs

Se os dados necessários para atender a um pedido por dados não estiverem disponíveis localmente na estação cliente, uma cópia destes dados pode ser trazida do servidor. Geralmente, a quantidade de informações trazidas é muito maior do que aquela realmente requisitada (por exemplo, arquivos inteiros ao invés de poucos blocos). Os acessos são então dirigidos à cópia em *cache* no lado do cliente. A idéia é reter os blocos mais acessados recentemente em *cache*, de tal forma que acessos repetidos para a mesma informação possam ser tratados localmente, sem gerar tráfego adicional na rede.

Quando uma cópia em *cache* é modificada, estas mudanças precisam ser refletidas na cópia mestre, e em todas as cópias em *cache*. Portanto, os acessos de gravação podem causar um *"overhead"* substancial. O problema da manutenção de cópias em *cache* consistentes com a cópia original é chamado de *problema de consistência de cache* [LS90].

No projeto de um esquema de *caching*, as seguintes decisões devem ser consideradas:

- a granularidade dos dados em *cache*;
- a localização da memória *cache*;
- como executar a propagação das atualizações;
- como determinar se os dados na *cache* estão consistentes.

A seguir falamos um pouco sobre cada uma destas decisões.

6.4.1.1. A Granularidade dos Dados em Memória *Cache*

A granularidade das informações em *cache* ([OV91] [Cla92b] [Cas93]) pode variar desde partes de um arquivo até um arquivo inteiro. Geralmente, mais dados são guardados em *cache* do que se precisa para atender a um único acesso, de tal forma que muitos acessos possam ser atendidos pelos dados em *cache*.

Calibrar o tamanho ideal da unidade de transferência para a memória *cache* não é tarefa simples. Aumentar o tamanho da unidade de *caching* irá incrementar a possibilidade de que os dados para o próximo acesso estejam armazenados localmente (isto é, aumenta-se a *taxa de acerto* da *localidade de referência*) ([Tan92] [Mil87] [Mae87]); por outro lado, o tempo requerido para a transferência dos dados e a possibilidade de que ocorram problemas de consistência, também serão proporcionalmente incrementados.

Normalmente, os esquemas de *caching* definem uma técnica chamada de leitura-adiantada ("read-ahead"), que é bastante útil para a leitura seqüencial de grandes arquivos. Blocos são trazidos do disco no servidor e armazenados em buffers tanto no servidor quanto no cliente, mesmo antes de serem requisitados, de tal maneira que acelera-se a leitura. Uma das vantagens no uso de grandes unidades de *caching* fica por conta da redução no "overhead" de tráfego na rede.

6.4.1.2. A Localização da Memória *Cache*

Em um ambiente distribuído que emprega o modelo cliente/servidor, existem quatro possibilidades para a alocação de arquivos, ou de partes destes arquivos [Dew94]: o disco do servidor; a memória principal do servidor; o disco na estação cliente (se estiver disponível); e a memória principal da estação cliente. A figura 6.11 ilustra estas possibilidades.

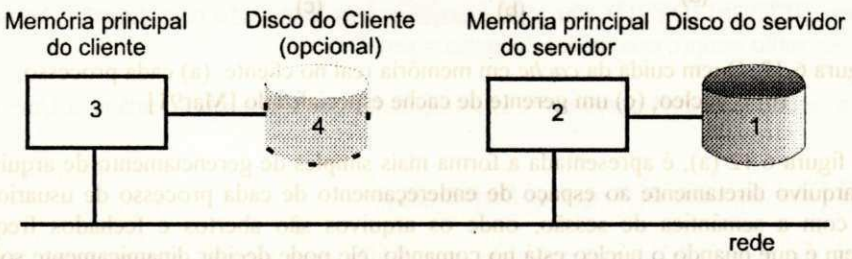


Figura 6.11. Possiveis localizações de arquivos na arquitetura cliente/servidor [Tan92]

Certamente, a localização original de todos os arquivos será o disco no servidor de arquivos (ou servidores, se for o caso). *Caches* em disco [Rei94b] apresentam uma vantagem clara: confiabilidade. Modificações nos dados em *cache* são perdidas, no caso de queda de energia, se os dados forem mantidos em memória real. Se os dados estiverem em disco, os dados continuarão lá durante a recuperação do servidor. O principal problema com o uso do disco no servidor refere-se às questões relacionadas com desempenho [How88]. Para que um cliente consiga ler um arquivo, este arquivo deve ser transferido do disco do servidor para a memória principal do servidor, e depois seguir pela rede para a memória principal da estação cliente.

Um ganho considerável de desempenho pode ser conseguido com a técnica de ***caching*** [Rei94b], isto é, as informações mais recentemente e freqüentemente acessadas no servidor devem permanecer na memória principal do servidor. Um cliente que deseje ler um arquivo que já esteja na memória do servidor irá ganhar tempo, pela eliminação da necessidade de um acesso ao disco. Por sua vez, a transferência pela rede ainda terá que ser feita, portanto, o tráfego não será eliminado.

As *caches* de memória real [Rei94b] apresentam várias vantagens. Em primeiro lugar, permitem estações cliente sem disco ("diskless"). Em segundo lugar, os dados podem ser acessados com maior rapidez da memória real com relação aos acessos a disco.

Vale a pena citarmos que as duas possibilidades para localização de memória *cache* (disco ou memória real) destacam diferentes funcionalidades. As *caches* em memória real enfatizam tempo de acesso reduzido; as *caches* em disco enfatizam um aumento na confiabilidade e autonomia para as máquinas individualmente. Em projetos de SADs modernos, o ideal é que sejam utilizadas grandes *caches* em memória real (no lugar de *caches* em disco), pois as mesmas são fáceis de serem implementadas e são totalmente transparentes para os clientes.

Além da possibilidade de *cache* em servidores, elas também podem existir nos clientes, em disco ou em memória real. Assim como nos servidores, a maioria dos cliente usa *caching* em memória real. Três opções de projeto determinam quem deve cuidar dos dados em *cache* de memória real em estações clientes, como ilustrado na figura 6.12 .

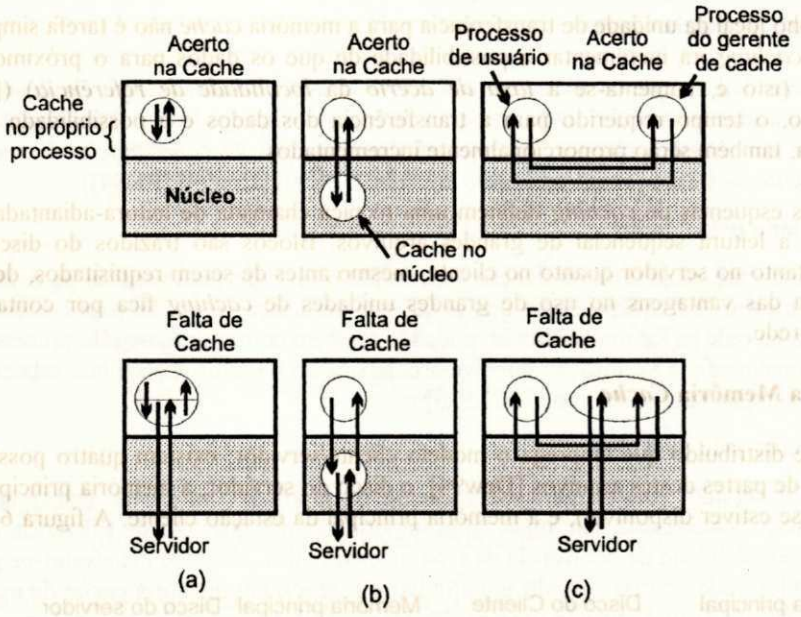


Figura 6.12. Quem cuida da *cache* em memória real no cliente. (a) cada processo; (b) o núcleo; (c) um gerente de cache especializado [Mar91]

No caso da figura 6.12 (a), é apresentada a forma mais simples de gerenciamento de arquivos em memória *cache*: liga-se cada arquivo diretamente ao espaço de endereçamento de cada processo de usuário. É uma técnica adequada para uso com a semântica de sessão, onde os arquivos são abertos e fechados freqüentemente por processos. A vantagem é que quando o núcleo está no comando, ele pode decidir dinamicamente sobre a quantidade de memória que precisa ser reservada para os programas propriamente ditos, e para a memória *cache*.

A segunda possibilidade de gerenciar a *cache* fica por conta do próprio núcleo do sistema operacional, como mostrado na figura 6.12 (b). A desvantagem óbvia é o "overhead" adicional gerado no núcleo.

A terceira alternativa é desenvolver um **gerente de *cache*** [Tan92] que rode em um processo separado, a nível de usuário. A vantagem é a liberação do núcleo para tarefas do próprio sistema operacional. Este tipo de solução é fácil de ser programada e é bem mais flexível. A figura 6.12 (c) ilustra os conceitos.

6.4.1.3. A Propagação das Atualizações

A seguir, usamos o termo *bloco sujo* para denotar um bloco de dados que tenha sido modificado na *cache* do cliente. Usamos também, o termo escoamento ("flush") para denotar a ação de se enviar blocos sujos para a cópia principal no servidor. A política usada para definir o escoamento de blocos sujos de volta para a cópia original no servidor afeta diretamente o desempenho e a confiabilidade dos SADs ([Ric94] [How88]). Podemos definir estas políticas em três: *grava-envia* ("write-through"), *grava-com-atraso* ("delayed-write"), e *grava-após-fechar* ("write-on-close").

A mais simples destas três políticas é a primeira: *grava-envia*. Os dados alterados na *cache* do cliente são imediatamente enviados para o servidor. A vantagem obtida é a confiabilidade, pois poucas informações são perdidas no caso de queda de um cliente. A desvantagem fica por conta de um desempenho que deixa a desejar.

A segunda opção, *grava-com-atraso*, atrasa as atualizações para a cópia mestre no servidor. As modificações são feitas na *cache* do cliente e enviadas mais tarde para o servidor. A vantagem é que as gravações ocorrem com maior rapidez, melhorando o desempenho no servidor. Infelizmente, os esquemas de gravação-com-atraso introduzem problemas na confiabilidade das informações, pois dados que ainda não tenham sido despachados para o servidor podem ser perdidos sempre que um cliente cair.

Finalmente, a política conhecida por *grava-após-fechar* grava os dados de volta para o servidor quando o arquivo é fechado. Esta é uma variação da política grava-com-atraso, que porém não traz vantagens de desempenho, pois o processo de serviço necessário para fechar os arquivos causa "overhead" adicional [How88]. O ganho de desempenho só ocorre quando o escoamento é mais freqüente para arquivos abertos por longos períodos de tempo ou para arquivos modificados com muita freqüência.

6.4.1.4. A Consistência da Memória *Cache*

Quando um cliente detecta que os dados em sua *cache* já estão desatualizados, ela deve ser descartada e novos dados devem ser buscados no servidor. Dois enfoques podem ser empregados na determinação da validade dos dados na memória *cache* em estações clientes, ou seja, na determinação da ***consistência da memória cache***: o *enfoque iniciado pelo cliente* ou *o enfoque iniciado pelo servidor* ([LS90] [Hon94] [Ric94]).

O Enfoque iniciado pelo Cliente

Neste enfoque, o cliente inicia uma verificação contatando o servidor para verificar se os dados locais estão consistentes com a cópia mestre. A freqüência com que se faz esta verificação é crucial para o sucesso do método, e pode gerar tráfego pesado na rede se for mal calibrada, podendo também consumir tempo de processamento precioso do lado do servidor. Normalmente, a verificação envolve a comparação de informações nos cabeçalhos ("headers") dos arquivos, como por exemplo, rótulos de tempo [Cla92b].

O Enfoque iniciado pelo Servidor

Neste enfoque, o servidor guarda para cada cliente os arquivos (ou partes deles) em sua *cache*. A manutenção destas informações adicionais tem implicações importantes nas questões de tolerância a falhas. Quando um servidor detecta uma possibilidade de ocorrência de uma inconsistência, ele deve manifestar-se para tentar evitá-la. Uma destas possibilidades pode ocorrer quando um arquivo é mantido em memória *cache* de maneira conflitante em dois clientes simultaneamente (isto é, quando um dos clientes especifica o modo de acesso de gravação).

Sempre que um servidor recebe um pedido para fechar um arquivo que tenha sido modificado, ele deve reagir e notificar a todos os clientes para descartar seus dados em *cache*, considerando-a inválida.

Um problema que pode surgir com o enfoque iniciado pelo servidor é que ele pode violar o modelo cliente/servidor tradicional, onde é da responsabilidade dos clientes iniciarem pedidos por serviços. Esta violação pode resultar em código complexo e irregular tanto para clientes quanto para servidores.

6.4.2. A Replicação de Arquivos

A ***replicação de arquivos*** [Ges93a] [Ric94] [Kor93]) é uma redundância útil para a melhoria da disponibilidade de arquivos. A replicação por múltiplas máquinas também pode beneficiar o desempenho, pois selecionar uma réplica próxima para servir a um pedido de acesso pode resultar em um menor tempo para o serviço. Nossa visão de replicação será a de que múltiplas cópias de determinados arquivos serão mantidas, e cada cópia em um servidor de arquivos separado.

A exigência básica para um esquema de replicação é que diferentes réplicas de um mesmo arquivo residam em máquinas independentes de falhas, isto é, a disponibilidade de uma réplica não pode ser afetada pela disponibilidade do restante das réplicas.

É interessante que os detalhes da replicação sejam escondidos dos usuários finais, e desta forma, deve-se implementar a transparência de localização em SADs preferencialmente. O controle de réplicas envolve a determinação do grau de replicação e alocação de réplicas. O principal problema associado com a replicação são as atualizações, e desta forma devemos avaliar cuidadosamente que aspectos devem ser privilegiados ao se usar a replicação de arquivos: a consistência, a disponibilidade dos dados, ou o desempenho [How88].

6.4.2.1. A Criação de Réplicas

Existem três maneiras de alocação para réplicas de arquivos ([Ges93a] [Hon94] [LS90]): (a) replicação explícita; (b) replicação lenta; e (c) replicação com grupos. Na *replicação explícita*, quando um processo constrói um arquivo, ele deve construi-lo em um servidor específico. A seguir, são feitas cópias adicionais em outros servidores. Este esquema pode gerar problemas de sincronização na criação das réplicas, de tal forma que o mesmo é pouco recomendado.

Na *replicação lenta*, somente uma cópia de cada arquivo é criada em um determinado servidor. A seguir, o próprio servidor cuida de construir réplicas em outros servidores automaticamente (via um processo que roda na retaguarda), sem que os programadores tomem conhecimento. O sistema deve fornecer esperteza suficiente para recuperar qualquer uma destas cópias se for necessário. Um problema que pode ocorrer é a alteração do arquivo original antes que as réplicas sejam construídas.

Na *replicação com grupos*, todas as operações de criação das cópias são transmitidas simultaneamente para todos os servidores ao mesmo tempo, de tal forma que as réplicas são criadas ao mesmo tempo que a cópia original. A figura 6.13 ilustra os três esquemas mais comumente usados na alocação de réplicas de arquivos.

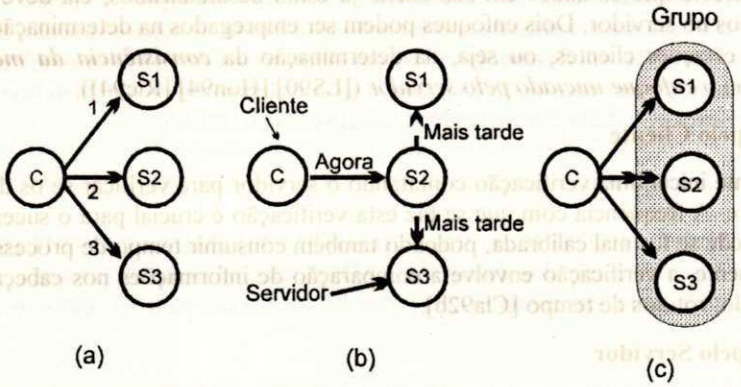


Figura 6.13. esquemas de alocação de réplicas: (a) replicação explícita; (b) replicação lenta; (c) replicação com grupos [Tan92]

No capítulo 11 sobre **Sistemas OLTP** [Cla92b] adiante, falamos com mais detalhes sobre o controle de concorrência em sistemas distribuídos. Especificamente, definimos conceitos como atomicidade de transações, protocolos de votação e o protocolo **2PC** (*two phase commit*) ([Cas93] [Dew93] [Dew94]), um dos mais eficientes para a garantia de consistência de informações em sistemas de arquivos distribuídos.

6.4.2.2. A Atualização das Réplicas

A disponibilidade de dados em um SAD é diretamente proporcional ao nível de controle de concorrência fornecido ([Hon94] [Tan92] [Byr94]). O principal enfoque para a melhoria da disponibilidade de dados é a replicação: se um dos servidores estiver indisponível, um servidor replicado pode atender ao pedido por serviço. Porém, quando atualizações concorrentes ocorrem, a existência de múltiplas cópias em diferentes sitios exige que algoritmos caros e complicados sejam implementados para garantir o controle da consistência [OV91].

Os mecanismos empregados para a manutenção da consistência dos dados em servidores replicados dependem de como as falhas são antecipadas [Nel90]. É possível, porém muito caro, que se acomodem os mais diversos tipos de falhas, onde servidores em falha continuam a responder aos pedidos, mesmo que de forma incompleta. Na prática, este modelo é relaxado, de tal forma que os servidores podem ser considerados como fora de operação de fato [Hon94]. Neste modelo, a reinicialização de um servidor que tenha caído necessita que se forneçam informações que estejam faltando ao arquivo quando ele estava sendo atualizado. Este esquema é geralmente implementado com o uso de um sitio mestre para cada arquivo, com as atualizações sendo transferidas destes sitios mestres para os sitios escravos que armazenam as réplicas. A figura 6.14 ilustra o conceito. Um cliente pode ler um arquivo em qualquer servidor, como msotrado em 6.14 (a) onde um servidor escravo atende ao pedido de leitura. As atualizações, no entanto, devem obrigatoriamente dirigir-se ao servidor mestre, como mostrado em 6.14 (b). Em alguns sistemas, podemos observar exatamente um servidor mestre para um determinado arquivo, enquanto em outros, cada cliente pode determinar seu próprio servidor mestre. As atualizações são então enviadas para servidores mestres, que são responsáveis, a seguir, pelo envio destas atualizações para os escravos. Se um servidor escravo estiver indisponível, ele deverá ser notificado após o seu reparo.

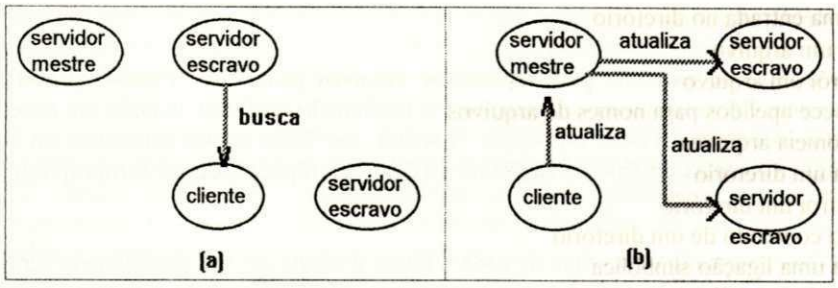


Figura 6.14. O esquema de servidores replicados mestre/escravo [Hon94]

6.5 Exemplos de Sistemas de Arquivos Distribuídos

Nesta seção examinamos exemplos de sistemas de arquivos distribuídos. Os dois primeiros que pretendemos apresentar, o **NFS** [SUN89] da Sun MicroSystems e o **AFS** [Coh93] da Transarc Co., são produtos bem conhecidos e que fazem muito sucesso nos ambientes comercias que implementam a computação distribuída, apesar do AFS ainda ser pouco usado na prática. Apresentamos também o **DFS** [OSF90b] da Open Software Foundation, que promete desempenhar um papel chave nos ambientes da computação distribuída. E adicionalmente, falamos sobre os sistemas de arquivos distribuídos que estão em desenvolvimento em laboratórios de pesquisa e universidades. Todos estes sistemas foram construídos levando em consideração o que já se conhece sobre os SADs existentes, e começam a direcionar novos rumos para questões de desempenho, confiabilidade e escalabilidade.

6.5.1. O Network File System da Sun - NFS

O **NFS** (*Network File System*) ([Bil94] [Com91a] [Dew93] [LS90] [Mal92b] [Mil91] [Ste91] [SUN89] [Tan92] [UI89b]) da Sun Microsystems é um dos primeiros SADs comercialmente disponíveis, e um dos mais conhecidos. O NFS apresenta uma série de características que se tornaram padrões entre os SADs, e que introduziram grandes inovações.

O NFS é um sistema aberto de fato, seu protocolo está especificado em documentos disponíveis ao público em geral, com detalhes suficientes para que ele possa ser implementado a partir de sua descrição. Uma inovação de projeto no NFS foi a introdução de um sistema virtual de arquivos, chamado de **VFS** (*Virtual File System*), que constitui uma interface abstrata entre o sistema operacional e o sistema de arquivos. A VFS permite múltiplas implementações de sistemas de arquivos locais, de tal forma que eles possam coexistir, e pode ser usada para suportar uma grande variedade de sistemas de arquivos, distribuídos ou locais.

O NFS é definido em termos de RPCs, e é implementado através de implementações de software para RPCs de domínio público: o pacote NFS RPC ([Mal92a] [Mil91]), que inclui uma linguagem para definição de dados externos chamada **XDR** (***eXternal Data Representation***) [Cas93] que é um protocolo de apresentação que suporta a troca de informações estruturadas entre sistemas de computadores em plataformas de hardware e software heterogêneas.

Todas estas características, que atualmente são encontradas em vários SADs, foram inovações de alto impacto quando apresentadas pelo NFS. Este desbravamento do mercado para SADs permitiu que a Sun definisse padrões que seriam adotados por muitos anos.

6.5.1.1. A Implementação do NFS

A interface de mais baixo nível entre as aplicações e o sistema de arquivos consiste de diretivas tais como: abrir arquivo, fechar arquivo, gravar arquivo, ler arquivo, etc. A maioria dos SADs apresentam operações similares a estas. A interface VFS abstrai estas operações do sistema de arquivos em uma lista genérica de operações [Hon94]:

open	para abrir um arquivo e retornar um descritor
close	para fechar um arquivo
rdwr	lê ou grava um arquivo
ioctl	executa uma operação especial e específica em um arquivo
select	espera por atividade em um arquivo
getattr	busca atributos de arquivo
setattr	estabelece atributos para arquivos
access	testa direitos de acesso de um arquivo ou diretório

Entretanto, servidores sem-estado tornam difícil a gerência de memórias *cache*. Pelo fato de serem servidores sem-estado, eles não tomam conhecimento de operações de *cache* que possam estar ocorrendo em clientes. Por exemplo, se um cliente guarda um rquivo em memória *cache*, ele deve verificar seu estado com o servidor antes de cada acesso, de tal forma que se assegure que o arquivo permanece atualizado. Esta verificação com o servidor requer uma RPC, porém executar esta RPC custa tão caro, tanto para o servidor quanto para o cliente, quanto um pedido por uma cópia adicional do arquivo para *cache*. Estes problemas, portanto, tornam os clientes NFS com *cache* uma opção pouco atrativa, devido ao desempenho ruim.

Existem várias outras maneiras que permitem o ganho em desempenho do lado dos clientes NFS. Por exemplo, pode-se assumir otimisticamente que os dados na memória *cache* são válidos por um breve período de tempo. É comum encontrarmos implementações de clientes NFS que dispensam a verificação de mudanças em arquivos, se o arquivo tiver sido verificado pelo menos há até três segundos decorridos. No caso de diretórios, considera-se um período de até 30 segundos. Estes detalhes de implementações podem gerar conseqüências graves se esta consideração otimista for violada, porém, os ganhos de desempenho conseguidos podem ser mais importantes em certos ambientes operacionais. Certamente, esta possibilidade de ocorrerem inconsistências em dados missão-críticos de uma organização (como em um banco comercial, por exemplo) não podem ser permitidas. Adiante, falamos sobre uma nova versão do NFS (Spritely NFS [Hon94]) que tenta evitar precisamente que estes tipos de problemas aconteçam.

O protocolo NFS também não fornece suporte explícito para o compartilhamento de arquivos, e como o uso de *cache*s em clientes no NFS pode causar visões incorretas do conteúdo dos arquivos, a Sun fornece um protocolo separado para o bloqueio de arquivos na rede [Mil91]. Infelizmente, este não é um produto facilmente encontrado em ambientes onde existem plataformas heterogêneas de hardware e software.

Um servidor NFS exporta partes de sua hierarquia de espaço de nomes. Estas hierarquias são montadas em clientes NFS. Por exemplo, vários dos servidores NFS podem exportar partes de seu sistema de arquivos, porém um cliente irá montar estas partes em uma hierarquia que seja de seu interesse. O NFS não fornece montagens remotas diretamente no servidor de arquivos; toda a manipulação do espaço de nomes é feita no lado dos clientes. Como resultado, é possível que clientes tenham acesso a arquivos idênticos, porém usando nomes completamente diferentes para estes mesmos arquivos. Este fato inibe a transparência de localização. Na figura 6.16, mostramos um exemplo do que acabamos de afirmar. Nesta figura, um servidor NFS exporta dois sistemas de arquivos, chamados de */usuário* e */fonte*. O cliente NFS **A** monta o sistema de arquivos exportado com os nomes correspondentes */usuário* e */fonte*. Por sua vez, o cliente NFS **B** monta o sistema de arquivos em /usr e /fnt, respectivamente. Assim, apesar dos dois clientes terem acesso ao mesmo sistema de arquivos, eles utilizam diferentes nomes de caminhos para os arquivos.

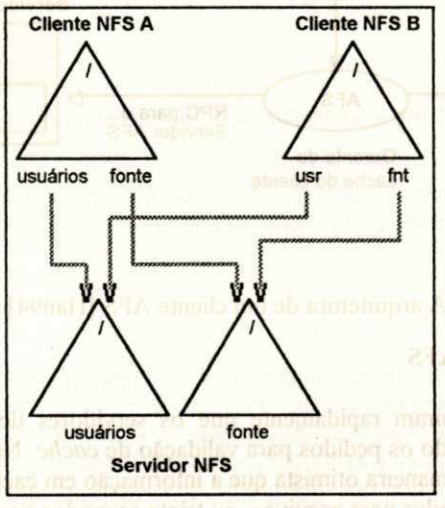


Figura 6.16. A montagem de hierarquias em clientes NFS [Hon94]

Não existe suporte de replicação para servidores no protocolo NFS. Uma aplicação chamada de ***automounter*** ([Cor91] [Blo92] [Ste91]) pode ser usada para associar um ponto de montagem a um conjunto de servidores NFS, e não a um único servidor. Como não existem mecanismos para garantir que estes servidores irão manter imagens idênticas para um mesmo arquivo que seja freqüentemente atualizado, o *automounter* é pouco usado para esta tarefa. O automounter é mais comumente usado para replicar arquivos de somente-leitura, como por exemplo os programas binários.

O protocolo NFS não especifica qualquer tipo particular de controle de acesso. No entanto, a Sun fornece vários métodos bastante úteis. Geralmente, em instalações com NFS, opta-se pelo uso de autenticação no estilo do Unix [Kay94a], onde cada cliente fornece um identificador numérico para o usuário que emite um pedido de acesso ao sistema de arquivos. Como conseqüência, é importante que exista um acordo entre clientes e servidores com relação ao mapeamento de nomes de usuários e seus identificadores numéricos. No capítulo 10 adiante, que trata especificamente de segurança em ambientes da computação distribuída, falamos com maiores detalhes de questões relacionadas com a autenticação de usuários em ambientes distribuídos.

6.5.2. O Andrew File System da Transarc Corporation - AFS

Na mesma época que a Sun começou a trabalhar no NFS, o *Information Technology Center* (**ITC**) da Universidade Carnegie Mellon foi criado para desenvolver um ambiente distribuído acadêmico que foi denominado de **Andrew** ([Tan92] [TVR85] [Cas93] [Dew93]), em homenagem a um dos fundadores da Universidade, o Sr. Andrew Carnegie. O sistema de arquivos produzido por este projeto, chamado de Vice, foi renomeado mais tarde para **AFS** (*Andrew File System*) ([LS90] [Hon94] [Leo94] [Mor86]). Com o passar dos anos, muitos dos cientistas do ITC abriram uma empresa chamada de Transarc Corporation, que é responsável atualmente pela distribuição comercial do AFS.

Em termos de arquitetura, o AFS possui muitos pontos em comum com o NFS: os clientes usam a interface VFS para acessar os arquivos AFS; o AFS é também baseado no modelo de RPCs; e, o AFS utiliza o mesmo formato para a representação externa de dados usada pelo NFS. As principais diferenças ficam pelo fato do AFS ter introduzido no mercado suporte de alto nível para *caching* em estações clientes, uma das fragilidades do NFS. As *cache*s em cliente AFS são enormes, geralmente ocupando vários megabytes de área em disco. O cliente AFS é um sistema de arquivos VFS, porém, como o AFS utiliza o sistema local para *caching*, este sistema de arquivos local será usado após a execução da RPCs. A figura 6.17 ilustra os conceitos. Nesta figura, os pedidos AFS chegam pela interface VFS e vão para o gerenciador de *cache*s AFS no cliente, o qual tenta processar o pedido com os dados na *cache* local, se for possível. Caso necessário, o gerenciador de *cache*s emite primeiramente uma RPC para o servidor AFS, e guarda em *cache* o resultado desta RPC.

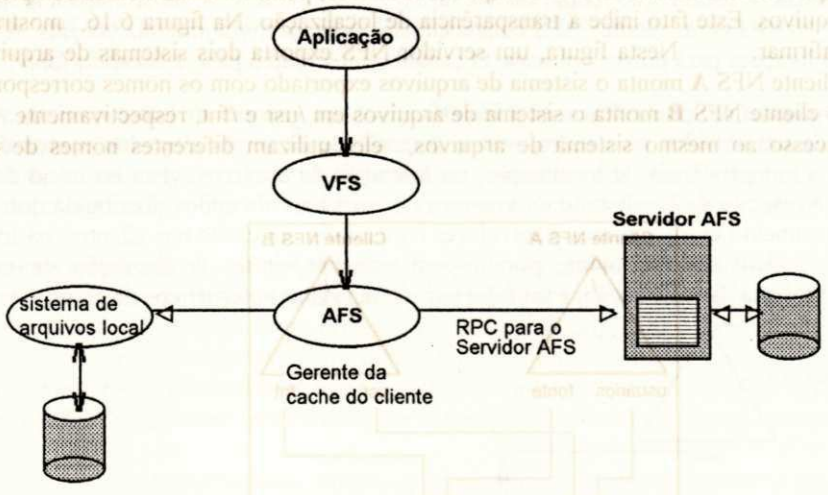


Figura 6.17. A arquitetura de um cliente AFS [Han94]

6.5.2.1. A Consistência de *Cache* no AFS

Os projetistas do ITC concluíram rapidamente que os servidores de arquivos no antigo sistema Vice gastavam um tempo significativo servindo os pedidos para validação de *cache*. No NFS, tenta-se evitar estes pedidos de validação de cache assumindo-se de maneira otimista que a informação em cache permanecerá válida por um certo período após a sua validação (três segundos para arquivos, ou trinta segundos para diretórios). Os projetistas do AFS queriam fornecer maiores garantias para a integridade dos dados.

A solução a que chegaram chama-se "*promessa de retorno*" ("*callback promise*") [Hon94]. Quando um cliente AFS vai buscar informações no servidor de arquivos, esta informação é fornecida juntamente com uma promessa de que o servidor voltará a contactar o cliente se a informação for alterada. Quando um cliente atualiza um arquivo no servidor de arquivos, o servidor cuida de informar a todos os clientes que mantêm caches para este arquivo de que suas cópias não estão mais valendo. Desta forma, o servidor retira as promessas de retorno quando a informação em cache é modificada.

Antes que um cliente AFS possa utilizar um arquivo em cache, ele verifica se existe alguma promessa de retorno para o arquivo. Se existir, o cliente tem a certeza de que os dados do arquivo estão atuais, de tal forma que estes dados podem ser usados, sem a necessidade de comunicação com o servidor. Se a promessa de retorno estiver corrompida, ou se tiver expirado, o cliente requisita ao servidor o número de versão do arquivo em cache. Se for revelado que a versão esta desatualizada, o cliente tem que carregar novamente o arquivo em cache a partir do servidor.

Obviamente, um servidor AFS jamais poderia ser do tipo sem-estado ("stateless") [Dew93]: ele cuida do conteúdo da cache de clientes de forma a manter as promessas de retorno. Para evitar que ele fique sobrecarregado pelo volume de informações, um servidor AFS acopla uma data de expiração a cada promessa de retorno. Adicionalmente, um servidor AFS estará livre para retirar as promessas de retorno a qualquer momento, para evitar que o "overhead" para manutenção de uma grande quantidade de informações sobre o estado dos arquivos em caches de clientes venha a exaurir os recursos do servidor.

A recuperação de falhas [Cri91] é complicada pelos mecanismos de invalidação de cache. As promessas de retorno são armazenadas em memória volátil, de tal forma que quando um servidor AFS é reinicializado, ele não tem mais registro das promessas de retorno emitidas antes de sua queda. Este fato introduz a possibilidade de ocorrerem inconsistências nos dados. As promessas de retorno emitidas antes de uma queda do servidor são perdidas e, portanto, não podem ser retiradas. Se um cliente atualizar um arquivo após a reinicialização de um servidor, qualquer cliente que estiver com uma promessa de retorno irá usar a versão antiga e desatualizada do arquivo. Para evitar que este tipo de coisa aconteça, os clientes AFS contactam regularmente os servidores para verificar se eles não foram reinicializados. Se um cliente descobre que um servidor foi reinicializado, ou se o servidor não responder, ele descarta quaisquer promessas de retorno fornecidas pelo servidor.

6.5.2.2. Nomes no AFS

O espaço de nomes do AFS é construído a partir de volumes [Ui89b] (o mesmo que partições em Unix). Um volume é uma hierarquia do sistema de arquivos que pode ser montada em uma hierarquia AFS já existente. Apesar da inicialização do procedimento de montagem no cliente, as montagens ficam armazenadas no próprio servidor, e visíveis para todos os usuários. Assim, o AFS fornece uma hierarquia de nomes comuns para todos os clientes, sem corromper a transparência de localização, como ocorre no NFS. Na figura 6.18, mostramos a montagem de volumes em um servidor AFS. Nesta figura, clientes AFS convencionais montam o volume raiz em /afs. A seguir, montagens remotas são feitas por ligações para outros volumes. O caminho mostrado é para */afs/celula/fonte*.

Como as montagens remotas referem-se a identificadores de volumes de baixo nível, os volumes podem ser movidos fisicamente de um servidor para outro, sem a necessidade de serem alterados pontos de montagem. Esta facilidade caracteriza a independência de localização, ou migração de arquivos, vista no início do capítulo. O AFS mantém informações a respeito da localização de volumes em uma base de dados distribuída que pode ser acessada pelos clientes. O mapeamento entre volumes e servidores é guardado em cache nos clientes, na forma de uma ***pista***. Os volumes também podem ser replicados, porém para somente-leitura. A abstração de volumes atende às necessidades de transparência de localização, e também permitem ajustes automáticos de carga.

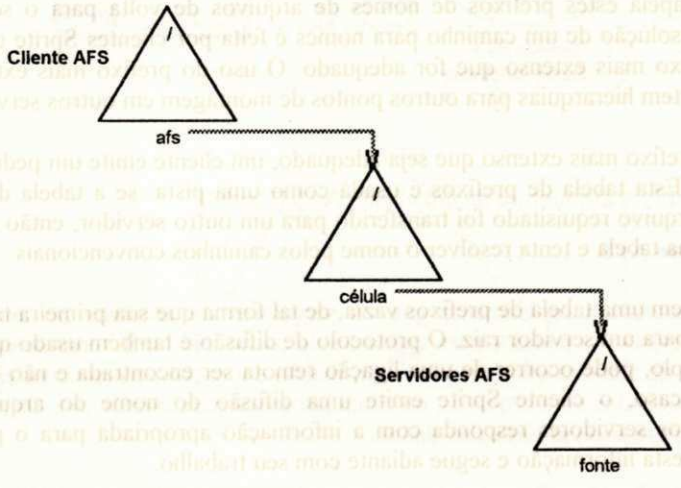


Figura 6.18. A montagem de volumes AFS [Han94]

O controle de acessos no AFS é garantido por listas de controle de acessos [Tan92] acopladas a cada diretório. Os tipos de acessos controlados no AFS são: leitura, gravação e bloqueio. Os próprios usuários podem alterar suas listas de controle de acesso (se forem autorizados para tal).

6.5.3. O Distributed File System da OSF

O ***Distributed File System*** - **DFS** ([OSF90b] [OSF92c]), que é o sistema de arquivos distribuído da infra-estrutura do DCE da OSF, foi desenvolvido a partir do AFS. Portanto, os dois são muito parecidos. A principal diferença reside na implementação de uma semântica Unix mais precisa por parte do DFS. Por exemplo, enquanto o AFS utiliza uma semântica do tipo abre/fecha (sessão) para atualização de arquivos, o DFS emprega níveis bem menores de granularidade para suas atualizações.

O DFS suporta invalidação de cache em cliente pelo uso de um mecanismo de fichas ("tokens") de acesso [Hon94], ao invés das promessas de retorno do AFS. Um determinado tipo de ficha fornece acesso para que clientes modifiquem dados em uma parte específica de um arquivo DFS. Com uma destas fichas em mãos, o cliente tem garantido o acesso exclusivo a esta parte do arquivo.

As fichas de acesso permitem gravações concorrentes sem causar "overhead", caso as partes do arquivo sendo gravadas por diferentes usuários sejam diferentes. Se um cliente busca uma ficha de acesso para gravação, que coincida com outra já em uso, o servidor cuida de liberar a ficha inicial após seu cliente tê-la usado, executa as atualizações necessárias, e reemite a ficha para o novo cliente. Outras fichas DFS atuam de maneira similar para fornecer semântica Unix a um custo bem baixo.

Em todos os outros aspectos, o DFS fornece os mesmos tipos de serviços que o AFS: caching em clientes, espaço de nomes uniforme, replicação de somente-leitura, etc. A principal diferença visível para os usuários com relação ao AFS (que fornece aos seus clientes uma série de serviços distribuídos a nível do sistema operacional local) é que o DFS é ele mesmo um cliente do ambiente de computação distribuída. Como resultado, o DFS é muito mais integrado com a base de serviços distribuídos [Cha92].

6.5.4. Sprite

O ***Sprite*** ([Wit91] [Hon94] [LS90] [Tan92] [TVR85]) é um sistema operacional desenvolvido na Universidade da California no início dos anos 80. Sua arquitetura prevê o uso de estações de trabalho sem disco, e com alta capacidade de armazenamento em memória real. O sistemas de arquivos Sprite merece destaque neste capítulo por ser inovador em duas áreas: nomes e caching.

O espaço de nomes em Sprite, como todos os outros vistos até agora, constitui-se de uma hierarquia de diretórios, que apresenta um mecanismo para a montagem de espaços de nomes usado para construir grandes hierarquias a partir de outras pequenas. Esta montagem é implementada por ligações remotas implícita no sistema de arquivos em servidores Sprite. Uma ligação remota aponta para um arquivo ou diretório em outro servidor Sprite.

O Sprite implementa um novo enfoque para a resolução de caminhos para nomes. Um servidor de arquivos Sprite associa uma hierarquia exportada a um prefixo de caminho para nomes e divulga esta associação. Cada cliente Sprite mantém uma tabela que mapeia estes prefixos de nomes de arquivos de volta para o servidor que fez a divulgação do recurso. Assim, a resolução de um caminho para nomes é feita por clientes Sprite que pesquisam em suas tabelas para encontrar o prefixo mais extenso que for adequado. O uso do prefixo mais extenso permite que servidores de arquivos Sprite exportem hierarquias para outros pontos de montagem em outros servidores.

Depois de encontrado o prefixo mais extenso que seja adequado, um cliente emite um pedido para o sistema de arquivos no servidor indicado. Esta tabela de prefixos é usada como uma pista: se a tabela de prefixos estiver incorreta, por exemplo, porque o arquivo requisitado foi transferido para um outro servidor, então o cliente cuida de descartar este prefixo inválido de sua tabela e tenta resolver o nome pelos caminhos convencionais.

Inicialmente, cada cliente tem uma tabela de prefixos vazia, de tal forma que sua primeira tarefa é emitir uma operação de difusão ("broadcast") para um servidor raiz. O protocolo de difusão é também usado quando a tabela de prefixos está incompleta. Por exemplo, pode ocorrer de uma ligação remota ser encontrada e não existir um prefixo adequado na tabela local. Neste caso, o cliente Sprite emite uma difusão do nome do arquivo que ele esta procurando, e espera que algum dos servidores responda com a informação apropriada para o prefixo. O cliente atualiza sua tabela de prefixos com esta informação e segue adiante com seu trabalho.

A tabela de prefixos age como uma cache para a pesquisa de caminhos para nomes, o que reduz sensivelmente o tráfego de pesquisa que seria normalmente necessário. Porém, por usar uma operação de difusão, o Sprite não é escalável além de um segmento isolado de rede local [Hon94].

O suporte a caches em clientes no Sprite é comparável ao do AFS: o servidor mantém informações sobre clientes que tenham cópias em cache de um dado arquivo, e invalida estas caches quando ocorrem modificações nas cópias. Se existir exatamente um cliente com um arquivo aberto, o cliente mantém as atualizações na sua própria cache. Se um servidor cuida dos compartilhamentos de gravação entre vários clientes, ele instrui a estes clientes para que eles não aloquem cópias de arquivos em suas caches locais. Com a desabilitação do mecanismo de caching nos clientes, todas as leituras e gravações tornam-se RPCs dos clientes para os servidores. Neste caso, problemas de desempenho começam a ocorrer, devido à semântica Unix que é imposta.

6.5.5. Spritely NFS

O ***Spritely NFS*** [Hon94] é um sistema de arquivos distribuído que foi desenvolvido em um dos laboratórios da DEC (Digital Equipment Corporation). Ele combina as melhores características do Sprite e do NFS, como sugere o seu nome. Do NFS são aproveitados o protocolo completo e os serviços de suporte que o acompanham, o que destaca uma enorme base instalada de sistemas compatíveis. Do Sprite, são aproveitadas as políticas de consistência de cache e seus respectivos algoritmos. Com a definição explícita de mecanismos para a consistência de caches, é fornecida semântica Unix, com um desempenho superior para os servidores de arquivos. No Spritely NFS, os servidores são do tipo de estado-cheio ("statefull"), o que complica as questões de recuperações em casos de queda.

São incluídas três RPCs adicionais no Spritely NFS com relação ao NFS puro: abrir arquivo, fechar arquivo, e chamada de retorno ("callback"). As RPCs para abertura e fechamento de arquivos permitem que o servidor administre o uso de arquivos por clientes, e estas RPCs são coordenadas pelas RPCs de chamada de retorno. A RPC de chamada de retorno não existe em ambientes NFS puros, como já afirmamos, pois ela é uma RPC enviada do servidor para os clientes usada para retirar o direito de cache para um determinado arquivo do lado do cliente, e serve também para exigir que arquivos alterados em caches de clientes sejam enviados de volta para o servidor.

Apesar de sua característica de estado-cheio [Dew93], o Spritely NFS não atende de maneira adequada as questões de recuperação de falhas, apesar destes problemas estarem sendo tratados em protótipos na DEC. Estes protótipos prevêem um mecanismo de recuperação de falhas onde o trabalho maior fica por conta dos clientes. Os servidores precisam de memória secundaria estável, de tal forma que se possa administrar os clientes que tenham cópias de um arquivo mestre. Após a reinicialização de um servidor, os clientes listados em sua memória estável são notificados que a recuperação do servidor esta se iniciando. Os clientes podem, então, informar ao servidor quais arquivos eles mantêm abertos, de forma que se possa reconstruir as informações sobre o estado destes arquivos nos servidores.

Existem algumas esquisitices no protocolo de recuperação, como clientes que não respondem ao pedido do servidor para iniciar a recuperação. A recuperação da queda de clientes é simples: quando um cliente é reinicializado, seus servidores são notificados durante o procedimento inicial de montagem. Os servidores utilizam esta informação para limpar quaisquer estados que estejam registrados a respeito deste cliente.

6.5.6. CODA

O ***sistema de arquivos distribuído CODA*** [Hon94], em desenvolvimento na Universidade Carnegie-Mellon, tem muito em comum com o AFS, pois também foi baseado no sistema Vice. Porém, o CODA adiciona duas características importantes ao AFS: total replicação de arquivos por vários servidores, e suporte a clientes desconectados.

A replicação de arquivos em diversos servidores é implementada com servidores que administram o histórico de cada um de seus arquivos. Servidores que possuem arquivos replicados, trocam informações sobre versões entre si, e mantêm estas informações em um vetor de números de versões. Nas situações normais, todos os servidores em comum para um determinado arquivo apresentam o mesmo conteúdo e o mesmo número de versão nos seus vetores. Quando um cliente atualiza um arquivo, ele envia estas atualizações para cada um dos servidores que mantêm este arquivo. Os servidores retornam por sua vez seus vetores de versões para o cliente, que os compara em busca de informações desatualizadas ou inconsistentes. Se um dos servidores tiver perdido uma das atualizações, o cliente cuida de enviar informações para que este servidor possa reparar a sua inconsistência. No caso de queda de um servidor, o mesmo irá descobrir que após sua reinicialização, ele pode ter perdido algumas atualizações, e pode então providenciar sua recuperação graciosamente. A dificuldade que pode surgir é o caso de servidores perderem contato uns com os outros por causa de problemas na rede. O CODA permite que sejam feitas atualizações concorrentes nas sub-redes envolvidas, onde não tenham ocorrido problemas. Após o reparo da rede, os servidores que ficaram indisponíveis podem constatar que suas cópias de um arquivo replicado estão inconsistentes. Nesta situação muito particular, o arquivo fica inacessível, e exige-se reparo manual da inconsistência.

Capítulo 7: Bancos de Dados Para a Arquitetura Cliente/Servidor

7.1. Introdução

A grande divulgação no uso das tecnologias de bancos de dados ocorreu durante os anos 70, e foi crucial para o desenvolvimento de *Sistemas de Gerenciamento de Bancos de Dados* (**SGBDs**) ([Dat81] [Set86] [Ull88] [Wie86]) integrados e poderosos. Os SGBDs permitiram que fossem alterados os antigos modelos de processamento, onde as aplicações individualmente definiam e administravam seus próprios dados, como mostrado na figura 7.1.

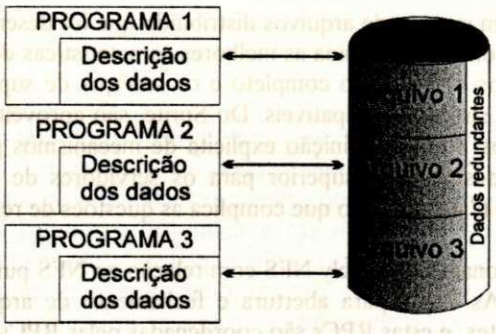


Figura 7.1. O processamento de dados tradicional [Wie86]

No novo modelo, os dados passaram a ser definidos e administrados centralizadamente, como mostrado na figura 7.2. Esta nova orientação resultou no princípio da *independência de dados* [Dat81], onde os programas de aplicações passaram a ser imunes às mudanças nas organizações físicas ou lógicas dos dados, e vice-versa.

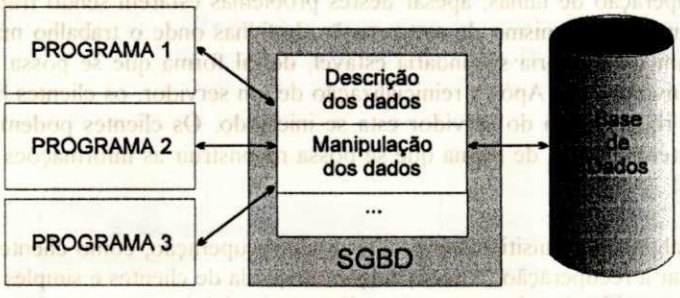


Figura 7.2. O processamento no estilo dos SGBDs [Wie86]

Uma das principais motivações por detrás do uso dos SGBDs foi o desejo de integrar os dados operacionais de toda uma empresa com o objetivo de fornecer um controle centralizado destes dados [Sin94]. Portanto, deve-se observar que o mais importante por detrás das tecnologias de bancos de dados é a integração dos dados, e não particularmente a sua centralização. É importante observarmos que é possível que se consiga um alto grau de integração de informações corporativas sem necessariamente existir a necessidade de que estas informações estejam centralizadas. É neste momento que destacamos a importância das redes de computadores, que combinadas com as tecnologias de SGBDs geraram uma série de novos conceitos sobre distribuição e controle de dados em ambientes da computação distribuída [Ric94].

A tecnologia para os sistemas de bancos de dados estabeleceu-se, e permitiu que se construísse um grande número de aplicações corporativas, muitas vezes cruciais para o negócio das empresas [Bur93c]. Atualmente, as redes de computadores têm-se tornado mais e mais sofisticadas e ubíquas, de modo que a integração das mais recentes tecnologias de bancos de dados com as redes tem resultado em avanços significativos na Tecnologia da Informação, particularmente com o aparecimento dos ***bancos de dados distribuídos*** ([OV91] [CP84] [SUN92e]), e mais recentemente, dos ***bancos de dados para a arquitetura cliente/servidor*** ([Sal93] [Cas93] [Bau94a]).

Problemas completamente novos surgiram na construção e implementação de bancos de dados distribuídos (incluindo os bancos de dados para a arquitetura cliente/servidor), e muita pesquisa foi necessária para solucioná-los. Os trabalhos de pesquisa realizados constituíram uma nova disciplina, com seus próprios princípios. Assim, pretendemos abordar neste capítulo os aspectos da nova tecnologia que consideramos os mais importantes para os bancos de dados na arquitetura cliente/servidor, e falamos um pouco, também, sobre os bancos de dados distribuídos.

7.2. As Opções para SGBDs

Podemos, de maneira mais geral, considerar os sistemas de gerenciamento de bancos de dados atuais em quatro categorias [Sal93] (levando-se em consideração as plataformas que os suportam). Para o nosso estudo não consideramos aqui opções adicionais como os *SGBDs Orientados a Objetos* [Ste94], ou os *SGBDs Baseados em Conhecimento* [Ull88]. As quatro categorias são:

- SGBDs Centralizados
- SGBDs para Redes Locais (ou Servidores de Arquivos)
- SGBDs Cliente/Servidor
- SGBDs Distribuídos

7.2.2. SGBDs Centralizados

Nesta categoria, a mais tradicional e mais comumente encontrada em plataformas baseadas em mainframes ou em minicomputadores, um computador "hospedeiro" roda todos os aplicativos, incluindo o SGBD, permitindo que as aplicações façam acesso às bases de dados e às facilidades de comunicações que cuidam da troca de mensagens entre terminais multi-usuários e o software no computador principal. Geralmente estes terminais são do tipo "mudos", possuindo muito pouco, ou mesmo nenhum poder de processamento autônomo. Atualmente é comum que microcomputadores, isoladosou em rede, possam rodar software de emulação de terminais para acessar os bancos de dados no mainframe, conseguindo inclusive recuperar informações para os seus discos locais, ou então enviar dados para a base de dados no mainframe (operações de "download"e "upload", conhecidas respectivamente como carga descendente e carga ascendente) [Dew93]. Existe ainda a possibilidade de que microcomputadores comuniquem-se com estes SGBDs centralizados por meio de "gateways" especializados para a ligação micro-mainframe [Hop93a].

Todo o processamento de dados de um sistema centralizado ocorre no computador hospedeiro, e o SGBD deve estar rodando ininterruptamente para atender aos pedidos de aplicações de usuários. Esquematicamente, podemos considerar que as aplicações de usuários comunicam-se com o SGBD através dos terminais, e que o SGBD comunica-se com os recursos locais (as bases de dados), e com as aplicações. Na figura 7.3, apresentamos um desenho esquemático do processamento de um SGBD em um ambiente centralizado.

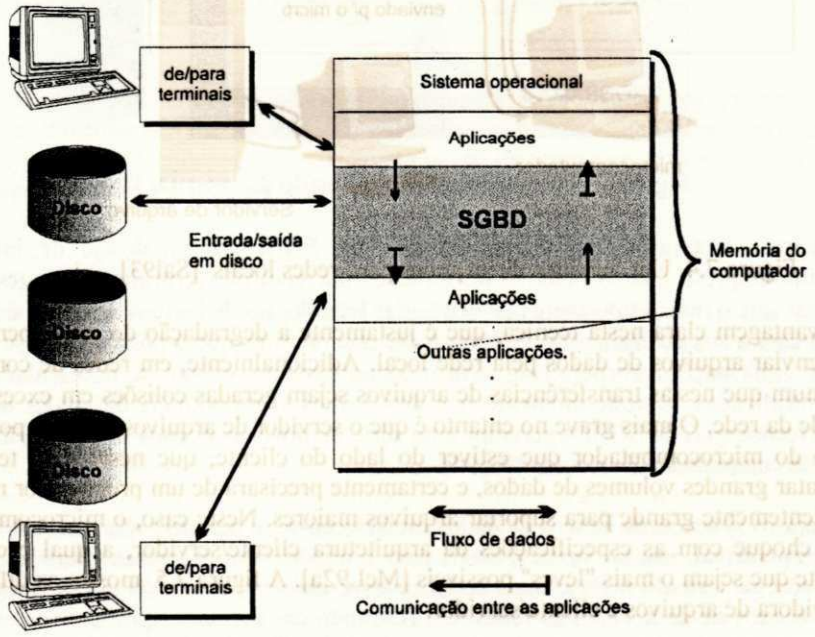


Figura 7.3. O Esquema de Processamento de um SGBD Centralizado [Sal93]

A maioria das plataformas centralizadas, com mainframes, implementam SGBDs Relacionais ([VN90a] [Dat81]), como o DB2 da IBM [Dat81], ou o ORACLE [Dew94] da Oracle Co., dentre outros. As principais vantagens para um sistema deste tipo são o alto nível de segurança oferecido e a capacidade para manipular enormes volumes de informações. Outra vantagem é o suporte a centenas de usuários simultaneamente: é comum encontrarmos ambientes deste tipo com SGBDs suportando mil ou mais usuários simultaneamente (por exemplo, um SGBD que suporta uma aplicação para reservas de passagens aéreas) [Cla92b].

Podemos citar como desvantagem o fato de SGBDs centralizados rodarem em plataformas que custam muito dinheiro (geralmente, alguns milhões de dólares, dependendo do porte), onde os custos de manutenção são bastante elevados e exigem facilidades extras como a necessidade de ferramentas de suporte sofisticadas, infra-estrutura específica, com sistemas de refrigeração e sistemas de controle climático, e uma equipe altamente preparada de operadores e analistas de suporte que possam manter o sistema ativo e no ar 24 horas por dia e por semanas a fio [Cas93].

7.2.3. Os SGBDs para Redes Locais (ou Servidores de Arquivos)

Em uma rede local, o servidor principal é o responsável pelos serviços de arquivos fornecidos para as estações clientes, graças às características implementadas pelos próprios sistemas operacionais para redes locais (S.O.R.s), como por exemplo, o NetWare da Novell, ou o Windows NT/AS da Microsoft [Red92a].

O servidor de arquivos busca em seus discos os dados requisitados pelos usuários, e envia estes dados pelos meios de acesso da rede em particular (por exemplo, cabo coaxial, fibra ótica, etc). Não ocorre nenhum tipo de processamento do lado do servidor. Quando o cliente recebe os dados requisitados, ele é o responsável pelo tratamento das informações relevantes, o que geralmente é efetuado em todo um arquivo. Assim, o cliente altera os dados que lhe interessam e devolve o arquivo alterado para o servidor, como mostrado na figura 7.4 .

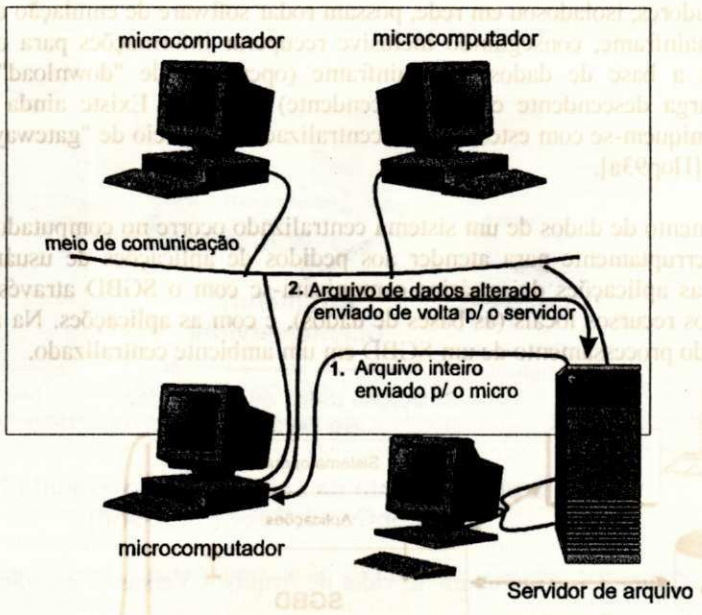


Figura 7.4. Um Servidor de arquivos para redes locais [Sal93]

Existe uma desvantagem clara nesta técnica, que é justamente a degradação do desempenho do servidor principal ao receber ou enviar arquivos de dados pela rede local. Adicionalmente, em redes de compartilhamento de acesso ao meio, é comum que nestas transferências de arquivos sejam geradas colisões em excesso, o que pode comprometer a velocidade da rede. O mais grave no entanto é que o servidor de arquivos terá seu poder limitado ao poder de processamento do microcomputador que estiver do lado do cliente, que neste caso tem que ser um equipamento capaz de tratar grandes volumes de dados, e certamente precisará de um processador mais robusto ou de um disco rígido suficientemente grande para suportar arquivos maiores. Neste caso, o microcomputador cliente entrará diretamente em choque com as especificações da arquitetura cliente/servidor, a qual prevê estações de trabalho do lado do cliente que sejam o mais "leves" possiveis [McL92a]. A figura 7.5 mostra as diferenças básicas entre as arquiteturas servidora de arquivos e cliente/servidor.

No exemplo mostrado na figura 7.5, digamos que a companhia telefônica do Ceará possua uma base de dados com todos os assinantes do estado. Se desejássemos acessar somente os assinantes de Fortaleza, a partir de uma estação cliente, o comportamento seria diferente para as duas arquiteturas. Na arquitetura servidora de arquivos, toda a base de dados de assinantes seria enviada para a estação cliente, enquanto que na arquitetura cliente/servidor, somente os dados relativos aos assinantes em Fortaleza seriam passados para a estação cliente.

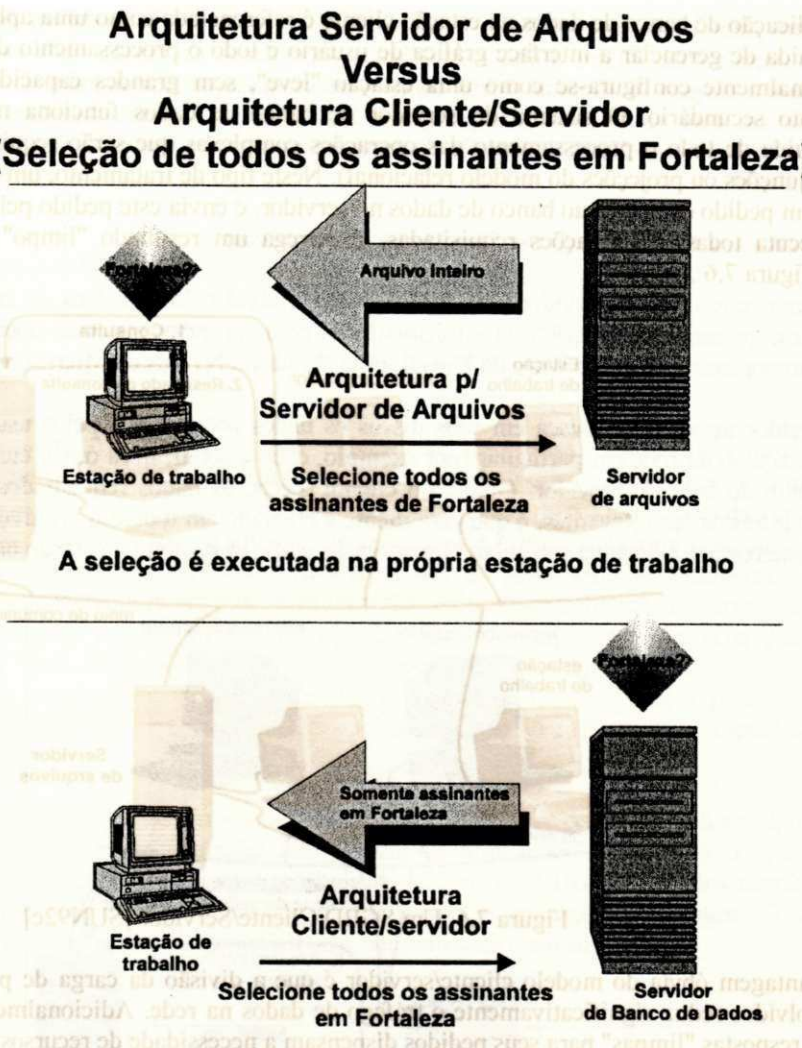


Figura 7.5. Arquitetura Servidor de Arquivos Versus Cliente/Servidor [And92]

É comum ouvirmos falar que servidores de arquivos em redes locais sejam também chamados de bancos de dados semi-relacionais [Sal93]. Neste modelo a maioria das características de sistemas relacionais autênticos não estão implementadas, como por exemplo, os princípios de integridade referencial, segurança, e outros. Podemos citar como representantes dos sistemas de arquivos em redes locais produtos como o R:Base da Microrim, o dBASE IV da Borland (adquirida pela Novell), o FoxPro da Microsoft, Paradox da Borland, todos estes fornecendo serviços para redes menores. Para redes um pouco maiores, onde existe a necessidade explícita de outras funcionalidades, existe ainda uma categoria de produtos mais robustos, porém que não implementam o modelo cliente/servidor, e podemos citar como exemplos o Dataflex da empresa Data Access, e o db_Vista III da empresa Raima Co. Todos estes produtos citados acima são descritos com detalhes em [Gar93].

7.2.4. SGBDs para a Arquitetura Cliente/Servidor

Um SGBD que implemente o modelo cliente/servidor divide a carga de processamento entre estações clientes e servidoras [Gra91]. Na estação cliente rodam as aplicações de usuários, incluindo a parte do SGBD relativa ao cliente, e nos servidores rodam as partes servidoras do SGBD. Normalmente, em uma opção como esta, se tivermos uma rede local, pode existir ao mesmo tempo a opção do servidor de arquivos, que irá fornecer espaço em disco para aplicações gerais, e recursos compartilhados, como por exemplo impressoras. Porém, o ideal é permitir que uma estação - normalmente uma estação mais poderosa, com processador(es) RISC e grande capacidade de armazenamento secundário - seja dedica para o servidor de bancos de dados [Mil94a].

A aplicação de banco de dados na estação cliente é referenciada como uma aplicação frontal ("front-end") [And92], e cuida de gerenciar a interface gráfica de usuário e todo o processamento de entrada/saída requisitado por ele. Normalmente configura-se como uma estação "leve", sem grandes capacidades de processamento ou armazenamento secundário. O sistema do servidor de banco de dados funciona na retaguarda ("back-end") [And92], e cuida de todo o processamento das operações complexas que serão requisitadas (como por exemplo operações de junções ou projeções do modelo relacional). Neste tipo de tratamento, um usuário via aplicação front-end executa um pedido de acesso ao banco de dados no servidor, e envia este pedido pela rede. O servidor de banco de dados executa todas as operações requisitadas, e entrega um resultado "limpo" para o usuário, conforme mostrado na figura 7.6 .

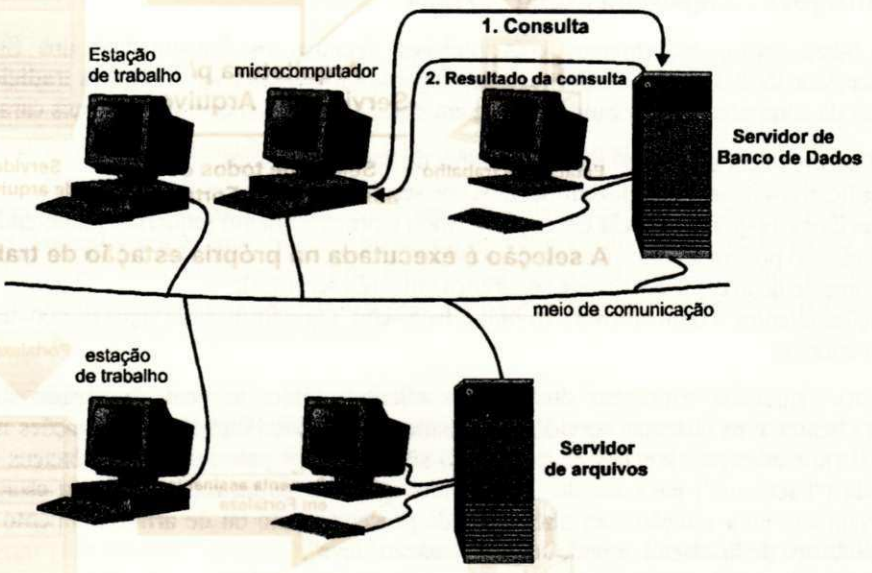


Figura 7.6. Um SGBD Cliente/Servidor [SUN92e]

A vantagem óbvia do modelo cliente/servidor é que a divisão da carga de processamento entre os dois sistemas envolvidos reduz significativamente o tráfego de dados na rede. Adicionalmente, as estações clientes ao receberem as respostas "limpas" para seus pedidos dispensam a necessidade de recursos locais sofisticados.

O objetivo deste capítulo é justamente enfatizar os SGBDs cliente/servidor, que começam a proliferar no mercado, e são hoje uma das soluções mais elegantes de processamento encontradas por empresas interessadas no uso das tecnologias disponibilizadas pela computação distribuída [Bur93c]. Sistemas "front-end" e "back-end" estão sendo projetados, implementados e lançados no mercado com um número cada vez mais impressionante de funcionalidades, e utilizam de maneira ótima o modelo cliente/servidor. Exemplos destes SGBDs são muitos, e citamos como um exemplo inicial os mais importantes, como o ORACLE, da Oracle Co., o Ingres da Ingres Co., o Informix, da Informix Software Inc., dentre muitos outros descritos com detalhes adiante, e referenciados em [Men94c].

7.2.5. Os SGBDs Distribuídos

O *SGBDs Distribuídos* (**SGBDDs**) ([OV91] [CP84]) vêm tornando-se mais importantes a cada dia para solucionar problemas de integração eficiente de dados corporativos distribuídos por uma grande área geograficamente dispersa. Eles estendem as limitações dos SGBDs centralizados e adaptam-se mais naturalmente às estruturas descentralizadas de muitas empresas. Podemos considerar de uma maneira simples um *Banco de Dados Distribuído* (**BDD**) como sendo um conjunto de dados que pertencem logicamente a um mesmo sistema, porém que se encontram dispersos pelos vários sítios de uma rede de computadores. Dois aspectos merecem destaque:

- *a Distribuição em si:* o fato de os dados não estarem todos concentrados em um único sítio, e dependentes de um processador único;
- *a Correlação Lógica*: o fato de os dados possuírem propriedades que os ligam uns aos outros.

Um BDD [CP84] corresponde a um conjunto de dados distribuído por diferentes computadores em uma rede. Cada sítio da rede possui capacidade autônoma de processamento e pode executar uma aplicação localmente. Cada sítio pode igualmente participar de aplicações globais, que exigem o acesso a dados de outros sítios, por meio de um subsistema de comunicações.

Um SGBDD por sua vez suporta a criação e a manutenção de bancos de dados distribuídos. Os serviços suportados por estes sistemas incluem: acesso remoto às bases de dados por programas de aplicações; transparência de distribuição dos dados; suporte à administração e controle distribuídos; suporte para mecanismos de controle de concorrência e recuperação em ambientes distribuídos.

7.3. SGBDs para a Arquitetura Cliente/Servidor

Nesta seção, examinamos com maiores detalhes os fundamentos dos SGBDs para a arquitetura cliente/servidor [Sal93]. Estes SGBDs seguem o paradigma cliente/servidor mais tradicional, largamente usado em ambientes da computação distribuída de hoje em dia. São sistemas com as seguintes características [Ull93]:

- os dados residem em um ou mais servidores de bancos de dados;
- os aplicativos de usuários fazem pedidos aos servidores de bancos de dados;
- o servidor é responsável pela execução de todo o processamento requerido (back-end);
- os dados só podem ser alcançados via servidor de bancos de dados (de tal forma que se asseguram mecanismos de controle de acesso, conflitos e segurança em cada servidor);
- estações clientes rodam software frontal ("front-end") que fornece serviços de consultas, atualizações e emissão de relatórios;

As principais vantagens dos SGBDs cliente/servidor surgem da divisão de processamento entre os sistemas clientes e os sistemas servidores de bancos de dados [Ric94]. As operações mais complexas da álgebra relacional (por exemplo, junções ou projeções) são efetuadas pelo servidor de bancos de dados, que trabalha na retaguarda ("back-end") para atender aos pedidos de usuários na rede. Assim, a estação cliente não precisa ser necessariamente uma estação com alto poder de processamento ou de armazenamento secundário, ela só precisa rodar o software de front-end, e pode ser uma estação "leve".

A distribuição do processamento reduz a carga de tráfego na rede, pois ao invés de enviar um arquivo de dados completo pelo meio de comunicações, como ocorre com os sistemas de arquivos vistos anteriormente, o tráfego reduz-se a consultas vindas dos clientes e respostas "limpas" vindas dos servidores de bancos de dados. É comum que servidores de bancos de dados possam armazenar procedimentos e consultas (chamados de procedimentos armazenados, "stored procedures" [Cla93b]) que serão disparados automaticamente dependendo de um determinado contexto de uso previamente definido, o que reduz ainda mais este tráfego.

Uma grande vantagem na separação de tarefas é a possibilidade de que se forneça uma independência de software muito interessante a nível das estações de trabalho, que podem ser baseadas nas mais diversas plataformas, como por exemplo, microcomputadores tipo PCs, Macintoshes, estações RISC com Unix, ou mesmo uma combinação destas, que podem ter os mais diversos sistemas operacionais instalados, como por exemplo o DOS e Windows da Microsoft, OS/2 da IBM, System 7 da Apple dentre vários outros.

A conseqüência da independência das estações de trabalho é a independência de aplicativos. As estações clientes não precisam obrigatoriamente utilizar o mesmo software de SGBD instalado no servidor de banco de dados [Men94c].

A grande desvantagem dos sistemas cliente/servidor [Hay93d] pode ser considerada do ponto de vista de custos extras com pessoal administrativo e de suporte, necessários para a manutenção do servidor de banco de dados. Os ambientes cliente/servidor também são mais complexos, e compostos por tecnologias recentes, o que justifica a falta de experiência e de pessoal especializado. A questão da independência de software tem seu lado negativo. A possibilidade de coexistirem vários front-ends em um mesmo ambiente pode aumentar consideravelmente as necessidades de suporte à programação, pois os códigos de programas a serem desenvolvidos e mantidos são os mais variados [Dew94].

Hoje em dia, o suporte para a interconectividade entre os diversos SGBDs cliente/servidor ainda deixa a desejar [Hop93a]. A maioria dos sistemas back-end só consegue partilhar dados com sistemas similares e quase todos os front-ends disponíveis suportam apenas poucas opções de combinações dentre os vários sistemas back-end disponíveis. Escolher o uso de um SGBD cliente/servidor pode implicar na escolha dentre poucos fornecedores de front-ends, o que pode limitar severamente a disponibilidade de ferramentas para o desenvolvimento de aplicativos nestes front-ends específicos [LD93b]. Espera-se que estes tipos de problemas diminuam à medida que SGBDs cliente/servidor passem a ser mais difundidos e que fornecedores de front-ends e back-ends forneçam maiores opções de combinações.

7.4. Tecnologias para os SGBDs Cliente/Servidor

Das muitas tecnologias disponibilizadas recentemente para os SGBDs, algumas são da maior importância para os ambientes da computação distribuída. Na maioria dos SGBDs para a arquitetura cliente/servidor disponíveis atualmente, estas características vão estar presentes, porém, muitas vezes a sua implementação irá apresentar mudanças sutís de SGBD para SGBD. Nesta seção, veremos estas novas tecnologias de um ponto de vista mais abrangente, e quando estivermos falando dos SGBDs para arquitetura cliente/servidor individualmente, destacamos as diferenças que venham a existir na implementação das mesmas.

7.4.1. Gatilhos (Triggers)

Um ***gatilho*** ([Sal93] [Men94c] [And92] [Dew94] [Cla93b]) consiste de um conjunto de declarações SQL (ou um conjunto de declarações em outra linguagem) que é invocado (ou "engatilhado") por eventos de transações de bancos de dados como inserções, deleções ou atualizações. Os gatilhos ficam embutidos nas próprias definições do banco de dados, e não podem ser usados isoladamente por usuários ou programadores.

Os gatilhos estão geralmente associados a uma tabela em particular. Sua implementação pode variar de um SGBD para outro. Por exemplo, em alguns SGBDs pode-se criar três gatilhos para cada tabela, um para inserção, um para deleção e um para atualização. Em outros SGBDs, os gatilhos podem ser disparados imediatamente antes, ou imediatamente depois de uma inserção/deleção/atualização. Outros SGBDs são mais limitados, oferecendo somente gatilhos de deleção ou inserção.

Uma característica importante para os gatilhos, e que vem sendo bastante estudada, diz respeito às questões de aninhamento de gatilhos, isto é, a possibilidade de um gatilho disparar outro. O perigo com estes tipos de gatilhos é que podem ocorrer laços sem fim em programas nos servidores, o que pode prejudicar sensivelmente o desempenho dos mesmos, devido às suas características recursivas.

Uma outra inovação com relação aos gatilhos diz respeito a uma maior flexibilidade na definição de eventos. Por exemplo, podem ser definidos eventos baseados em tempo, e eventos de alerta. Um gatilho baseado em tempo pode ser usado para disparar rotinas de operação que devam rodar à noite em um CPD, sem intervenção de operadores. Um evento de alerta pode ser usado para examinar valores em bancos de dados quando certas condições forem atingidas. Por exemplo, um evento de alerta pode ser usado para indicar, em uma aplicação na Bolsa de Valores, se o preço de determinada ação caiu para um preço abaixo do nível aceitável, de tal forma que os corretores comecem a vendê-la imediatamente.

7.4.2. Procedimentos Armazenados

Os ***procedimentos armazenados*** ([Sal93] [Men94c] [And92] [Dew94] [Cla93b]) são declarações SQL que são compiladas, otimizadas, e armazenadas no servidor. Elas podem ser invocadas sem o "overhead" da compilação e otimização que poderiam ter sido requeridas anteriormente. O uso de procedimentos armazenados pode melhorar sensivelmente o desempenho em consultas onde o processo de otimização seja significativo com relação ao tempo total de execução da mesma.

Os procedimentos armazenados são da maior importância para os administradores de bancos de dados (DBAs - Database Administrators), pois consultas dirigidas por parâmetros podem ser otimizadas pelo DBA e gerenciadas como se fossem um objeto do banco de dados. Por exemplo, privilégios de segurança podem ser fornecidos ou revogados para procedimentos individualmente, como se estes procedimentos fossem um objeto como outro qualquer na base de dados.

Os procedimentos armazenados são também da maior importância para os desenvolvedores de aplicações, pois eles facilitam a reutilização de código. Fornecem também maior flexibilidade, pois o controle do fluxo de programas, como laços e desvios, pode ser armazenado na forma de procedimentos armazenados.

Em alguns SGBDs os procedimentos armazenados não são implementados por completo, eles são simplesmente compilados e armazenados, porém não são otimizados. Esta funcionalidade é conhecida como "comandos armazenados".

7.4.3. Gerenciamento de Transações

Os SGBDs devem sempre garantir a integridade de dados. As transações especiais, tipo **"commit"** ou **"rollback"**, garantem que uma unidade inteira de processamento seja completada ou abandonada. Esta unidade inteira de processamento é também conhecida como uma unidade ***atômica*** de processamento [Cla92b]. Nestes casos, a unidade completa-se por inteiro ou nada é realizado na base de dados.

Em casos onde existam transações mais complexas que aquelas do tipo *ou-faz-tudo-ou-não-faz-nada*, a solução ideal é permitir que parte de uma transação seja desfeita ("rolled-back"). Após este "rollback" parcial, pode ser preferível que se continue o processamento, ou que se dê por encerrada a transação. O particionamento de transações, pode ser feito de duas maneiras: com *transações aninhadas*, ou com *pontos de commit* para as transações [OV91].

Nas *transações aninhadas*, cada transação recebe um nome e é aninhada com uma ou mais transações. Quando a transação mais externa é completada ("commited"), suas transações componentes são também completadas. Neste enfoque, pode-se completar ("commit") ou desfazer ("rollback") componentes isolados de uma transação (pelo seu nome) sem afetar os outros componentes da transação.

Nos *pontos de commit*, permite-se que o desenrolar natural da transação seja completado, mesmo que a parte mais recente de uma transação falhe. Quando ocorrem erros, o comando ROLLBACK pode ser emitido usando-se a informação adicional do ponto de commit como um qualificador para definir até onde o rollback deve ser executado.

Uma outra consideração das mais importantes em ambientes de processamento de transações distribuídas por muitos servidores é o suporte ao *protocolo de commit de duas fases*. No protocolo de commit de duas fases (**2PC** - *Two Phase Commit*) [Dew93] adiciona-se outro commit antes do commit final. Um dos servidores age como um coordenador de transações, e cada um dos servidores envolvidos é requisitado para preparar-se para o commit. Quando o coordenador recebe a confirmação de que cada participante está apto para o commit, o commit final é então emitido. Se o sistema cai neste ponto, o coordenador poderá ser consultado por todos os participantes após a reinicialização do sistema, para verificarem se a transação deve ser completada ("commited") realmente ou desfeita (rolled-back).

Todos os detalhes envolvidos com protocolos de commit de duas fases serão vistos pormenorizadamente no capítulo 11, quando falamos dos sistemas OLTP.

7.4.4. O Suporte a Cursores

Cursores ([Sal93] [Men94c] [And92] [Dew94] [Cla93b]) apresentam diferentes significados nos diversos ambientes de SGBDs. Um cursor pode ser referenciado como um apontador para registros de rolagem ("scroll") ou pode também referir-se a uma conexão de banco de dados.

Com relação à sua primeira funcionalidade, um cursor usado como apontador pode suportar a rolagem para trás ou não. Normalmente, a rolagem para frente é suportada em todos os produtos, enquanto que a rolagem para trás ainda é implementada de maneira restrita.

Outra característica importante para os cursores diz respeito aos buffers de busca. Um grande buffer de busca aumenta o desempenho se muitas linhas devem ser trazidas da base de dados. Porém, efeitos colaterais podem ocorrer devido ao tamanho destes buffers, principalmente, com relação às questões de controle de concorrência, devido aos bloqueios necessários para o nível de isolamento desejado.

7.4.5. Controle de Concorrência

Nas conexões de bancos de dados, uma das questões principais diz respeito ao acesso concorrente aos mesmos dados. Os *níveis de isolamento* definem o comportamento que se deve esperar do processamento de duas ou mais transações simultaneamente. O ANSI, quando publicou o padrão **SQL-92**, definiu quatro níveis de isolamento, dos quais falamos a seguir [Cla93b]:

1. *Leitura sem commit*: este tipo de conexão lê dados partilhados com transações que foram processadas por outras conexões, mas que ainda não foram completadas. Se uma destas transações for desfeita ("rolled-back"), processos que tenham lido os valores não completados (não "commited") irão incluir dados não confiáveis nas bases de dados locais. Geralmente este tipo de conexão só é permitida para o modo de somente-leitura;

2. *Leitura com commit*: as inserções, atualizações, ou deleções só ficam disponíveis para outras transações depois de serem completadas (commited);

3. *Leitura repetida*: este nível de isolamento lê dados de transações completadas (commited), porém mantém um bloqueio nas tuplas envolvidas, o que garante que múltiplas leituras das mesmas tuplas nesta transação produzam resultados consistentes; e

4. *Serialização*: mesmo com o nível de isolamento anterior, pode-se criar um conjunto de transações que produzam resultados inconsistentes. Isto ocorre quando o critério de pesquisa produz um conjunto diferente de tuplas devido a mudanças processadas por outra transação. Como estas novas tuplas não estiveram envolvidas na leitura original, não foram mantidos bloqueios, de tal forma que outro processo pôde modificá-las. O nível de isolamento com serialização faz com que o SGBD crie uma ordem serial das transações que produzirá um resultado consistente. Observe que a manutenção de bloqueios a nível de tabelas irá produzir os mesmos resultados, com o custo extra do controle de concorrência.

Os níveis de isolamento determinam a duração e a extensão de bloqueios nas bases de dados, portanto, é importante que se compreenda um outro conceito muito importante neste contexto: o *nível de granularidade* dos bloqueios. Os principais esquemas são: *bloqueio a nível de linhas* (tuplas) ou *bloqueio a nível de páginas*. Em alguns produtos existe a possibilidade de se especificar o *bloqueio a nível de tabelas*.

O bloqueio a nível de linhas bloqueia somente as linhas envolvidas na transação. O bloqueio a nível de página bloqueia toda uma página de dados que contenha a(s) linha(s) envolvida(s) na transação. Se as páginas de dados contiverem mais de uma linha, pode-se causar o bloqueio desnecessário de linhas. O principal argumento a favor do bloqueio a nível de páginas fica por conta de um desempenho superior. Os fornecedores que implementam bloqueio a nível de linhas argumentam que os ganhos em níveis de concorrência justificam o overhead extra que surge com a manutenção de atualizações neste nível.

Alguns produtos empregam um esquema de múltiplas versões para cada linha envolvida em uma determinada transação ***tx1***. Estas versões são mantidas pelo período que existirem *leitores* ativos. Quando todos os leitores tiverem completado suas transações, as versões antigas de quaisquer linhas envolvidas na transação ***tx1*** são liberadas, e quando for ativado o processo de coleta de lixo ("garbage collection"), o espaço ocupado será então liberado. Este enfoque garante a leitura repetida de linhas, pois cada linha envolvida em uma transação é mantida pela duração desta transação.

7.4.6. Otimização de Consultas

No início do uso de SGBDs, os DBAs gastavam horas ajustando as consultas às bases de dados, de tal forma que se soubesse exatamente como os dados deviam ser acessados e que índices deviam ser usados [Ull88]. Nos SGBDs do modelo CODASYL este era um fato [Set86]. Certamente, este é um processo tedioso, porém muito simples, pois o plano de acesso acabava sendo determinado por um conjunto de regras baseadas na sintaxe da linguagem de manipulação de dados envolvida.

No início do uso de SQL em SGBDs relacionais, o processo de otimização de consultas também era feito desta maneira. Criava-se uma tabela com um conjunto de regras baseadas na sintaxe de SQL. Por exemplo, a primeira junção em uma declaração SQL (em uma leitura de cima para baixo) deveria ser sempre executada em primeiro lugar. Este tipo de otimização é chamado de *otimização baseada em sintaxe* (ou *baseada em regras*). O principal problema com este enfoque é a necessidade de manutenção constante. À medida que o tamanho de várias tabelas fosse mudando, as consultas deveriam ser re-otimizadas manualmente.

Atualmente, existem disponíveis algoritmos de otimização bastante sofisticados, que avaliam as cláusulas em uma declaração SQL [Mye93b]. Baseados em estatísticas sobre as tabelas envolvidas e nos índices disponíveis, um plano de acesso é criado para minimizar operações de Entrada/Saída e tempo de processamento. Geralmente, estes *otimizadores baseados em custos* podem produzir o melhor plano de acesso para uma consulta. Eliminam também as horas extras de ajuste. No entanto, este otimizadores não são totalmente confiáveis, e as técnicas de otimização empregadas variam de fornecedor para fornecedor. Observe que o programador deve poder passar por cima destes otimizadores quando necessário. Em muitos produtos esta opção esta disponível, onde pode-se sobrepassar o otimizador e forçar o servidor a usar a otimização baseada em sintaxe. Pode-se ainda influenciar o otimizador, forçando-o a usar um índice específico indicado pelo programador.

7.4.7. Processamento Distribuído

Com a proliferação de bancos de dados e dos sistemas distribuídos, a integração de múltiplos servidores e/ou múltiplos sítios não é tarefa das mais fáceis. Atualmente, as facilidades incluídas em alguns SGBDs fornecem *transparência de localização* [LS90]. Todos os usuários acessam seus dados em seu sítio primário. Um DBA pode mover os dados de um servidor para outro sem que os usuários ou as aplicações precisem ter conhecimento deste fato. Adicionalmente, os *serviços de replicação* [Ges93a] podem ser usados para copiar fisicamente dados de seu sítio primário para outros sítios, automaticamente. Todos estes conceitos foram vistos detalhadamente no capítulo 5 e 6 anteriores, quando falamos de serviços de nomes e de arquivos distribuídos, e aqui eles são aplicados de maneira consistente.

À medida que os SGBDs cliente/servidor tornam-se mais populares, maiores, e mais dispersos geograficamente, uma série de novas preocupações começa a surgir. Teoricamente, pode-se distribuir um banco de dados por diferentes nós em uma rede de longo alcance, e pode-se ainda ter acesso em tempo real a estes dados. Teoricamente, pode-se também separar a estação cliente de seu servidor via ligações de longa distância. Alguns SGBDs de mercado disponibilizam na prática estas duas possibilidades, através de rotinas ou esquemas de nomes muitas vezes proprietários (por meio de *catálogos*). Por exemplo, uma tabela em uma declaração SELECT pode ser definida localmente, ou ela pode ser um apelido para uma tabela em outro servidor. Existe implementada em alguns produtos até a possibilidade de se tratar junções remotas (por exemplo, juntar uma tabela Oracle em São Paulo com uma tabela DB2 em Fortaleza. Já pensou!). Porém produtos deste tipo são vendidos a parte, via gateways, proprietários na maioria dos casos [Wat93b].

Outra funcionalidade que pode ser encontrada é a possibilidade de se permitir a duplicação de dados em alguns sistemas, por razões puramente práticas. O principal motivo para esta duplicação é a dispersão geográfica. A segunda é a utilização racional dos servidores, evitando gargalos. No entanto, a simples duplicação de dados pode gerar conflitos ("*deadlocks*").

Para resolver este problema, surgiram os *serviços de replicação* [Ges93a]. Os servidores de replicação podem ser usados para mover dados de uma base de dados para outra de maneira automática. Pode-se usar uma sintaxe tipo SQL para "cadastrar" ou "registrar" os interesses em um subconjunto particular de transações. Por exemplo, pode-se requisitar cópias de todas as transações processadas em São Paulo. Quando estas transações são processadas, elas são enviadas para um banco de dados remoto e processadas por lá também. Adicionalmente, pode-se configurar a freqüência com a qual as transferências são feitas de uma base de dados para outra. Com uma freqüência de segundos ou minutos, pode-se dispor de acesso quase em tempo real aos dados em sítios remotos. Se uma das necessidades em uma organização usuária de um banco de dados deste tipo for reduzir os custos de telecomunicações, pode-se definir um intervalo maior entre as transferências.

Com serviços de replicação, pode-se potencialmente tratar os conflitos. Pode-se designar uma determinada localização como a dona de partes do banco de dados. As atualizações só poderiam ser feitas pelo dono dos dados. Este modelo é conhecido como *modelo de **dono** primário* [Dew94]. Em alguns casos, pode-se permitir que todos os sítios possam fazer atualizações, de tal forma que os conflitos sejam resolvidos via programação, ou via módulos sofisticados de controle de concorrência que implementam o protocolo 2PC, por exemplo.

7.4.8. Suporte a Múltiplos Processadores

Para que se aproveitem as máquinas multiprocessadas, o servidor deve ser implementado como um programa *multi-threaded* [Pou93a]. O sistema operacional projetado para rodar em servidores multi-processados despacha partes de programas para cada um dos processadores. Estas partes são chamadas de *threads,* também já apresentadas aos leitores no capítulo de introdução e conceitos básicos. Em sistemas operacionais com multiprocessamento simétrico, suporta-se uma área de memória partilhada que coordena as várias tarefas, e o sistema operacional é responsável pelo ajuste de carga das diversas tarefas entre os diferentes processadores [PB92]. Assim, tarefas de bancos de dados que utilizam CPU de maneira intensiva, como a carga de bases de dados, montagem de índices, ou junções podem ser despachadas para múltiplos processadores, para que se obtenha uma vazão ("throughput") máxima dos mesmos. Alguns SGBDs já implementam a construção de índices, classificações, recuperação lógica, e cópias/recuperação de segurança (backups) para arquivos, tudo em paralelo. Oracle e Informix, em suas versões mais recentes (versão 7 e versão 7.1, respectivamente) implementam níveis elevados de paralelismo em seu código.

7.4.9. Índices

O principal objetivo por detrás do uso de índices diz respeito às questões de desempenho. Dependendo da natureza dos dados, diferentes tipos de índices podem levar a diferentes características de desempenho. Portanto, é muito importante que se determine exatamente o tipo de índice disponível para uma determinada aplicação.

As possibilidades incluem *árvores balanceadas ("B-Trees")*, **ISAM** (*Indexed Sequential Access Method*), índices randômicos ("*hash*") e "*clusters*" [Wie86].

Os *índices com árvores balanceadas* empregam uma estrutura do tipo raiz e folhas, e geralmente fornecem bom desempenho. Os *índices em "clusters"* armazenam dados juntamente com um índice para referenciá-los. Em alguns casos, os dados já são classificados por natureza, devido à ordenação imposta pelo índice, o que elimina operações de classificação dispendiosas. Devido a esta organização, requisita-se uma menor quantidade de acessos a discos. Não existe desgaste ("thrashing") de busca de índices e de páginas de dados. Os *índices de randomização* armazenam os dados em ordem randômica, baseados em um valor de chave de randomização. São de extrema utilidade para acessos aleatórios. Sua funcionalidade pode ser degradada se múltiplas linhas partilharem o mesmo valor de índice (ocorrem as colisões de randomização).

No caso de chaves primárias e secundárias serem incluídas nas definições de tabelas, e estas definições serem usadas para forçar a integridade referencial, esta funcionalidade é chamada de *integridade referencial declarativa*. Por default, a tentativa de remover uma linha referenciada como uma chave estrangeira em outra tabela é proibida, ou restrita, isto é, emite-se uma mensagem de erro. Em alguns casos no entanto, a remoção pode ser propagada (cascateada) pelas tabelas relacionadas. Em alguns produtos, pode-se ainda substituir os valores da linha com nulos, ao invés de removê-la.

7.4.10. Transferência de Dados em Grandes Volumes

Existem vários tipos de aplicações que podem ter seu desempenho melhorado pela transferência de dados em grandes volumes. O exemplo típico é a carga inicial do banco de dados. Por exemplo, quando se faz a conversão de uma aplicação legada ("legacy") [Ges93c] para um SGBD cliente/servidor relacional, existe geralmente uma grande quantidade de dados que devem ser importados para as novas bases de dados. Existem outras situações, como por exemplo a consolidação de dados ou a reorganização de um departamento, onde os dados devem ser movidos por inteiro de um servidor para outro. Freqüentemente estas operações ocorrem "off-line", ou seja, com as aplicações da rede fora do ar.

7.5. Os Padrões e Extensões para a Linguagem SQL

A linguagem SQL foi desenvolvida no início dos anos setenta pelo Dr. Edgar Codd [Dat81], na época um programador da IBM. Com o tempo, e com a proliferação dos SGBDs relacionais, SQL tornou-se o padrão de facto, e mais recentemente um padrão de jure, para a manipulação de bancos de dados.

Em 1986, o ANSI publicou o primeiro padrão oficial para SQL, chamado de "Database Language SQL", ou padrão ANSI X3.135-1986, popularmente conhecido por SQL-86. Este padrão representava uma espécie de denominador comum para as implementações SQL existentes no mercado, cada uma com um conjunto particular de extensões. O que mais fazia falta em SQL-86 era a falta da integridade referencial. Em 1989 a ANSI publicou o padrão "Database Language SQL With Integrity Enhancement", ou padrão ANSI X3.135-1989, conhecido por SQL-89. A principal mudança foi justamente a inclusão da integridade referencial em suas recomendações. Em 1992 a ANSI publicou a última revisão SQL disponível, que é o padrão ANSI X3.135-1992, ou SQL-92. Espera-se que o ANSI libere até 1996 uma nova revisão do padrão, que irá incluir o suporte à tecnologia de orientação a objetos [Men94a].

O fato é que com a proliferação de padrões de conectividade (que veremos a seguir) e de bancos de dados heterogêneos no mercado, os padrões para SQL têm se tornado mais importantes. Porém, das implementações disponíveis no mercado, é pouco comum encontrarmos todas as especificações recomendadas pelos padrões oficiais. Assim, podemos considerar que, na realidade, existem três padrões de indústria (*de facto*) disponibilizados a nível de produtos implementados. Chamamos a estes três padrões de facto de ***SQL Inicial, SQL Intermediária,*** e ***SQL Cheia*** [Cla93b]. SQL inicial consiste da implementação das especificações do SQL-89 com poucas correções e melhorias. SQL Inicial corrige pequenas inconsistências da especificação padrão SQL-89, e adiciona suporte de interfaces para linguagens como Ada, C, e MUMPS. SQL Intermediária representa a implementação de aproximadamente metade das especificações da SQL-92, e SQL Cheia inclui adicionalmente características que foram consideradas particularmente difíceis de serem implementadas, ou que foram consideradas de menor importância para o mercado, do conjunto total das especificações SQL-92.

Características de SQL Intermediária e SQL Cheia

A Tabela 7.1 (baseada em [Men94a]) lista as características incluídas em SQL Intermediária e SQL Cheia. Algumas características obviamente fazem falta em SQL-92, como gatilhos e procedimentos armazenados. É importante que se esclareça que nenhum produto disponível implementa todo oconjunto destas características.

- *SQL Dinâmica*: dizemos que uma declaração, ou algum de seus componentes, são dinâmicos, quando não são determinados até que sejam executados. Conseqüentemente, o servidor deve ligar esta declaração, e retornar alguma informação sobre o resultado desta ligação para o programa chamador. Em SQL-92 foram criados padrões para declarações DECLARE, PREPARE, DESCRIBE e EXECUTE.

- *Manipulação de esquemas*: fornece suporte a operações DROP, ALTER e ADD sobre tabelas depois que elas são criadas.

- *Controle de transações melhorado*: são introduzidos níveis de isolamento. Existem agora operações do tipo READ ONLY/READ WRITE.

- *Remoção cascateada*: pode-se declarar como parte de uma tabela, uma ação a ser tomada quando uma referência a uma chave estrangeira se altera.

- *Conjunto de operações avançadas*: estes novos operadores incluem UNION JOIN, INTERSECT e EXCEPT. Uma união simples (UNION) requisita dois conjuntos com a mesma estrutura. Em UNION JOIN, concatena-se dois conjuntos sem levar em consideração que colunas existem nos dois conjuntos envolvidos. As colunas do primeiro conjunto são nulificadas no segundo conjunto, e vice-versa. INTERSECT retorna todas as linhas no primeiro conjunto que encontram-se também no segundo conjunto. EXCEPT retorna todas as linhas que estão no primeiro conjunto mas não estão no segundo.

SQL Intermediária	SQL Cheia
SQL Dinâmica (DECLARE, PREPARE, DESCRIBE, EXECUTE)	Cursores de rolagem
Declarações para manipulação de esquemas	Subconsultas
Níveis melhorados de isolamento e controle de transações	Restrições de nomes/checagem postergada de restrições
CASCADE DELETE	CASCADE UPDATE
Conjunto avançado de operações (UNION JOIN, EXCEPT, INTERSECT)	Auto-referenciamento p/ DELETE/UPDATE
CASE, CAST, COALESCE	Características de internacionalização
Cadeias de caracteres de tamanho variável	Tipo de dados "bit"
Construtores de valores p/ tabelas e linhas (p.ex.: literais)	Tabelas temporárias
Gerenciamento da conexão (CONNECT, DISCONNECT)	Asserções (restrições entre tabelas)
Funções Datetime e tipos de dados/operações de intervalo	Funções Datetime adicionais e outros tipos de dados/operações de intervalo
Operações avançadas para cadeias de caracteres	
Suporte a domínios (classes e tipos de dados)	
Portabilidade de aplicações	
Informações adicionais de diagnóstico	

Tabela 7.1. Características incluídas em SQL Intermediária e SQL Cheia [Men94a]

- *Valores construtores de linhas e colunas:* pode-se qualificar as colunas em várias tabelas. Por exemplo, a sintaxe a seguir poderia ser usada em uma comparação entre campos de duas tabelas.

WHERE (A.Col1, A.Col2, A.Col3) = (B.Colx, B.Coly, B.Colz)

- *CASE, CAST, COALESCE:* a declaração CASE pode ser usada em declarações como UPDATE ou SELECT, o que pode facilitar o tratamento de valores condicionais. Por exemplo, se um programador desejar preparar um relatório sobre o estado do estoque local em uma empresa, poder-se-ia escrever a seguinte declaração:

```
SELECT Num_peça, Nome_peça,
  CASE
      WHEN  Qtde_estoque = 0
            THEN "Estoque zerado!"
      WHEN  Qtde_estoque <= Qtde_compra
            THEN "Emitir pedido de compra para abastecer estoque!"
            ELSE "Estoque OK!"
  END,
      Nível_estoque
FROM Estoque
```

A declaração CAST fornece conversão de um tipo de dado para outro. COALESCE é simplesmente uma opção especial para o CASE, que retorna o primeiro valor não nulo de uma lista de valores.

- *Operações Datetime e tipo de dados/operações de intervalo:* existem tipos de dados padrões, chamados tipos *datetime*, para DATE, TIME e TIMESTAMP. Os intervalos podem ser declarados na forma: *Ano-mês,* ou *Dia-hora*. Um intervalo ano-mês representa o número de anos e meses entre dois valores tipo *datetime*. Um intervalo dia-hora representa o número de dias, horas, minutos, e/ou segundos entre dois valores datetime. Valores datetime ou intervalos podem ser adicionados ou subtraídos para resultarem valores em programas que computam diferenças de datas (o que em termos de programação é excelente). Por exemplo, poder-se-ia escrever um programa para marcar datas de visitas a um médico para check-up da seguinte maneira:

```
SELECT nome-paciente, última-visita,
      Próxima-visita = última-visita + INTERVAL "6" MONTH
FROM Paciente
```

- *Operações avançadas com cadeias de caracteres:* funções do tipo Char_length(), Posítion(), Extract(), Substring(), e Trim() foram incluídas para suportar o tratamento sofisticados de tipos de dados cadeias de caracteres.

- *Suporte a domínios:* pode-se declarar tipos de dados particulares, de uso privado para um programador.

- *Portabilidade de aplicações:* pode-se indicar explicitamente por meio de um "flag" se foram usadas declarações não-padronizadas no programa.

- *Informações adicionais de diagnóstico:* transações podem declarar uma área de diagnósticos e podem usar a declaração GET DIAGNOSTICS para gerar relatórios com informações adicionais de erros.

7.6. Padrões para a Conectividade de Bancos de Dados

Nos ambientes cliente/servidor da computação distribuída, é muito comum a interação entre diferentes SGBDs. Por exemplo, uma aplicação cliente rodando em uma estação de trabalho pode utilizar uma API do Informix para acessar os dados em um servidor remoto, ao mesmo tempo que utiliza o SQL Server para acessar a base de dados local.

Dos padrões para conectividade de bancos de dados [Hop93a], que forneçam facilidades como a do exemplo anterior, podemos destacar os seguintes: o padrão de jure da ISO, ainda em desenvolvimento, voltado para fornecer acesso geral a SGBDs remotos, chamado **RDA** (*Remote Database Access*) ([Hal93b] [Wat93b] [Dew93] [Dew94]); o padrão de facto, desenvolvido pela Microsoft, e objeto de estudo deste parágrafo, chamado **ODBC** (*Open Data Base Connectivity*) ([Hop93a] [Hop93b] [Hop94] [Col93b]); o padrão proposto pelas empresas Borland, IBM, Novell e WordPerfect, também ainda em desenvolvimento, chamado **IDAPI** ([Wat93b] [Col93c]); e, finalmente, o padrão da IBM, chamado **DRDA** (*Distributed Relational Database Architecture*) ([Wat93b] [Dew93] [Dew94]). A seguir, falamos um pouco sobre cada um destes padrões.

RDA da ISO

RDA (Remote Database Access) ([Hal93b] [Wat93b] [Dew93] [Dew94]) é o protocolo da ISO proposto para o processamento de transações em múltiplos sítios. Este protocolo não é uma infra-estrutura ou uma arquitetura geral, é simplesmente um protocolo de comunicação otimizado para o acesso a dados remotos. Na realidade, RDA adequa-se ao modelo de referência de sete camadas OSI da ISO. O padrão é dividido em um módulo genérico, que define uma sintaxe comum de transferência de dados e um protocolo de acesso a base de dados, mais especializações, que adequam o modelo genérico para uso com modelos de dados ou linguagens específicos.

No ambiente DRDA da IBM, como veremos a seguir, uma aplicação é conectada a uma base de dados de cada vez. No RDA da ISO, uma aplicação pode ser conectada a mais de uma base de dados várias vezes. No entanto, o enfoque RDA ainda não resolveu na prática uma série de problemas cruciais relacionados com a atualização de múltiplas bases de dados, como por exemplo a recuperação de falhas, e a coordenação destas recuperações.

Remote Database Access corresponde à proposta da ISO para um padrão geral de conectividade avançada de bancos de dados. É um padrão em início de desenvolvimento, e temos que esperar ainda alguns anos até que seja publicada, pelo menos, a primeira proposta rascunho. RDA consiste de um protocolo que busca a universalidade entre os diversos servidores de bancos de dados. O Grupo de Padronização de SQL (SQL Access Group) formado pela DEC, Sybase, Informix e Ingres, dentre outros, endossa completamente a proposta RDA [Cas93].

DRDA da IBM

DRDA (*Distributed Relational Database Architecture*) ([Wat93b] [Dew93] [Dew94]) é uma arquitetura proprietária para interoperabilidade entre SGBDs cliente/servidor em ambientes IBM e não-IBM. DRDA baseia-se em outras tecnologias IBM, como por exemplo, SNA e LU.62 [Cas93]. DRDA fornece a conectividade necessária entre SGBDs relacionais, e pode operar em ambientes homogêneos (só IBM) e heterogêneos. SQL é usada como a linguagem para acesso comum.

DRDA consiste de um enfoque de protocolo comum para grandes sistemas. Suporta uma série de funcionalidades que permitem o gerenciamento facilitado de grandes ambientes de computação, como alarmes NetView, blocagem de dados e sintaxe SQL. Seu futuro será determinado por bancos de dados e soluções de conectividade ofertadas por fornecedores para a conexão com servidores não-IBM. Apesar de 14 fornecedores já terem adquirido uma licença arquitetural para o DRDA, nenhum dos principais fornecedores de bancos de dados anunciou conectividade aos seus servidores via o DRDA. A Tabela 7.2 fornece uma comparação entre as especificações dos padrões propostos pela ISO e pela IBM.

DRDA/IBM	DRA/ISO
Voltada para desempenho	voltada para portabilidade facilitada
Baseada em SNA	baseada no modelo de referência OSI/ISO
Cada servidor possui SQL nativo	Somente um subconjunto de SQL suportado
Maximiza o suporte ao "legacy"	Minimiza a necessidade de ferramentas diferentes em diferentes servidores
Enfoca a interoperabilidade com o mundo IBM	Enfoca a interoperabilidade em ambientes abertos, heterogêneos e com múltiplos fornecedores
Bancada pela IBM mais outros nove fornecedores	bancada pela grande maioria dos fornecedores de bancos de dados, exceto a IBM

Tabela 7.2. DRDA/IBM comparado com RDA/ISO [Dew94]

ODBC da Microsoft

ODBC (Open Database Connectivity) ([Hop93a] [Hop93b] [Hop94] [Col93b]) foi criada pela Microsoft para tentar resolver problemas que surgem na implementação de múltiplas APIs para bancos de dados. Geralmente, em uma arquitetura de bancos de dados distribuído, o desenvolvedor de aplicações precisa conhecer diferentes bibliotecas de aplicações, o que pode ser muito desgastante em termos de produtividade, pois complica tanto o desenvolvimento quanto a manutenção do código. A cada vez que surge no mercado uma nova versão para um SGBD, deve-se alterar o conjunto de funções implementada. Portanto, suportar múltiplas APIs é algo desastroso para o desenvolvimento de aplicações distribuídas. Buscas por soluções novas oferecidas em SGBDs melhorados acabam sendo desencorajadas em ambientes como este, pois os programadores têm que ser treinados novamente na nova API, e o código antigo tem que ser convertido para o novo protocolo.

Com ODBC parte destes problemas estão resolvidos. Um desenvolvedor de aplicações só precisa aprender somente os detalhes de uma única biblioteca de chamadas ao banco de dados, que é justamente a API ODBC. Os detalhes a respeito de um SGBD em particular ficam escondidos do desenvolvedor, pois o ODBC cuida de rotear as chamadas de funções para um **driver** criado especificamente para o banco de dados de destino. Este conceito é similar àquele de independência de hardware em sistemas operacionais. Assim, os programadores que estiverem usando ODBC não precisam preocupar-se com a implementação de cada banco de dados, os drivers cuidam dos detalhes para eles.

Um servidor de banco de dados em ODBC é chamado de Fonte de Dados ("*data source*"), que consiste das bases de dados mais os módulos de acesso necessários, incluindo a rede e o sistema operacional. O Gerente de Drivers ("driver manager") é um programa ODBC que roteia consultas para os drivers apropriados. A Figura 7.7 ilustra a arquitetura das aplicações ODBC.

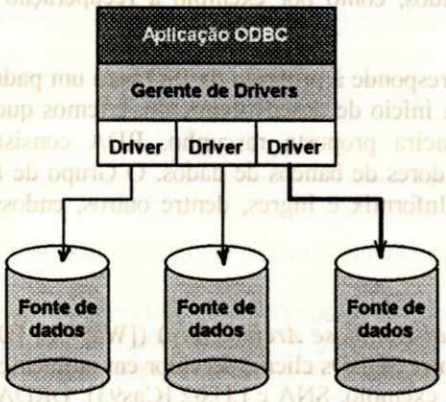


Figura 7.7. A arquitetura de componentes do ODBC [Hop93a]

Para resumir a arquitetura, uma aplicação emite uma chamada para a camada ODBC que inclui todos os serviços ODBC (como por exemplo, o processamento de consultas). Esta chamada resulta em outra chamada para o Gerente de Drivers, que cuida de estabelecer qual dos drivers é apropriado para a fonte de dados envolvida, e roteia a chamada para este driver. Se necessário, o driver fará a tradução da chamada em sintaxe SQL para uma sintaxe que o servidor possa entender. O driver cuida então de despachar a consulta para a fonte de dados.

IDAPI

IDAPI ([Wat93b] [Col93c]) corresponde a uma resposta de vários fornecedores ao padrão de facto ODBC da Microsoft. Os principais participantes do IDAPI são Borland, Novell, IBM, e WordPerfect. IDAPI ainda está sendo desenvolvida, e parece configurar-se como um padrão mais robusto que o ODBC. IDAPI é uma interface comum para a conectividade de bancos de dados, isto é, a interface do cliente assume a responsabilidade de fazer com que todos os servidores de bancos de dados envolvidos em uma aplicação distribuída comportem-se de maneira comum. Diversas empresas apoiam a especificação IDAPI.

7.7. Front-Ends para SGBDs Cliente/Servidor

Os softwares de front-end ([Gre93] [Sal93]) que rodam nas estações clientes acessam os servidores back-end para efetuar pedidos do tipo consultas, atualizações ou deleções nas bases de dados. Estes softwares podem ser pacotes prontos para usuários finais ou podem ser programados diretamente por usuários especializados que tenham em mãos ferramentas adequadas para desenvolvê-los. A seqüência de processamento de pedidos aos servidores back-end já foi apresentada anteriormente, quando falamos sobre a arquitetura cliente/servidor, mas vale a pena revermos como ela se desenvolve. Na figura 7.8 abaixo fazemos um resumo rápido da seqüência de eventos que ocorrem quando um pedido de usuário é submetido ao servidor back-end.

Neste caso, o usuário submete um pedido, que é traduzido pelo software front-end para a linguagem SQL (na versão implementada pelo servidor back-end), e a envia pela rede. O servidor checa os direitos de acesso do usuário e processa todas as operações complexas envolvidas, enviando a seguir os resultados de volta pela rede. O front-end cliente recebe os dados e os formata para apresentação seja na tela seja em um relatório impresso.

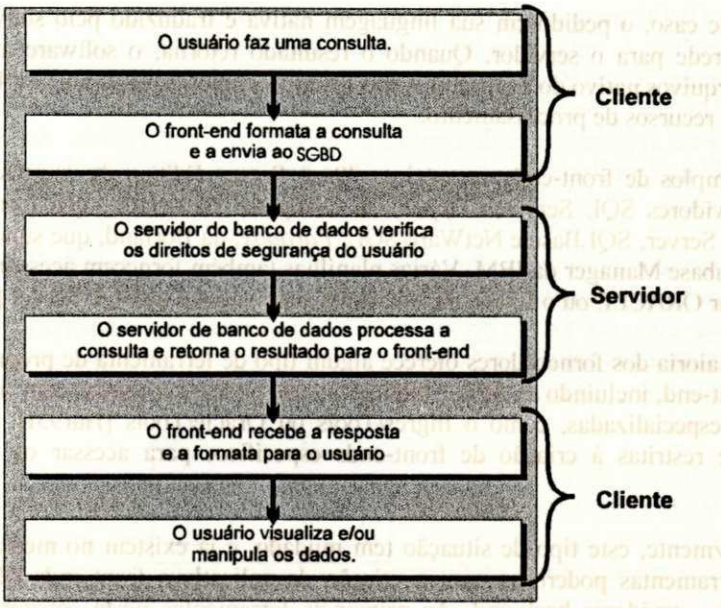


Figura 7.8. O processamento no estilo cliente/servidor para SGBDs back-end [Sal93]

Observe que existe a necessidade de traduzir-se o pedido original para uma versão SQL adequada para o servidor back-end que estiver sendo usado. Isto é necessário porque a linguagem SQL apesar de ser um padrão endossado por órgãos internacionais de padronização como a ANSI, apresenta muitas variantes, com extensões proprietárias e ligeiramente incompatíveis de SGBD para SGBD. Este fator limita, portanto, as ofertas de front-ends para um conjunto reduzido de servidores back-end. Adicionalmente, não basta fazer a tradução para SQL no front-end. Os projetistas dos fornecedores de front-ends devem conhecer as APIs adequadas para o driver de comunicação do SGBD escolhido.

O tipo mais comum de software front-end são chamados de *acessórios* [Sal93]. Estes acessórios são fornecidos por produtos típicos do ambiente dos PCs, e de uso facilitado, como por exemplo os servidores de arquivos do tipo xBase ou as planilhas eletrônicas. A vantagem no uso destes produtos fica por conta da redução no custo de desenvolvimento de aplicativos front-end, pois não existe necessidade de treinamento extra para que os usuários possam acessar os servidores de bancos de dados por meio de ferramentas já conhecidas.

Quando estes softwares acessórios são utilizados, a seqüência de eventos para um pedido básico muda um pouco, e fica como mostrado na figura 7.9.

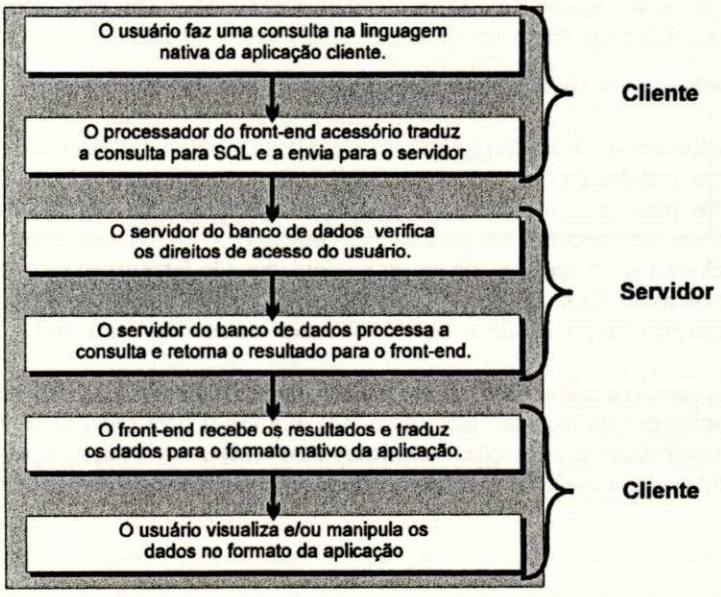


Figura 7.9. O processamento no estilo cliente/servidor para front-ends acessórios [Sal93]

Neste caso, o pedido em sua linguagem nativa é traduzido pelo software acessório para SQL, o qual é enviado via rede para o servidor. Quando o resultado retorna, o software acessório traduz o resultado para o formato de arquivos nativo do front-end. Estas traduções adicionais geram overhead na estação cliente, o que passa a exigir mais recursos de processamento.

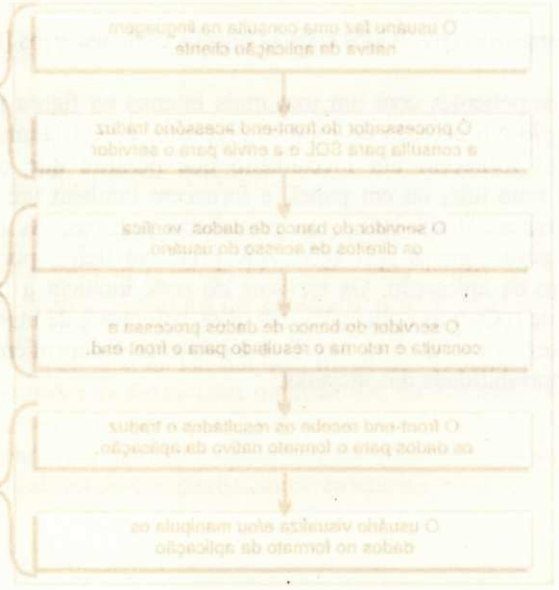
Exemplos de front-ends acessórios: *dBase Server Edition* da empresa Borland, projetado para fornecer acesso a servidores SQL Server; *Clipper*, da Computer Associates, que fornece acesso tipo front-end para os SGBDs SQL Server, SQLBase e NetWare SQL; *Paradox*, da Borland, que suporta ORACLE, NetWare SQL, SQL Server e Database Manager da IBM. Várias planilhas também fornecem acessórios de front-end, como o Lotus 1-2-3 para acessar ORACLE ou o Excel, da Microsoft, para acessos ao SQL Server [Hur93g].

A maioria dos fornecedores oferece algum tipo de ferramenta de programação para o desenvolvimento de software front-end, incluindo as APIs e bibliotecas que podem ser usadas com linguagens de terceira geração, ou as ferramentas especializadas, como o Ingres/Tools ou Oracle/Tools [Hur93f]. No entanto, estas ferramentas são normalmente restritas à criação de front-ends específicos para acessar os SGBDs do próprio fornecedor da ferramenta.

Felizmente, este tipo de situação tem mudado, e já existem no mercado bons produtos de terceiros que fornecem ferramentas poderosas para a criação de aplicativos front-ends ([Mye93b] [Bau94c] [Bau94c]) para acessar vários servidores back-end. As principais ferramentas nesta categoria são o Power Builder, e o SQL Windows.

O PowerBuilder [Hor93a], da empresa PowerSoft Co., é um ferramenta poderosa para os projetistas de front-ends, que fornece tecnologia de orientação a objetos e é voltada principalmente para programadores experientes que desenvolvem aplicações departamentais ou corporativas. Permite acesso a diversos servidores de bancos de dados, incluindo: SQL Server, SQLBase, ORACLE, Ingres e DB2. Ele não fornece suporte a bancos de dados nativo, e depende de um destes servidores para a construção e testes de aplicações.

O SQLWindows [God93b], da empresa Gupta, é também uma poderosa plataforma para o desenvolvimento de aplicativos. Permite acesso a servidores de bancos de dados baseados em SQL Server, ORACLE, Ingres, NetWare SQL, e SQL 400. É também um ambiente orientado a objetos, e fornece uma linguagem chamada SQLWindows Application Language (SAL), que fornece capacidade interna para depuração e criação de aplicativos em módulos executáveis. Existem ainda ferramentas disponíveis para os usuários menos experientes, de tal forma que eles possam criar front-ends sem precisarem ser programadores experientes.



Capítulo 8: Interfaces Gráficas para Usuários

8.1. Introdução

Era muito comum, em ambientes tradicionais da computação, encontrarmos sistemas que comumente não forneciam um componente de interface com o usuário de forma explícita. O trabalho da implementação da interface com o usuário ficava por conta do próprio programa a ser desenvolvido, que utilizava outros componentes de sistema (como por exemplo, geradores de telas orientadas a caracteres e funções especiais para a ativação destas telas, a serem incluídas no código dos programas).

A interface de usuários é um dos componentes mais importantes em sistemas abertos. Todos aqueles que utilizam sistemas de computadores estão interessados em características de uso fácil e comuns nos mais diversos produtos, com a conseqüente redução da necessidade de retreinamento de pessoal e com uma sensivel diminuição na curva de aprendizado. A estas características comuns denominamos de características do tipo "*olhar-e-sentir*" (*"look and feel"*). Portanto, a busca é por aplicações que forneçam o mesmo "look and feel" aos seus usuários. As ***interfaces gráficas de usuários*** ([Cas93] [Gra91] [Dum91] [Ber91] [Sey89] [Dew93] [Dew94] [KA90]) foram desenvolvidas para atender a estas necessidades.

8.1.2. Conceitos Básicos e Visão Geral

Vários subsistemas estão envolvidos com a disponibilização de interfaces para usuários. A figura 8.1. mostra um modelo que identifica a localização das interfaces de usuários com relação a estes subsistemas.

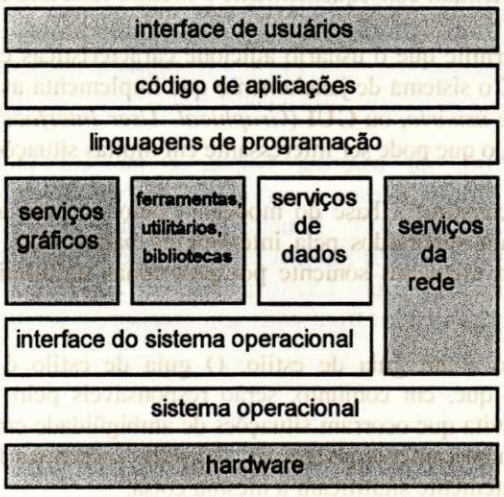


Figura 8.1. Os subsistemas envolvidos com as interfaces de usuários [Cas93]

Marcamos alguns destes componentes com um tom mais intenso na figura para falarmos um pouco de cada um deles. A *interface de usuário* é o coração do modelo proposto, e dela falamos extensivamente ao longo deste capítulo. Os *serviços gráficos* fornecem um mecanismo que permite que os programas manipulem e desenhem contornos arbitrários em uma tela, ou em papel, e fornecem também um meio para que os usuários possam utilizar recursos do tipo mouses, "track-balls", ou pranchetas gráficas. As *bibliotecas* são usadas para disponibilizar os serviços gráficos para aquelas aplicações que precisem deles, por meio de funções a serem invocadas a partir do próprio código da aplicação. Os *serviços da rede* incluem a transferência de dados entre terminais e estações hospedeiras, onde rodam as aplicações, em altos volumes e de maneira altamente confiável. O *hardware* envolvido deve, por sua vez, fornecer uma interface totalmente independente da aplicação, de forma a tornar transparentes as questões de portabilidade das mesmas.

8.1.2.1. O Modelo de Referência em Camadas para as Interfaces de Usuários

Um modelo de camadas é proposto em [Gra91] para descrever os componentes dos sistemas de interfaces de usuários. Este não é um modelo formal, porém será muito útil para ajudar a didática referente a este capítulo. Este modelo proposto está ilustrado na Figura 8.2.

Figura 8.2. O Modelo de referência em camadas para as interfaces de usuários [Gra91]

No modelo, a camada mais alta, contém os *programas de aplicações* que irão acessar a interface de usuário por meio de ferramentas poderosas que serão fornecidas na camada imediatamente inferior, um *"toolkit"* de *ferramentas da interface de usuário*. Estas ferramentas são voltadas para tarefas como a criação e a manutenção de objetos como menus e janelas, e fornecem facilidades para redefinição de tamanho e movimentação destes objetos na tela dos terminais. São facilidades deste tipo que compõem o que já chamamos de "look and feel". Aplicações que utilizam o mesmo toolkit apresentam conseqüentemente o mesmo look and feel [Dum91].

- O *sistema de janelamento* permite que o usuário adicione características extras ao "look and feel" fornecido pelo toolkit de ferramentas. É o sistema de janelamento que implementa as características básicas necessárias para uma *interface gráfica de usuário*, ou **GUI** (*Graphical User Interface*). Esta camada pode dar suporte a diferentes toolkits acima dela, o que pode ser interessante em muitas situações.

- A *interface de baixo nível* constitui a base do modelo. Como no caso anterior, é possível que diferentes sistemas de janelamento sejam suportados pela interface de baixo nível. O mais comum é que acessos à camada de baixo nível sejam efetuados somente por programas de administração da própria interface de usuário.

A parte final do modelo é um guia de estilo. O guia de estilo é um documento que descreve os componentes e os procedimentos que, em conjunto, serão responsáveis pelo "look and feel" de uma GUI em particular. É o guia de estilo que evita que ocorram situações de ambigüidade em programas escritos por diferentes programadores. Por exemplo, semanticamente podem ser definidas diferentes funcionalidades para os comandos "copiar" ou "duplicar", que aparentemente significam a mesma coisa.

8.2. As Interfaces de Usuários Orientadas a Caractere

Antes de falarmos sobre as GUIs em particular, vale a pena darmos uma olhada nas interfaces de usuários mais convencionais, usadas há longas datas e típicas nos ambientes de computação até recentemente, que são as interfaces orientadas a caractere ([CTR92a] [UI89b]).

Os terminais que dão suporte às interfaces orientadas a caractere permitem que sejam criadas telas geralmente de 24 ou 25 linhas por 80 colunas, com cada uma destas 1920 (ou 2000) posições na tela sendo endereçáveis. Existem vários padrões definidos para estes tipos de telas [Hub92], como o padrão **ANSI X3.64** de 1979 ou o padrão **ISO 6429** de 1988. O padrão X3.64 fez história ao longo dos anos, e os terminais aderentes a ele são os da família VT100 (e compatíveis) da Digital que, no entanto, incluem diversas características proprietárias, as quais vieram a tornar-se padrões *de facto* com o passar dos anos. Outros terminais aderentes ao padrão ANSI são os terminais emulados por microcomputadores compatíveis com o IBM PC. Na prática, os grandes fornecedores de terminais de hoje em dia fogem quase que completamente dos padrões ANSI ou ISO, incluindo em seus produtos um número infindável de características proprietárias.

8.2.1. Mecanismos para Controle de Terminais Orientados a Caractere

É interessante que em um ambiente de computação, onde existam diferentes equipamentos, sejam suportados diferentes tipos de terminais. É importante que este suporte permita que os programas de aplicações possam escolher de forma transparente em qual destes terminais eles irão rodar. Esta é uma necessidade explícita dos ambientes da computação distribuída.

Em ambientes proprietários, é comum encontrarmos situações onde o próprio programa cuida de incluir o código necessário para controlar as funções de um tipo particular de terminal. Neste caso, as aplicações precisam saber lidar com o tipo específico de terminal suportado no ambiente [Cas93]. Assim, o suporte a terminais em ambientes fechados é tratado de maneira muito simples a partir da inclusão de código especial nos programas de aplicações. Por exemplo, consideremos um caso onde uma aplicação precise limpar a tela do terminal e em seguida responder a uma tecla de função programada. Na Figura 8.3, apresentamos um esquema para esta interação da aplicação com o terminal. Observe que neste caso, se o programa precisar rodar em outro tipo de terminal, parte do programa terá que ser reescrito.

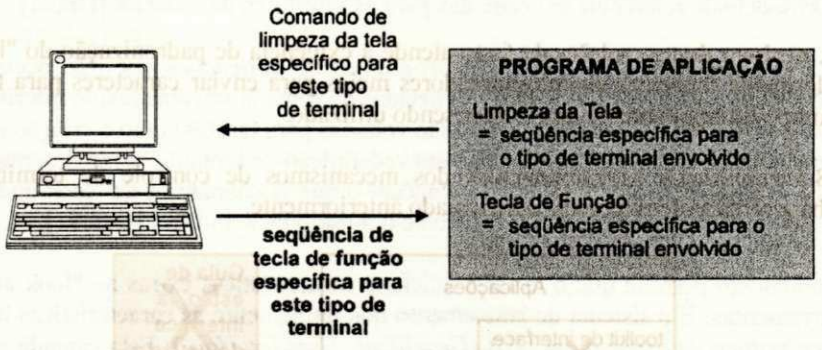


Figura 8.3. Um programa voltado especificamente para determinado tipo de terminal [Hub92]

Para tornar possível a portabilidade de aplicações por diversos tipos de terminais, foi desenvolvido um mecanismo onde as aplicações pudessem recuperar as informações de controle necessárias para tipos particulares de terminais em uma base de dados especializada [UI89]. A aplicação na Figura 8.4 ilustra este mecanismo.

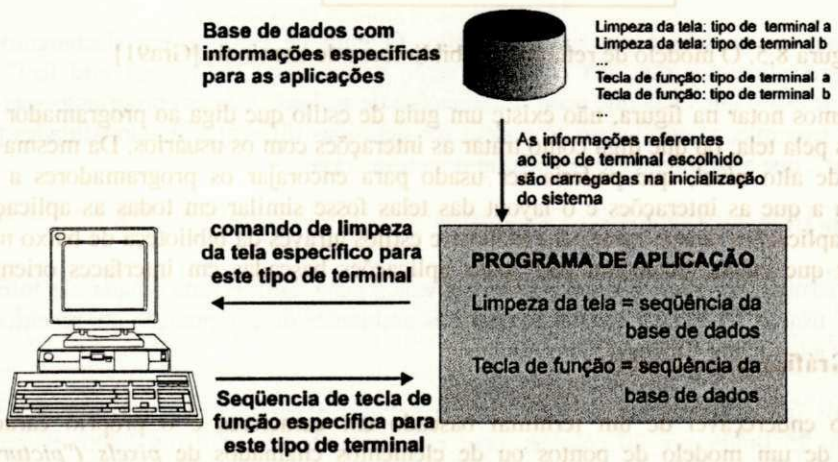


Figura 8.4. Um programa independente do tipo de terminal [Hub92]

Este esquema permite que novos tipos de terminais sejam suportados através de sua inclusão na base de dados. Não é mais necessário que se altere o código das aplicações. Soluções mais sofisticadas para este problema incluem não somente as características do terminal envolvido, mas uma biblioteca completa com a descrição dos mais diversos tipos de terminais disponíveis no mercado.

No mundo Unix, soluções deste tipo existem há muito tempo, e são consideradas soluções padrões *de facto,* e disponíveis em vários ambientes operacionais, inclusive em microcomputadores com MS-DOS ou OS/2, ou em mainframes como no VMS da Digital. Um destes padrões chama-se ***termcap*** [Gra91], que é uma biblioteca de baixo nível usada juntamente com uma base de dados contendo a descrição de todos os tipos de terminais conhecidos pelo ambiente. Ela fornece um meio de se acessar as características individuais de cada tipo de terminal, como limpeza de tela, caractere de inserção, e outros. Termcap forma a base do ***curses***, uma biblioteca de mais alto nível, que permite que programas tratem a tela dos terminais como janelas que contêm caracteres. O curses cuida dos detalhes da tradução das operações de manipulação de dados das janelas nos programas em uma seqüência de operações de baixo nível que irão atualizar a imagem na tela. Termcap foi desenvolvido em 1980, e é considerada obsoleta hoje em dia por desconsiderar características muito importantes para diversos tipos de terminais.

Em 1984, a AT&T desenvolveu uma biblioteca que substituiu o termcap. A nova biblioteca chama-se ***terminfo*** ([Gra91] [UI89]) e foi também desenvolvida uma nova biblioteca de alto nível que continuou a chamar-se *curses*. Curses é considerada um padrão de facto, e é recomendada pelo guia de portabilidade do X/Open (XPG) [HM91].

Infelizmente, nenhum destes padrões de facto atende à exigência de padronização do "look and feel" das aplicações; eles simplesmente fornecem aos programadores meios para enviar caracteres para telas de terminais independentemente do tipo de equipamento que estiver sendo utilizado.

Na Figura 8.5 resumimos os componentes dos mecanismos de controle de terminais orientados a caracteres, tendo por base modelo de referência apresentado anteriormente.

Figura 8.5. O modelo de referência e bibliotecas de terminais [Gra91]

Como podemos notar na figura, não existe um guia de estilo que diga ao programador como distribuir o layout de seus dados pela tela, ou que diga como tratar as interações com os usuários. Da mesma forma, não existe um toolkit padrão de alto nível, que poderia ser usado para encorajar os programadores a adotar um estilo particular, de forma a que as interações e o layout das telas fosse similar em todas as aplicações. Portanto, os desenvolvedores de aplicações devem criar seus toolkits e estilos através da biblioteca de baixo nível, e assim, não podemos considerar que exista "look and feel" para aplicações baseadas em interfaces orientadas a caractere [Wat93a].

8.3. As Interfaces Gráficas para Usuários

O elemento endereçável de um terminal baseado em caracteres é o próprio caractere, geralmente construído a partir de um modelo de pontos ou de elementos chamados de *pixels ("picture elements").* As *interfaces gráficas para usuários*, ou **GUIs**, aproveitam-se das vantagens oferecidas em telas que permitem o endereçamento dos pixels individualmente. Estas telas são chamadas de telas mapeadas em bits ("*bit-mapped*") [Hub92], porque cada pixel corresponde a um bit único na memória (ou eventualmente mais de um pixel, como em casos onde permite-se a combinação de cores em várias escalas de tonalidade). O conceito exposto está mostrado na Figura 8.6.

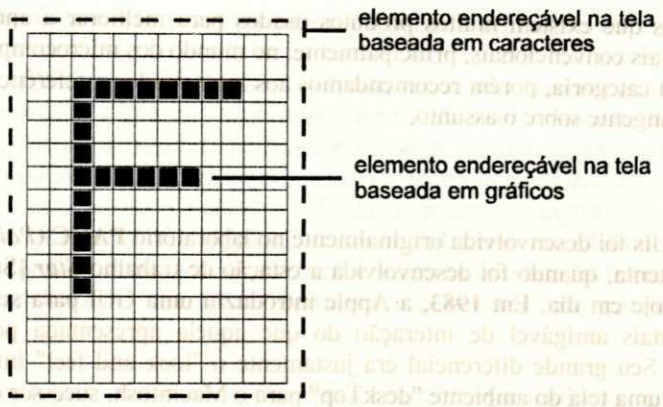


Figura 8.6. Elementos endereçáveis em uma tela [Ber91]

A maior precisão dos displays "bit-mapped" permitem que os caracteres sejam apresentados em vários tamanhos e tipos e que, portanto, sejam completamente substituídos por informações gráficas. As características comumente encontradas em interfaces de usuários orientadas a capacidades gráficas, com displays bit-mapped, incluem: **janelas**, que são áreas retangulares auto-contidas; **ícones**, que são pequenos simbolos que indicam as facilidades oferecidas; e **menus**, que são listas horizontais ou verticais que representam ações ou facilidades alternativas. Um elemento indispensável são os periféricos apontadores, tipo *mouses* ou *track-balls*. A figura 8.7 mostra os elementos que compõem uma GUI.

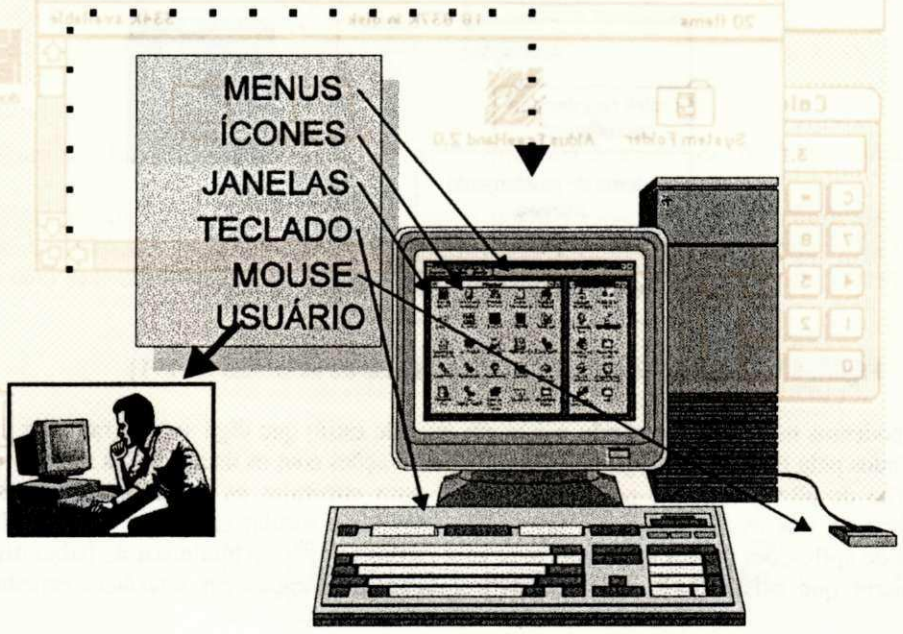


Figura 8.7. Elementos de uma GUI [Dum91]

Em sistemas tradicionais, se uma aplicação desejar produzir uma saída gráfica em um terminal de computador, ela irá geralmente chamar uma biblioteca ou uma subrotina gráfica disponível a nível de software de sistema. Esta subrotina irá executar a tarefa requisitada (por exemplo, desenhar uma linha) e assim que a tarefa seja completada, o controle é retornado para a aplicação. A figura 8.8 ilustra o exemplo da tarefa de desenhar uma linha.

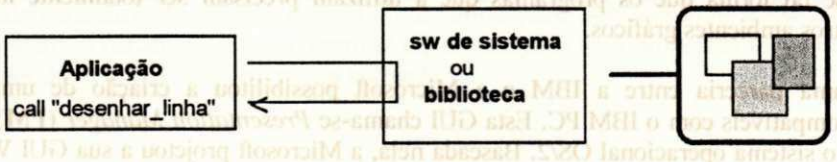


Figura 8.8. Uma saída gráfica tradicional [Wat93a]

Vale a pena citarmos que existem muitos produtos usados para melhorar a apresentação de telas com janelas em sistemas operacionais convencionais, principalmente, no mundo dos microcomputadores. Neste capítulo não abordamos produtos desta categoria, porém recomendamos aos interessados a referência [Wsz93], que fornece uma apresentação clara e abrangente sobre o assunto.

8.3.1. Histórico

A tecnologia para GUIs foi desenvolvida originalmente no laboratório **PARC** (*Palo Alto Research Center*) da Xerox, durante os anos setenta, quando foi desenvolvida a estação de trabalho ***Star*** [Sey89]. Este trabalho deu origem a todas as GUIs de hoje em dia. Em 1983, a Apple introduziu uma GUI para seu computador Lisa, que constituiu uma alternativa mais amigável de interação do que aquela apresentada pelos microcomputadores compatíveis com o IBM PC. Seu grande diferencial era justamente o "look and feel" introduzido para o grande público. A figura 8.9. ilustra uma tela do ambiente "deskTop" para o Macintosh, sucessor do Lisa.

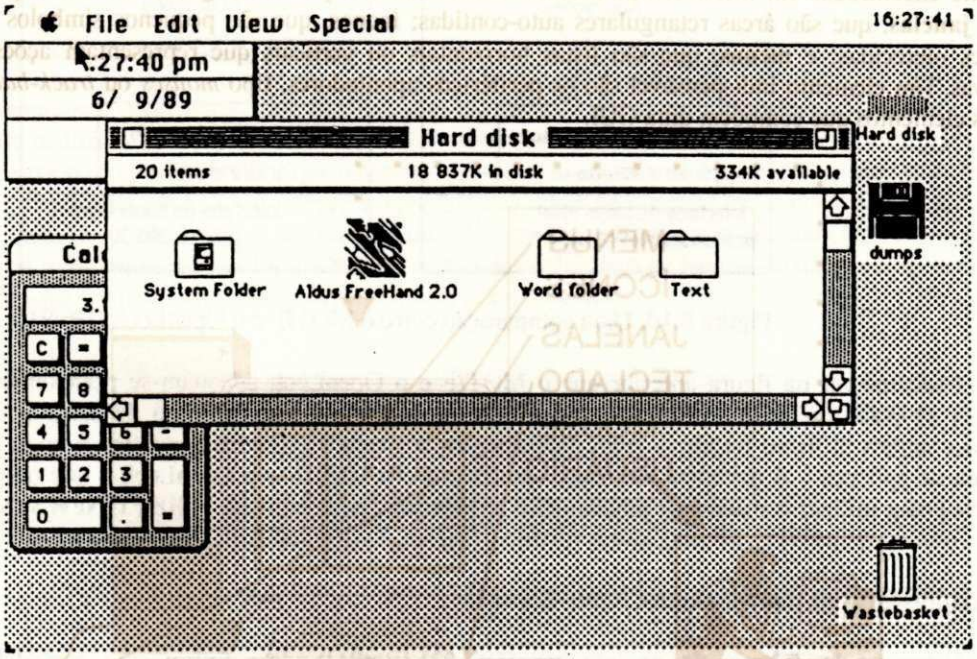


Figura 8.9. Exemplo de uma tela do "deskTop" para Macintosh [Sey89]

A Apple publicou em conjunto com sua GUI um guia de estilo, permitindo que através dele e das ferramentas de alto nível fornecidas no seu toolkit (chamada de Toolbox), os programadores pudessem criar programas compatíveis com versões futuras de seu sistema operacional. Assim, o "look and feel" das aplicações no Macintosh passaram a representar um diferencial de flexibilidade com relação aos outros produtos disponíveis no mercado. Apesar de tudo, a GUI para o Macintosh não é considerada um padrão de sistemas abertos, pois a Apple protege, a sete chaves, as funcionalidades de seu produto, disponibilizando-a somente para computadores da família Macintosh, de tal forma que os programas que a utilizam precisam ser totalmente modificados para poderem rodar em outros ambientes gráficos.

Em 1987, uma parceria entre a IBM e a Microsoft possibilitou a criação de uma GUI para os microcomputadores compatíveis com o IBM PC. Esta GUI chama-se *Presentation Manager* (**PM**), e foi projetada especificamente para o sistema operacional OS/2. Baseada nela, a Microsoft projetou a sua GUI Windows para os ambientes DOS [Cas93].

No mundo Unix, uma importante GUI padrão *de facto* é chamada de **MOTIF** ([Kob91] [Ber91] [Kla93] [Gra91]), que foi também derivada a partir do Presentation Manager. O "look and feel" destas duas GUIs é muito próximo, apesar de existirem diferenças em detalhes. Outra importante GUI para o mundo Unix, e a principal concorrente do MOTIF no mercado comercial durante muito tempo, é o **OpenLook** ([CTR92a] [Sey89] [USL92]) da Sun Microsystems (desenvolvido conjuntamente com a AT&T), e com muitas partes de sua tecnologia licenciadas diretamente da Xerox. O principal parque do OpenLook são as máquinas da própria Sun que, no entanto, desde 1993 tem liberado versões do MOTIF para substituir o Open Look em suas estações em todo o mundo.

Apesar de apresentarem aos usuários interfaces um pouco diferentes, tanto MOTIF quanto OpenLook são baseados em tecnologias muito parecida. A figura 8.10 ilustra os pontos em comum entre estas duas GUIs.

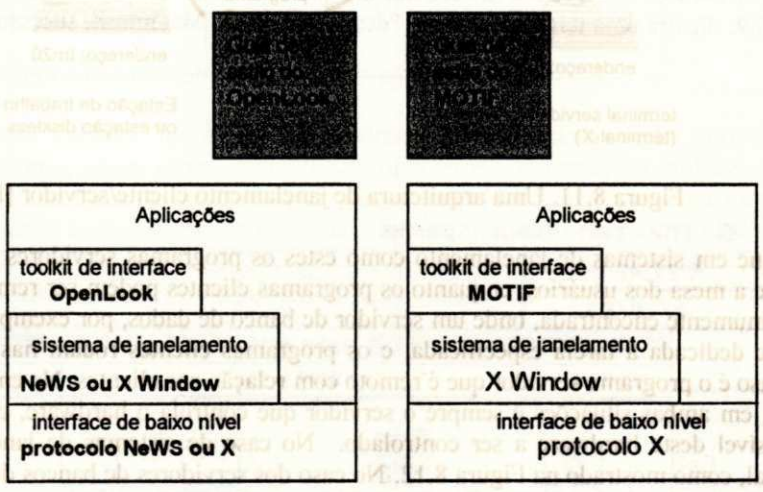


Figura 8.10. Uma comparação entre o MOTIF e o OpenLook [Gra91]

Podemos ver na figura anterior que o MOTIF e o OpenLook adequam-se perfeitamente ao modelo de referência proposto no início deste capítulo. Ambos fornecem toolkits para uso do sistema de janelamento **X Window**,e ambos utilizam terminais com displays remotos com relação ao computador onde estiver rodando o programa de aplicação. A principal diferença técnica entre o MOTIF e o OpenLook é que no caso deste último, além dele poder utilizar o sistema de janelamento **X Window**, pode também utilizar o **NeWS** (*Network-extensible Window System*) ([Cas93] [CTR92a]) desenvolvido pela Sun, e considerado obsoleto hoje em dia.

8.3.2. Arquiteturas de Janelamento Cliente/Servidor

O projeto de ***sistemas de janelamento*** ([USL92] [Ber91]) adota, tipicamente, o modelo cliente/servidor como base de sua arquitetura. Os clientes, neste caso, são os programas de aplicações que requisitam acesso a janelas de telas e permitem a entrada de dados de usuários por meio de teclados e periféricos de apontamento. Os servidores são os programas de sistemas que controlam o hardware do terminal dos usuários e atendem aos pedidos dos clientes através deste controle.

A figura 8.11 ilustra o esquema de um sistema simples com dois terminais, cada um deles com seus próprios programas servidores (sA e sB), e mais dois programas clientes (c1 e c2). O programa cliente c1 está rodando em um computador que não possui terminal (provavelmente servidores de arquivos espalhados pela rede). Ao acessar os programas servidores sA e sB na rede local, o cliente c1 poderá apresentar uma janela simultaneamente nos terminais gerenciados por estes programas servidores (trn10 e trn20). Por sua vez, o programa cliente c2 roda em uma estação de trabalho com terminal acoplado (trn20). Pela comunicação com o servidor local sB, o cliente c2 pode controlar uma janela na tela da estação de trabalho (trn20). Ao mesmo tempo, o cliente c2 poderá acessar o servidor sA pela rede, e desta forma apresentar uma janela em um terminal remoto (trn10).

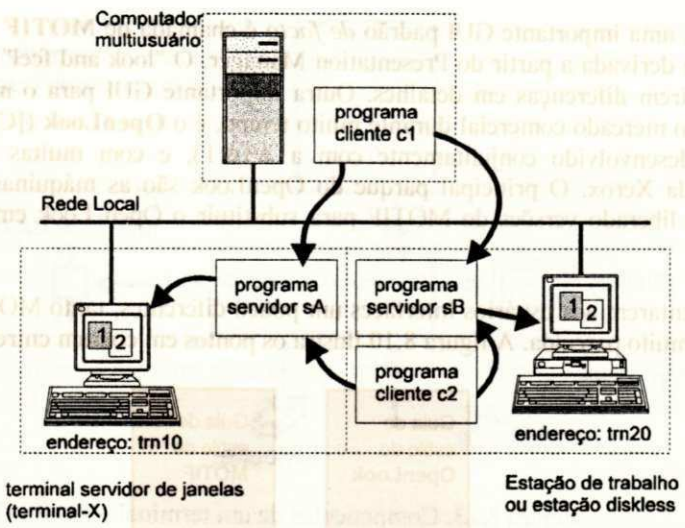


Figura 8.11. Uma arquitetura de janelamento cliente/servidor [Ber91]

Observe que em sistemas de janelamento como estes os programas servidores rodam nos equipamentos que estiverem sobre a mesa dos usuários, enquanto os programas clientes podem ser remotos. Esta é uma situação inversa daquela comumente encontrada, onde um servidor de banco de dados, por exemplo, roda em uma máquina isolada, geralmente dedicada à tarefa especificada, e os programas clientes rodam nas estações de trabalho dos usuários, e neste caso é o programa servidor que é remoto com relação aos clientes. No entanto, independentemente de sua localização, em ambas situações é sempre o servidor que controla o hardware, e desta forma deve ficar o mais próximo possível deste hardware a ser controlado. No caso de sistemas de janelamento, o servidor fica próximo do terminal, como mostrado na Figura 8.12. No caso dos servidores de bancos de dados, estes devem ficar próximos dos discos que contêm as informações relevantes para o sistema.

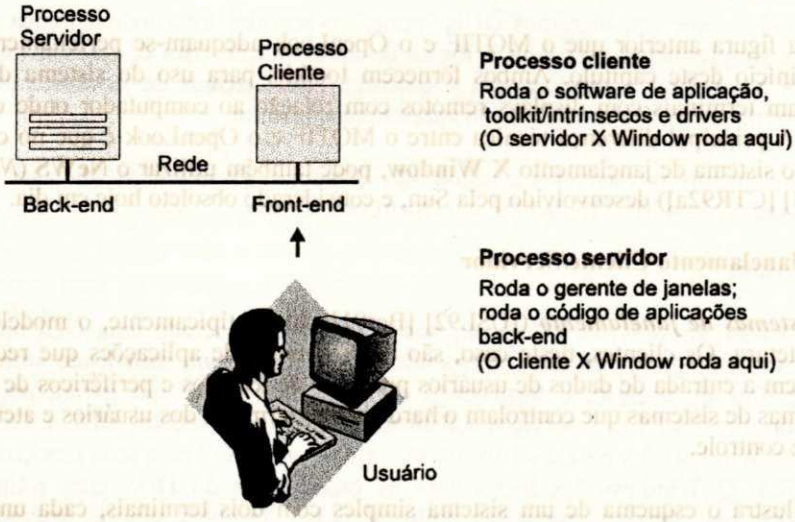


Figura 8.12. Um sistema de janelamento cliente/servidor [Dum91]

8.3.2.1. Terminais X

Nos **terminais X** ([Gra91] [Dum91] [Ber91] [CTR92a]) o servidor é o único programa que está executando. Estes terminais não conseguem rodar software de aplicação convencional, e fornecem aos usuários facilidades gráficas de altíssimo nível. Porém dependem de computadores remotos para poderem executar as aplicações que irão utilizá-los. Esta situação representa a analogia moderna para os terminais "mudos" do passado baseados em caracteres. Estes antigos terminais dependeram sempre de um computador remoto para executar as aplicações submetidas a partir deles. Na Figura 8.13 apresentamos os componentes de um terminal X. Ele contém software servidor para um sistema de janelamento em particular e memória de somente-leitura. São equipamentos que proliferaram bastante nos Estados Unidos, e dependem de uma tecnologia chamada X Window, que veremos na seção a seguir. Aqui no Brasil, seu uso está restrito a laboratórios e centros de pesquisa que necessitam de altas capacidades gráficas, como por exemplo, em projetos de CAD.

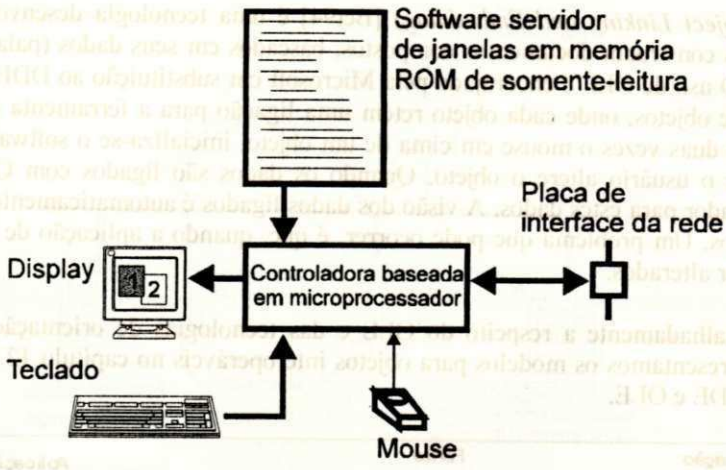


Figura 8.13. Componentes de um terminal X [O'R88]

8.4. Principais GUIs

A seguir, falamos com um pouco mais de detalhes sobre as GUIs que consideramos as mais importantes atualmente. Todas estas GUIs baseiam-se fortemente no sistema operacional que as suporta. Dos produtos que apresentamos a seguir, é bom que o leitor esteja atento ao fato de podermos classificar as GUIs em duas categorias: as *GUIs de janelamento* e as *GUIs X Window* [Dew93].

Nas GUIs de janelamento, ***parte*** da lógica das aplicações é executada adicionalmente à lógica de apresentação. O servidor nestes casos é um repositório de dados e ao mesmo tempo um gerente de dados para a aplicação. O servidor cuida de processar, a nível de lógica de aplicações, aquelas tarefas relacionadas com questões de integridade e uso de recursos de maneira intensiva.

Nas GUIs X window, que consideramos GUIs "puras", o servidor trata toda a lógica de processamento (aplicações) e os clientes simplesmente executam a lógica de apresentação (isto é, apresentam as telas em seus terminais). Nos sistemas X window, o servidor tem o controle. Nos sistemas de janelamento, o cliente tem o controle [Sey89]. Detalhes do sistema X window serão apresentados adiante.

As GUIs de janelamento apresentadas a seguir são Windows, Presentation Manager, e Macintosh. As GUIs X Window apresentadas são o MOTIF da OSF e o OpenLook da Sun.

8.4.1. Windows da Microsoft

O Windows ([Dew94] [Sey89]) da Microsoft, não pode ser considerado uma GUI na acepção pura do conceito que estamos vendo até agora, pois ela corresponde na realidade a um sofisticado ambiente operacional, incluindo a interface gráfica de usuário, e fornecendo características como um gerenciamento de memória poderoso, bibliotecas padronizadas DLL, e serviços sofisticados como os protocolos DDE e OLE, sobre os quais falamos a seguir. A versão Windows 3.x baseia-se no sistema operacional DOS para seus serviços, e portanto herda muitas de suas limitações. O Windows 3.x incrementa as capacidades do DOS com rotinas próprias para gerenciamento de memória que simulam operações multi-tarefadas. Também suporta interações dirigidas por eventos, com filas de mensagens de entrada [Dew94].

As bibliotecas **DLL** *(Dynamic Link Libraries)* [Dew94] permitem que rotinas codificadas possam ser modularizadas e melhoradas em termos de desempenho e de facilidade para manutenção de uma aplicação. O código pode ser empacotado de acordo com as considerações a respeito das funcionalidades desejadas e de considerações sobre desempenho. Esta facilidade é também suportada pela GUI Presentation Manager.

A facilidade **DDE** (*Dynamic Data Exchange*) [Dew94] fornece a possibilidade de troca automática de informações entre aplicações. DDE pode ser usado para ligar um grupo de chamadas entre dois programas como, por exemplo, a planilha eletrônica Excel e o processador de textos Word, ambos da Microsoft. Se parte da planilha, ou um gráfico da planilha, for carregado em um documento do editor de textos, quando estes dados forem alterados diretamente na planilha, a área correspondente ao documento do editor de textos será alterada automaticamente.

O **OLE** (*Object Linking and Embedding*) [Bet94] é uma tecnologia desenvolvida pela Microsoft para permitir que usuários construam documentos compostos, baseados em seus dados (palavras, números, gráficos) e não nas aplicações. O uso de OLE é encorajado pela Microsoft em substituição ao DDE. Um documento é tratado como uma coleção de objetos, onde cada objeto retém uma ligação para a ferramenta que tenha sido usada para criá-lo. Pressionando duas vezes o mouse em cima de um objeto, inicializa-se o software de aplicação original, o que irá permitir que o usuário altere o objeto. Quando os dados são ligados com OLE, o Windows cuida de armazenar um apontador para estes dados. A visão dos dados ligados é automaticamente alterada quando os dados originais são alterados. Um problema que pode ocorrer, é que, quando a aplicação de ligação estiver off-line, os dados não poderão ser alterados.

Falamos detalhadamente a respeito do OLE e das tecnologias de orientação a objetos propostas pela Microsoft quando apresentamos os modelos para objetos interoperáveis no capítulo 12. Na figura 8.14 ilustramos as diferenças entre DDE e OLE.

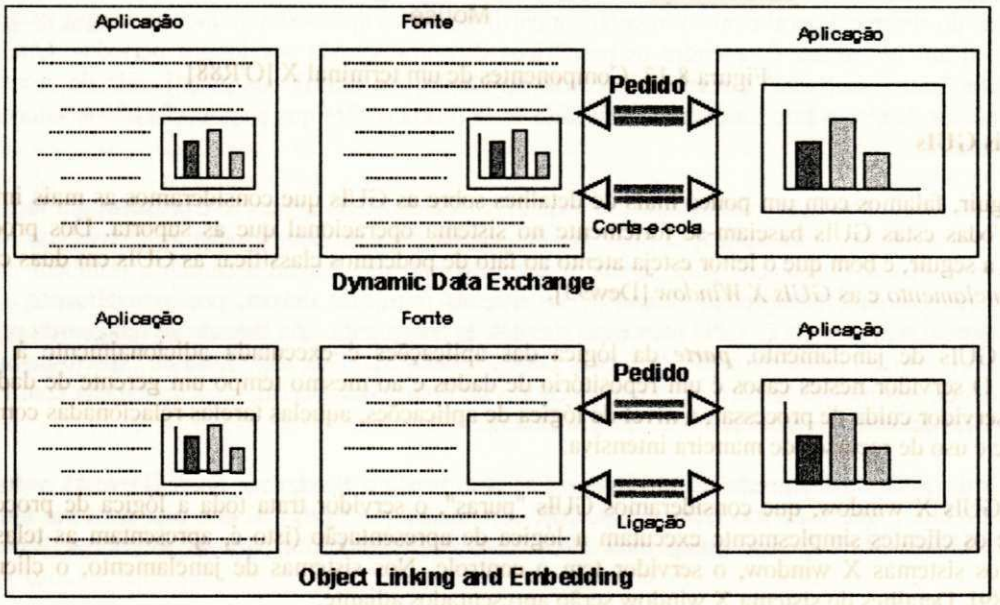


Figura 8.14. Uma comparação entre DDE e OLE [Dew94]

O nosso interesse ao falarmos, explicitamente, sobre o Windows diz respeito ao fato do mesmo ter 60 milhões de cópias vendidas em todo o mundo (dezembro de 1994 [Bet94]) e ser considerado como o front-end padrão de indústria para as estações clientes das aplicações do modelo cliente/servidor, dos ambientes de computação distribuída.

A versão mais recente do Windows chama-se Windows 95, a qual fornece um "look and feel" similar aquele do Macintosh, e inclui nela mesma muitas das funções básicas de sistemas opracionais antes fornecidas pelo próprio DOS.

A Microsoft pretende transformar o Windows em uma interface comum para as redes corporativas. Para permitir que usuários acessem informações e serviços espalhados por plataformas heterogêneas, incluindo mainframes e midranges, a Microsoft disponibiliza a infra-estrutura **WOSA** (*Windows Open Services Architecture*) [Dew94], que consiste de um conjunto de APIs proprietárias que podem ser usadas para integrar Windows e aplicações baseadas em janelas nos sistemas corporativos maiores. WOSA inclui em suas propostas o **ODBC** (*Open Database Connectivity*) e **MAPI** (*Message Application Programming Interface*).

8.4.2. Presentation Manager

O *Presentation Manager* (**PM**) [Sey89] é a interface gráfica do sistema operacional OS/2 da IBM, e possui suporte interno para as características como multi-tarefamento, bibliotecas de ligação dinâmica, e "named pipes". O PM não pode ser suportado por outros sistemas operacionais, como DOS, Windows ou Macintosh.

Para que aplicações possam rodar sob PM, elas são registradas e assinaladas a um controlador de blocos tipo "âncora" usado para ligar classes de janelas. As classes e objetos em PM relacionam-se às janelas e seus tipos. Cada uma destas classes de janelas recebe um nome que as identifica unicamente. As aplicações e seus esquemas de janelamento são construídos baseados nestas classes de janelas.

Como o sistema operacional OS/2 é *multi-threaded* a partir de sua versão 2.x [[Pen93], cada aplicação tem uma thread primária, e pode também ter threads adicionais, chamadas de threads secundárias. O sistema para gerenciamento de filas de mensagens em OS/2 recebe entradas (do teclado ou de um mouse) de usuários contendo pedidos por serviços, e as encaminha para um motor de inferência ("engine") primário da aplicação pertinente. O PM cuida de traduzir estas entradas em mensagens dele mesmo, que são empilhadas e processadas, uma de cada vez, através de um *roteador de entrada* que também faz parte do ambiente PM.

O roteador de entrada direciona as mensagens para as filas de mensagens das aplicações apropriadas. Quaisquer tratamentos ou roteamentos subseqüentes destas mensagens são tratados pela própria aplicação. Como a fila de mensagens do sistema é geralmente alimentada a partir de várias aplicações ao mesmo tempo, as aplicações individualmente ficam impedidas de prender os periféricos de entrada/saída por longos períodos. Quando uma mensagem do tipo "quit" (cair fora) é recebida por uma aplicação, ela destrói sua própria fila de mensagens, e termina o controlador de blocos tipo "âncora", notificando desta forma ao PM que a aplicação não se encontra mais ativa para esta sessão.

As threads secundárias são inicializadas por uma thread primária da aplicação e possuem um laço baseado em mensagens. Por exemplo, uma aplicação pode ser desenhada para processar todas as mensagens na thread primária, ou para re-rotear mensagens para serem processadas em uma thread secundária (também conhecida como thread de retaguarda ("background thread")). As threads primárias devem, preferencialmente, rotear as tarefas que consomem mais tempo de CPU para estas threads de retaguarda. As threads de retaguarda processam as tarefas enquanto a thread primária cuida de aceitar outras mensagens. Este algoritmo de "aperto-de-mão" traz uma nova camada de complexidade para os desenvolvedores de aplicações, porém, suas vantagens com relação à velocidade de processamento o tornam particularmente atraente.

A partir de 1995 a IBM pretende substituir o PM pelo produto *Workplace Shell* [Dew94], voltado para arquiteturas de 32 bits e que roda sob DOS, OS/2, AIX, e também no Talligent, que está para ser lançado no mercado. O Workplace Shell é visto como uma alternativa multiplataforma para as interfaces Windows da Microsoft.

8.4.3. Macintosh

O Macintosh ([Cas93] [Sey89] [Gra91]) da Apple foi o introdutor do mouse e das interfaces com ícones para o grande público. Sua interface foi desenvolvida pelo laboratório PARC (Palo Alto Research Center) da Xerox, especificamente voltada para crianças. A Apple gastou toda a década de 80 para convencer os empresários que esta interface não era brincadeira de criança.

A interface Macintosh usa um modelo chamado desktop. Este modelo é constituído de formulários ("folders") que podem conter outros formulários ou outros tipos de objetos, como programas ou arquivos de dados. Os objetos são representados por ícones.

Os usuários selecionam um objeto e, a seguir, podem selecionar uma ação a partir de um menu especificado por um comando, ou executando o objeto diretamente, por meio do pressionamento duplo do mouse. Este pressionamento duplo abre os objetos selecionados. Se o objeto for um formulário, seu conteúdo é apresentado na tela. Se for uma aplicação, o programa é trazido para memória e assume o controle. Se o objeto for um arquivo de dados, o próprio programa que tenha criado este arquivo é trazido para memória e assume o controle, abrindo o arquivo selecionado, de tal forma que o usuário pode começar a trabalhar com ele quase que imediatamente. Assim, esta manipulação direta da interface libera o usuário da necessidade de ter que se lembrar que programa criou qual arquivo de dados.

8.4.4 MOTIF

O MOTIF da OSF ([Kob91] [Ber91] [Kla93] [Gra91]) é implementado usando uma API única para todas as plataformas que o suportam. É um produto baseado na tecnologia DECWindows, e apresenta um "look-and-feel" similar ao PM. O MOTIF é baseado nos padrões da OSF e é considerado uma tecnologia totalmente aberta. A definição de recursos é armazenada separadamente do código de aplicação, o que simplifica o seu tratamento. O MOTIF é quase completamente independente de características particulares de sistemas operacionais e protocolos de rede. O ambiente de desenvolvimento do MOTIF inclui as seguintes características:

- *Toolkit X para interface de usuários*: contém os objetos gráficos (widgets e intrínsecos) usados pelo MOTIF;
- *Linguagem de interface de usuários:* é usada para descrever os aspectos visuais da GUI MOTIF em uma aplicação, como por exemplo, menus, telas, e botões. Os desenvolvedores podem criar um arquivo texto que contém a descrição de cada objeto com suas funções relacionadas. Este arquivo texto é então compilado pela linguagem de interface de usuário, e é carregado na hora da execução do programa; e,
- *Gerente de janelas:* permite que janelas sejam movimentadas na tela, tenham seu tamanho redefinido, ou sejam reduzidas em ícones. É este gerente que apresenta a aparência em três dimensões para o MOTIF.

O MOTIF suporta ainda os padrões recomendados pelo guia de estilo XPG3 do consórcio X/Open para linguagens nativas, e está disponibilizado na maioria das plataformas de hardware e software que oferecem versões do Unix ([UI89] [CTR92a]).

8.4.5. OpenLook

O OpenLook ([SUN92a] [CTR92a] [Sey89] [USL92]) foi desenvolvido pela Sun e pela AT&T, sendo muito popular em suas respectivas plataformas. O gerente de janelas OpenLook é necessário para a comunicação entre clientes nestas plataformas. O "look-and-feel" do OpenLook é bem diferente do "look-and-feel" das GUIs derivadas do Macintosh, como o PM, o Windows 3.x e o próprio MOTIF. De maneira similar ao MOTIF, quase não existem dependências de sistema operacional ou de protocolos na rede. Três APIs estão disponíveis para o desenvolvimento de aplicações aderentes ao OpenLook.:

- *O ambiente de desenvolvimento NeWs da Sun*: fornece APIs para plataformas Sun com um interpretador de emulação postscript, modificado para suportar sistemas de janelamento. Este ambiente consiste de clientes e servidores, que interagem para gerar a GUI OpenLook;
- *A API XT+ da AT&T*: contém os objetos gráficos do OpenLook e é usada em plataformas AT&T; e,
- *A API XView:* usada nas plataformas SPARC da Sun, em sistemas VAX da Digital, em Intel 80x86, e em Motorola 680x0. Estas plataformas implementam a API OpenLook no topo do nível XLib do X Window. (Só para podermos comparar, o MOTIF implementa a sua API no toolkit X construído no nível de intrínsecos X11 do X Window, que é um nível mais elevado que o de Xlib, como mostramos adiante).

8.5. O Sistema X Window

O MIT (Massachussetts Institute of Technology) começou a desenvolver o ***sistema X Window*** ([O'R88] [Gra91] [Dum91] [Dew93] [Cas93] [CTR92a] [Ber91]) em 1984, e seu nome deve-se simplesmente ao fato dele ser um projeto melhorado de um sistema anterior batizado de sistema W (assim como a linguagem C foi desenvolvida a partir de uma linguagem de programação mais antiga, chamada de linguagem B).

Em capítulos anteriores, já falamos do projeto Athena do MIT. Este foi um dos projetos que mais influenciaram os rumos da computação distribuída. Porém, dos muitos desenvolvimentos associados com o projeto Athena, o sistema X Window é provavelmente o seu rebento mais famoso, e o mais largamente utilizado.

A manutenção e aprimoramento do sistema X Window é suportado por um consórcio chamado de X Window Consortium, que compreende um grupo formado pelos mais importantes fornecedores de computadores, incluindo a Digital, IBM e a Hewlett-Packard. O sistema X Window pode ser obtido em código fonte em linguagem C por um custo praticamente zero, e pode também ser encontrado em formato binário para diversos ambientes operacionais, o que o tornou um padrão *de facto* indiscutível. O sistema X Window é endossado como um padrão de facto por praticamente todos os órgãos de padronização relacionados ao sistema operacional Unix. A OSF especifica o X Window como um de seus padrões de "nível zero" [OSF90a]; e o X/Open dedica todo um volume de seu guia de portabilidade XPG ao mesmo [UI89].

Em termos de padrões de jure, o X Window está sendo formalizado pelo IEEE, que já liberou padrões formais para o X Window Toolkit API (padrão IEEE 1201.1) e para o X Window Recommended Practice (padrão IEEE 1201.2), ambos publicados a partir de novembro de 1992 [Dew93].

8.5.1. A Arquitetura do X Window

Diferentemente dos sistemas que geram saídas gráficas convencionais, como aquele apresentado no início da seção 8.3 e ilustrado com um exemplo na Figura 8.8 anterior, no sistema X Window o software de sistema é substituído por uma aplicação especial chamada de **servidor X** [O'R88]. É esta aplicação que tem controle total sobre as telas de apresentação (ou telas de display). Uma aplicação que desejar produzir saídas gráficas no terminal deve instruir o servidor X a executar uma tarefa específica, pelo envio de uma mensagem com informações que descrevem a tarefa requisitada. O envio desta mensagem para um servidor X devolve imediatamente o controle para a aplicação, e pode ou não gerar uma resposta do servidor.

Os diferentes tipos de mensagens são referenciadas em conjunto como o **protocolo X** [O'R88]. Uma mensagem pode, por exemplo, ser usada para requisitar que se desenhe uma linha, enquanto outra mensagem pode requisitar que se desenhe um círculo, e outra pode ainda requisitar que um texto seja impresso. Qualquer aplicação que possa exibir as saídas gráficas por meio de troca de mensagens via o protocolo X é chamada de um cliente X. A figura 8.15 mostra o esquema usado pelo X Window para gerar apresentações nas telas dos terminais de vídeo.

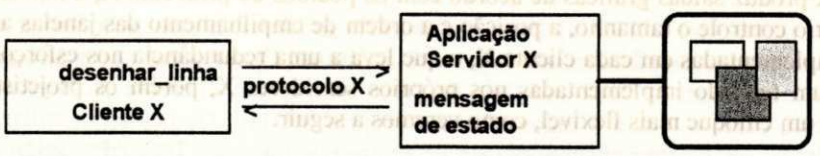


Figura 8.15. A saída gráfica do sistema X Window [Dum91]

Em resposta ao pedido, o servidor X poderá enviar de volta para o cliente X mensagens especiais, como por exemplo mensagens de estado ou mensagens de erro. Estas mensagens especiais também fazem parte do protocolo X. Os clientes X geralmente criam janelas como parte de suas saídas. É possível que um único cliente X crie e utilize várias janelas na tela do servidor simultaneamente. Note que o servidor X pode tratar saídas gráficas para múltiplos clientes X concorrentemente, e só compreende pedidos do protocolo X.

O X Window é um sistema independente de hardware e independente de sistema operacional. Esta independência significa que nenhuma das máquinas na rede precise ser do mesmo fornecedor, ou deva rodar o mesmo sistema operacional. Toda a comunicação entre clientes e servidores em X é feita por meio de um sistema sofisticado de troca de mensagens chamado de protocolo X. Naturalmente, um programa em formato binário não pode ser copiado para um tipo diferente de máquina na rede e rodar imediatamente. Ele precisa ser recompilado para se adequar à arquitetura da nova máquina e ao seu respectivo sistema operacional.

O sistema X Window adequa-se à arquitetura cliente/servidor, onde um servidor X corresponde a uma aplicação gráfica especial do sistema X Window que controla as telas em um terminal de vídeo, permitindo que se desenhem janelas, texto, linhas, figuras, círculos, polígonos, dentre outros, de acordo com os pedidos na forma de mensagens enviados por uma aplicação. Um servidor X pode tratar uma tela simultaneamente para várias aplicações, onde cada aplicação irá tipicamente apresentar informações com uma ou mais janelas na tela.

O cliente X corresponde a um programa de aplicação do sistema X Window (como por exemplo, uma planilha eletrônica ou um programa gráfico) que se comunica com um servidor X. Um sistema de troca de mensagens muito bem definido permite que clientes e servidores comuniquem-se pela rede: este sistema constitui a suite de protocolos X [Kob91].

8.5.2. Um Sistema Orientado a Eventos

O X Window é um sistema orientado a eventos [God93a]. Isto é, os clientes X ficam suspensos até que uma ação ocorra no servidor X no qual eles tenham algum tipo de interesse. Os clientes X são reinicializados pelo servidor X pelo envio de uma mensagem especial do protocolo X. Mensagens de eventos incluem aquelas que cuidam de exigir que um cliente X redesenhe suas janelas (por exemplo, se parte de suas janelas forem sobrepostas pelo movimento de outras janelas), ou que informam ao cliente X que o tamanho de uma janela foi alterado ou que determinada tecla foi pressionada. Um cliente X processa estas mensagens, produzindo qualquer tipo de saída que seja necessária, e depois retorna para um estado suspenso até que outra mensagem seja recebida.

Este esquema é exatamente o contrário do que ocorre nas aplicações tradicionais, onde estas aplicações são *orientadas-a-procedimentos*, e são escritas para assumir uma função ativa no interrelacionamento entre o usuário e o programa de aplicação. Geralmente, o programa irá guiar o usuário pela execução da tarefa corrente, forçando-o em direção a um conjunto restrito de opções pre-definidas. Uma aplicação de pedido de compras bascada em menus é um bom exemplo deste tipo de programa orientado-a-procedimento.

As aplicações orientadas a eventos tomam uma função mais passiva, pois somente respondem aos usuários ou ao sistema em períodos imprevistos. Estes tipos de aplicações podem fornecer uma infra-estrutura mais flexível para os usuários. Geralmente, não existem procedimentos pré-definidos, o que fornece muitas possibilidades para que se complete uma tarefa. Um usuário estará livre para utilizar quaisquer ferramentas que a aplicação venha a fornecer, e de tal forma que o resultado final será obtido da maneira que lhe for mais conveniente. Um programa gráfico para gerar gráficos ou cronogramas de projetos é um bom exemplo de uma aplicação orientada a eventos.

8.5.2.1. O Gerente de Janelas - Window Manager

O servidor X produz saídas gráficas de acordo com os pedidos do protocolo X, e não fornece funções que permitam que o usuário controle o tamanho, a posição e a ordem de empilhamento das janelas apresentadas. Estas funções devem ser implementadas em cada cliente X, o que leva a uma redundância nos esforços de programação. Estas funções deveriam ter sido implementadas nos próprios servidores X, porém os projetistas dos sistema X Window optaram por um enfoque mais flexível, como veremos a seguir.

Um cliente X especial roda (local ou remotamente) para cada servidor X, e é chamado de **gerente de janelas** (*"window manager"*) [O'R88]. Ele recebe privilégios especiais e recebe a permissão para supervisionar todas as janelas sendo apresentadas por um determinado servidor X. O gerente de janelas cuida de colocar algum tipo de decoração de janela ao redor de cada janela de um cliente X, incluindo botões de redefinição do tamanho de janelas, ou botões de movimentação, ou ainda, uma barra de título. A seguir, fica por conta do gerente de janelas cuidar de redefinições do tamanho de janelas, movimentações de janelas, e de rearranjar uma janela de acordo com os desejos do usuário, por meio de pressionamentos do mouse sobre decorações da janela, ou a partir de um menu de seleção, também fornecido pelo gerente de janelas. Os objetos gerenciados em uma janela estão mostrados na Figura 8.16.

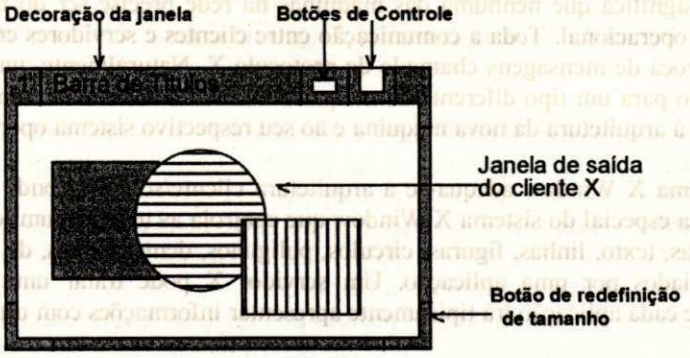


Figura 8.16. Objetos gerenciados pelo Gerente de janelas [UI90a]

Atualmente, existem vários gerentes de janelas para o sistema X Window, e os mais famosos são justamente aqueles fornecidos pelo toolkit do MOTIF [Kob91] da OSF e pelo toolkit do OpenLook [USL92] da Sun. Estes toolkits são bibliotecas de programas disponíveis para os desenvolvedores de clientes X, e são responsáveis pela definição do "look and feel" das aplicações, que poderão apresentar uma aparência MOTIF ou uma aparência OpenLook, por exemplo. Observe que o toolkit sempre contém o módulo gerente de janelas para o sistema envolvido.

Graças ao enfoque adotado, um gerente de janelas pode ser fechado, e a seguir outro pode ser inicializado imediatamente, sem afetar nenhum dos clientes X sendo apresentados em uma tela. Neste caso, as decorações da janela antiga desaparecem da tela e são substituídas por novas decorações criadas pelo novo gerente de janelas.

8.5.3. O X Window e o Modelo de Referência

O sistema X Window foi desenvolvido originalmente para fornecer as funcionalidades requsitadas nos diversos níveis do modelo de referência para interfaces de usuários proposto anterioremente. No primeiro nível, foi definida uma suite de protocolos usada para viabilizar a comunicação entre clientes e servidores. No segundo nível, vem a biblioteca Xlib, que compreende um conjunto de ferramentas para a definição e caracterização das janelas, e para informar aos programas de aplicações a respeito de eventos como o pressionamento do teclado ou os movimentos do mouse. Observe porém que a este nível não ficam definidas características relacionadas com o "look and feel" para as aplicações. Assim, no terceiro nível do modelo de referência são fornecidos toolkits que irão restringir a flexibilidade da Xlib a um nível desejado de "look and feel". Usando a terminologia própria do sistema X Window, estes toolkits consistem de ***widgets*** [Kob91] (uma palavra sem tradução literal, que consiste da junção de "window" com "gadget" [um mecanismo bem projetado]) e ***intrínsecos***. Em termos de operacionalidade, um widget apresenta algo na tela, como por exemplo, um botão de confirmação, enquanto que um intrínseco controla a comunicação entre os widgets e a biblioteca de baixo nível.

Existem vários toolkits disponíveis que podem ser usados acima de Xlib, e muitos deles foram desenvolvidos no próprio MIT. A figura 8.17 mostra que estes toolkits abrangem interfaces de usuários completas, com um "look and feel" específico e guias de estilo próprios. Exemplos são os toolkits do próprio MIT, chamado de **Xt**, ou o toolkit **Andrew**, do ambiente operacional de mesmo nome desenvolvido pela Universidade Carnegie Mellon.

Figura 8.17. Guias de estilo e widgets no sistema X Window [Gra91]

8.5.3.1. Xlib

Para que um cliente X esteja apto a comunicar-se com um servidor X, ele precisa gerar pedidos no formato do protocolo X para transmiti-los ao servidor. Construir estes pedidos pode ser muito trabalhoso, e portanto uma biblioteca chamada **Xlib** ([O'R88] [Dum91] [Ber91] [Gra91]) foi criada para aliviar a carga de trabalho extra. Xlib é geralmente a interface de mais baixo nível que um cliente X utiliza para comunicar-se com um servidor X. Ela constitui-se de um conjunto de subrotinas em linguagem C que são, na sua maioria, um mapeamento do tipo um-para-um entre uma função em C e um pedido do protocolo X, apesar de algumas funções em Xlib serem capazes de gerar múltiplos pedidos do protocolo X. Por exemplo, se um cliente X utiliza a função *XDrawLine*, ele irá chamar o código apropriado dentro da biblioteca Xlib, que irá gerar um pedido X do tipo *PolySegment*, o qual será transmitido para o servidor X, como mostrado na Figura 8.18.

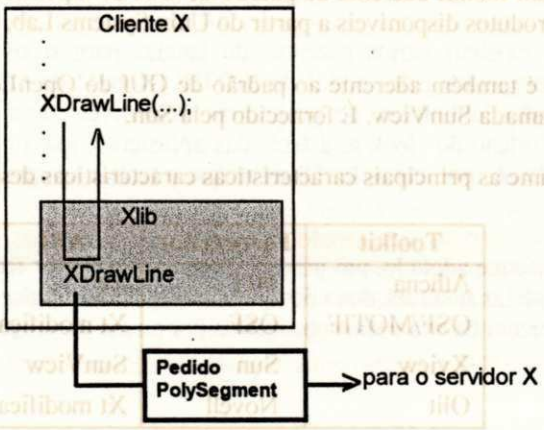


Figura 8.18. Exemplo de uma chamada a uma função em Xlib [O'R88]

Note que a biblioteca Xlib não impõe qualquer tipo de "look and feel" a um cliente X. Ela consiste meramente de subrotinas para criar janelas, traçar linhas, imprimir determinado texto, etc. A aparência da aplicação é, normalmente, determinada por outra biblioteca de programa: o toolkit. Observe porém, que apesar deste fato, clientes X podem ser escritos utilizando-se somente a biblioteca Xlib, sem a biblioteca de toolkits, o que é certamente bem mais trabalhoso para os programadores.

8.5.3.2. Toolkits

Como acabamos de ver, a biblioteca Xlib é muito rudimentar em termos de suas capacidades. Portanto, em uma camada acima dela, fica o toolkit de ferramentas ([Dum91] [Ber91] [Wat93a]). Os toolkits, geralmente, fornecem rotinas que permitem que sejam construídos menus, botões pressionáveis, controles de deslizamento e outros. Como é função do toolkit gerar estes componentes básicos em janelas para os clientes X, eles são os responsáveis pelo "look and feel" real das aplicações.

Uma única função do toolkit pode chamar várias funções da Xlib que por sua vez pode criar múltiplos pedidos do protocolo X. Por exemplo, se um cliente X deseja efetuar uma apresentação do tipo de uma janela ("pop up"), por meio de um toolkit específico, ele pode chamar a rotina ***XtPopup*** para executar esta função. XtPopup, por sua vez, irá executar várias chamadas Xlib que poderá gerar múltiplos pedidos do protocolo X. A figura 8.19 ilustra o conceito.

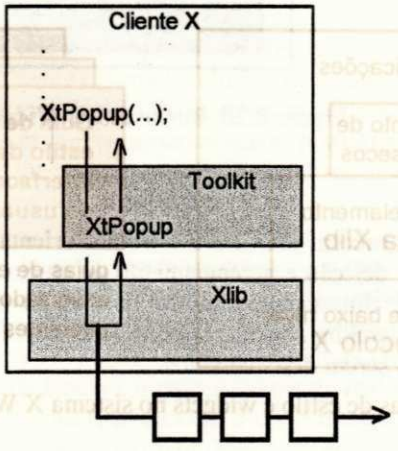


Figura 8.19. A chamada de uma função toolkit [O'R88]

Os toolkits mais conhecidos são [Dum91]:

- **Toolkit Athena**: é o toolkit Xt já citado anteriormente. Consiste de funções consideradas muito simples, e foi criado no próprio MIT;
- **Toolkit OSF/MOTIF**: é o toolkit fornecido pela OSF e fornece um visual em terceira dimensão;
- **Toolkit Olit**: é um toolkit aderente ao padrão de GUI do OpenLook, e também fornece aparência em terceira dimensão. São produtos disponíveis a partir do Unix Systems Lab, agora sob o controle da Novell; e,
- **Toolkit XView**: é também aderente ao padrão de GUI do OpenLook, porém com uma diferente interface de programação chamada SunView. É fornecido pela Sun.

A tabela 8.1 resume as principais características características destes toolkits.

Toolkit	Fornecedor	API	Interface
Athena	MIT	Xt	rudimentar
OSF/MOTIF	OSF	Xt modificada	OSF/MOTIF
Xview	Sun	SunView	OpenLook
Olit	Novell	Xt modificada	OpenLook

Tabela 8.1. Resumo das características de toolkits disponíveis no mercado [Dum91]

8.5.3.3. Intrínsecos e Widgets

Os toolkits são geralmente divididos em duas partes: os intrínsecos e o conjunto de widgets [Ber91].

O conjunto de widgets

Widgets [Kob91] são objetos abstratos de dados como botões, barras de rolagem e outros deste tipo. Um cliente X pode ser facilmente construído a partir de vários widgets. O cliente X não possui controle direto sobre a aparência real de um determinado widget, somente de sua forma geral, tamanho ou conteúdo. A aparência (o "look and feel") é determinada pelo toolkit completo (conjunto de widgets mais intrínsecos). Um conjunto de widgets utiliza tanto chamadas a funções dos intrínsecos quanto da biblioteca Xlib. A figura 8.20 ilustra os conceitos.

Figura 8.20. Intrínsecos e Widgets [Kob91]

Os Intrínsecos

Os intrínsecos [O'R88] fornecem uma infra-estrutura orientada a objetos da qual o conjunto de widgets depende. Eles cuidam da criação, deleção e gerenciamento de objetos, assim como das mensagens de eventos. É possível para um cliente X chamar diretamente uma das funções dos intrínsecos, ou uma das funções do conjunto de widgets (e claro, da Xlib). Os toolkits Athena, OSF/MOTIF e Olit consistem de um conjunto de widgets e do intrínseco Xt - um dos primeiros a serem desenvolvidos.

X11

O MIT em associação com o Consórcio X libera as fitas de distribuição MIT X, que contêm as bibliotecas Xlib e Xt, bem como amostras de clientes X e de um servidor X [Dum91]. É a partir destas fitas que todos os outros toolkits e produtos da família X são baseados. A revisão corrente destas fitas de distribuição constituem a versão 11 do sistema X Window, conhecida como versão **X11 R6.**

8.6. Conclusões

Com a proliferação de microcomputadores baratos e poderosos, e de serviços sofisticados nas redes de computadores, as GUIs estão tornando-se cada vez mais populares e indispensáveis no dia a dia da maioria dos usuários dos ambientes modernos de computação. Antigamente, com interfcae sgráficas orientadas a caractere, cada programador era responsável por criar sua própria biblioteca de apoio, onde a aparência externa e o comportamento das aplicações era bem variado, o que de certa forma confundia os usuários finais. Diferentes plataformas costumavam fornecer diferentes sistemas de janelamento, o que tornava a migração de aplicações um problema a mais. Portanto, foram criadas várias ferramentas para permitir a portabilidade facilitada de aplicações, reduzindo-se a necessidade de retreinamento de usuários. De maneira geral, uma GUI apresenta duas características principais com relação as antigas interfaces baseadas em caracteres [Eng94c]:

- A primeira destas características são os *símbolos gráficos*, que, se desenhados com cuidado, são mais facilmente reconhecidos e memorizados. O cérebro humano simplesmente possui uma maior habilidade para absorver informações gráficas do que para processar textos baseados em caracteres;
- A segunda característica diz respeito ao fato de GUIs permitirem que o próprio usuário estabeleça parâmetros e execute comandos em uma *manipulação direta* dos objetos na tela do terminal. A chamada de funções por meio desta interação física direta, com uma retroalimentação instantânea, reduz a curva de aprendizado e fornece ao usuário um maior sentimento de controle sobre o seu computador.

As manipulações diretas são controladas por periféricos especiais, como por exemplo, um mouse. Um usuário pode especificar uma localização na tela pela movimentação direta do mouse para o ponto desejado, com um simples movimento de sua mão. Um apontador visivel na tela traça constantemente a posição do mouse. Dependendo desta posição, diferentes ações podem ser iniciadas.

Assim, as GUIs representam hoje em dia uma das maiores esperanças da indústria de computadores para finalmente fornecer aos seus usuários uma facilidade que irá garantir a tão prometida melhoria de produtividade. As GUIs fornecem uma interação mais natural com os computadores, pois permitem que os usuários manipulem objetos no lugar de palavras ou números [Ber91].

Como dissemos há pouco, um dos principais benefícios do uso de GUIs é a redução da curva de aprendizado, e conseqüentemente, uma redução do tempo de treinamento. As antigas interfaces baseadas em caracteres foram desenvolvidas sem levar em consideração aspectos de padronização, o que significa dizer que as habilidades adquiridas no uso de uma determinada aplicação não eram aproveitadas nas outras. Nas aplicações desenvolvidas com o uso de GUIs, o aspecto de "olhar e sentir" ("look and feel") consistente é fornecido, pois a padronização das mesmas procura estender a uniformidade entre as diversas aplicações. Desta forma, podemos considerar que o problema da migração de habilidades fica parcialmente resolvido com o uso de GUIs consistentes. Esta é a grande argumentação de produtividade, pois a padronização de operações de uma aplicação para outra economiza tempo tanto na curva de aprendizado quanto no uso das aplicações [Eng94c].

Portanto, as GUIs são cruciais para que os computadores cheguem ao dia a dia de pessoas que ainda não se sentem muito a vontade com eles. Um ambiente orientado a linhas de comandos exige que os usuários aprendam como operar o computador antes de poderem utilizá-lo de maneira produtiva, e muitos usuários resistem a este esforço extra. As GUIs, por sua vez, permitem que os usuários sejam produtivos quase que imediatamente, sem a necessidade deles conhecerem as peculiaridades do computador (ou de seu sistema operacional, melhor dizendo) com antecedência [Bry93].

Em termos de padronização das GUIs, várias tecnologias estão competindo para uma aceitação como um padrão recomendado por organismos internacionais. Vimos que as GUIs podem ser bem definidas em termos de um modelo de referência em camadas, onde a camada mais próxima do sistema operacional é representada por um padrão aceito como *de Jure*, que é o sistema X Window. Nos níveis mais elevados do modelo, ficam definidas características como funções para a criação de janelas, menus e caixas de diálogos. Nestes níveis existem padrões de facto, e dois particularmente destacam-se: o MOTIF da OSF e o OpenLook da Sun. Apesar de ambos rodarem sob X Window, e serem similares em vários aspectos, existem diferenças em detalhes ([CTR92a] [Gra91]).

8.6.1. Tendências

Vale a pena falarmos brevemente a respeito de algumas tecnologias que estão por surgir baseadas nos conceitos inerentes às GUIs. Falamos a seguir de computação colaborativa, multimídia, e vídeo com janelas.

Computação Colaborativa

A comunicação com vídeofone, a base para a *computação colaborativa* [HL93], irá tornar-se um modo de comunicação tão comum quanto o telefone de hoje em dia. A comunicação visual por meio de telas de computadores irá abrir um novo mundo de serviços de comunicações jamais vistos até então. As características de identificação rápida e operação imediata de funções na comunicação em tempo real irá forçar a busca por soluções antes impensadas em termos de interfaces gráficas para usuários. A qualidade do projeto visual destas novas aplicações será crucial para facilitar o uso de dispositivos tão complexos quanto os equipamentos de comunicação dinâmica em vídeo, e ao vivo.

Detalhes do projeto visual, como por exemplo, o tamanho de telas de vídeo, o tamanho das áreas de controle, a facilidade de identificação de funções, devem ser pensadas de maneira a não confundirem os usuários. O usuário deve estar apto a acessar imediatamente as inúmeras funções disponíveis, como por exemplo, o controle de claridade ou de volume. Estes tipos de controles devem ser projetados de maneira a serem prontamente reconhecidos e manipulados de maneira simples e intuitiva.

O vídeofone irá alterar também nossos hábitos cotidianos de comunicação. A necessidade de sabermos se estamos apresentáveis ou não para determinadas situações torna-se um problema real quando falamos de facilidades como estas. Poderíamos, por exemplo, incluir no projeto de uma interface gráfica deste tipo uma janela de apresentação pessoal, que permitiria que o usuário verificasse se ele está ou não apresentável, antes de iniciar uma sessão de comunicação visual. Na prática, poder-se-ia programar a câmera do equipamento local para um zoom especial na imagem do usuário, que retornaria para o mesmo antes ser passada adiante.

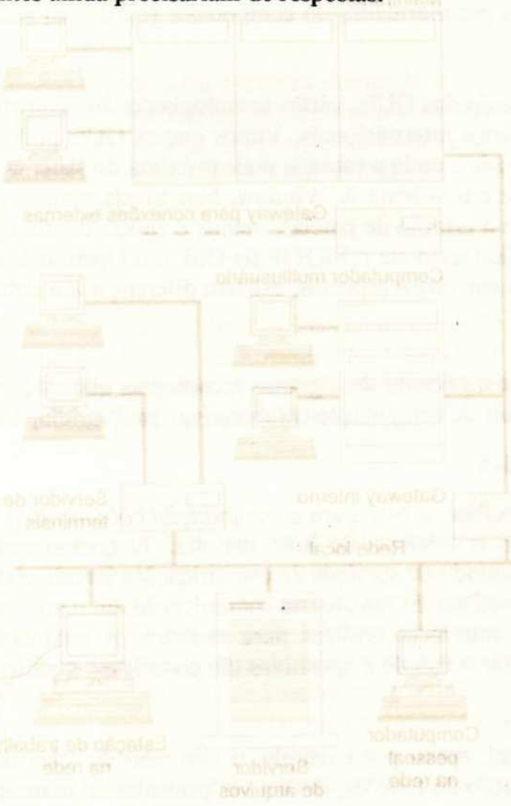
Vídeo em Janelas

Quando as aplicações de vídeo tornarem-se comuns nos ambientes de computação, o que irá depender de redes com larguras de faixa que dêem conta do recado, as GUIs poderão fornecer um diferencial de visualização claro entre a imagem de vídeo e os mecanismos para controlar a sua apresentação. Em ambientes com múltiplas janelas, uma fita de vídeo poderá ser vista em uma das janelas, enquanto algumas medidas podem ser anotadas em um gráfico em outra janela, para descrever detalhes sobre o assunto sendo apresentado. Controles devem ser disponibilizados para que a operação destas janelas não atrapalhem a apresentação do vídeo, que é o mais importante no processo.

Estes controles devem ser colocados no topo ou na parte de baixo do vídeo, de tal forma que os usuários possam ter acesso rápido aos mesmos sem a necessidade de pressionar um botão em um local diferente a cada momento. Aqui verificamos que o projeto de GUIs poderá enfrentar problemas sérios, e que muitos dos conceitos atuais poderão ajudar a solucioná-los [Wat93a].

Multimídia

A multimídia [Kob91] promete trazer novas características e melhorar nossa experiência ao trabalharmos com computadores. Os desafios que estão surgindo para as aplicações multimídia são da responsabilidade dos desenvolvedores. A interação visual por si implica em questões que serão muito demoradas de serem resolvidas. As questões relacionadas com ferramentas de proteção de direitos autorais, bibliotecas de imagens, sons, animação, etc, irão ocupar advogados especializados em patentes e em direitos autorais por várias décadas. Mesmo que esta tecnologia estivesse plenamente disponível hoje em dia, muitas questões com relação ao projeto das interfaces de usuários adequadas para estes ambientes ainda precisariam de respostas.



Capítulo 9: A Gerência da Rede Corporativa

9.1. A Necessidade de Gerenciamento: Visão Geral

Computadores requisitam um alto grau de gerenciamento, principalmente nos ambientes de computadores de grande porte, suportados por equipamentos tipo "mainframes". Equipamentos sofisticados e caros sempre exigiram analistas de suporte e operadores para mantê-los operacionais, e também para ajustá-los às necessidades de seus usuários. Mais recentemente, as técnicas usadas no gerenciamento dos mainframes têm melhorado consideravelmente. Hardware cada vez mais confiável, e a possibilidade de se automatizar procedimentos, reduziram a necessidade de pessoal especializado, o que gerou uma economia em necessidades extras para a operação destes equipamentos.

Em instalações compostas por ambientes relativamente homogêneos, constituídas por mainframes e midranges, os sistemas eram gerenciados separadamente das redes, e estas eram gerenciadas a partir de ferramentas fornecidas pelos fabricantes do hardware. Atualmente, o ambiente é composto por redes e computadores (servidores e clientes) de diversos fabricantes, o que muda bastante o cenário de gerenciamento.

9.1.1. Os Problemas com o Gerenciamento

Infelizmente, a padronização do gerenciamento de sistemas ainda deixa a desejar [CW94b]. Na figura 9.1, podemos ver um sistema corporativo típico, que consiste de uma variedade de tipos de computadores, e uma gama de equipamentos para redes locais ou redes de longo alcance. Quase tudo mostrado na figura precisa de algum tipo de gerenciamento. Analisamos rapidamente as questões de gerenciamento relacionadas com os componentes do ambiente distribuído corporativo:

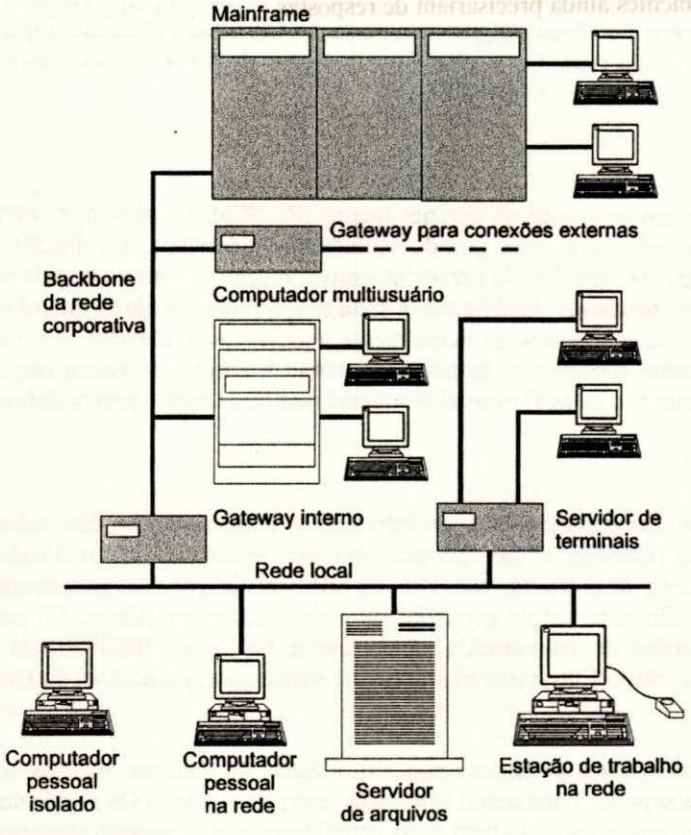


Figura 9.1. Um ambiente de computação distribuída corporativo [JM90]

Mainframes

Esta é uma das áreas de gerenciamento mais bem definidas e amadurecidas, e permite que estes ambientes sejam automatizados a tal nível que não exista a necessidade de intervenção humana. Existe uma grande disponibilidade de software sofisticado para diagnósticos remotos e tratamento de situações de exceção [Cas93].

Computadores multi-usuários

Em parte, a tecnologia para o gerenciamento dos mainframes tem migrado para os computadores de médio porte (midrange) e para os minicomputadores. Equipamentos como o AS/400 da IBM ou da família VAX da Digital, precisam da atenção exclusiva de um analista de suporte. Computadores que implementam o sistema operacional Unix requerem serviços de gerenciamento também dedicados ([Tay93a] [Bur93e]). Fornecedores independentes começam a disponibilizar no mercado pacotes de software de gerenciamento destes ambientes, que agora incorporam características de interfaces de gerenciamento unificadas para serem usadas em uma grande variedade de equipamentos [Tho92].

Servidores na rede

Um servidor na rede pode ser um equipamento dedicado, ou então pode ser um computador multi-usuário de propósito geral, o qual, dentre outras funções pode armazenar dados para sistemas remotos. Este servidor deve cuidar de tarefas como "backup" de dados, e de identificar usuários autorizados [Koe93].

Microcomputadores pessoais isolados

Estes são raramente bem administrados. Como o equipamento é pessoal, fica por conta da pessoa que o utiliza executar as funções de gerenciamento, como por exemplo, backup de arquivos.

Estações de trabalho na rede

Podemos considerar estes equipamentos em um nível entre os microcomputadores isolados e os servidores. Se uma estação de trabalho não possuir disco próprio (uma estação "diskless"), fica claro que a responsabilidade pelos backups deve ser do gerente da rede. Os usuários nas estações de trabalho são responsáveis somente por aspectos como a configuração e segurança de dados privados.

Redes Locais

As redes locais exigem um alto grau de gerenciamento [Red92b]. Os sistemas operacionais para redes locais [Red92a] mais modernos oferecem uma grande variedade de serviços, geralmente suficientes para o gerenciamento de pequenas redes. As questões de gerenciamento começam a tornar-se mais complexas à medida que a rede cresce, ou quando uma mudança significativa é feita em seu uso, ou ainda quando começam a ocorrer problemas não esperados. Redes locais maiores, principalmente aquelas que envolvem o uso de facilidades como "gateways", por exemplo, ou muitas tecnologias interligadas, como Ethernet ou Token-ring, apresentam nestas circunstâncias situações excepcionais, e exigem pessoal altamente qualificado para gerenciá-las.

Redes de longo alcance

Assim como ocorre no gerenciamento dos mainframes, o gerenciamento das redes de longo alcance (WANs - Wide Area Networks) [BRI93b] exige enormes esforços de gerenciamento. Produtos comerciais dos principais fabricantes encontram-se disponíveis, como os da IBM ou da AT&T, juntamente com produtos de empresas especializadas. Todos são sistemas de gerenciamento de redes proprietários. Só para ilustrar, citamos alguns exemplos de produtos, todos de reconhecida competência: NetView [BRI93b] da IBM, CA-NetMan [Cas93] da Computer Associates, EMA (Enterprise Management Architecture) [BRI93b] da DEC e AccuMaster da AT&T [Cas93].

Estes são produtos típicos para o gerenciamento centralizado de sistemas operacionais com serviços de redes, que no entanto começam a suportar o ambiente distribuído em vários níveis. Os problemas do gerenciamento nas configurações mais dispersas certamente são bem mais complexos do que aqueles encontrados no mundo dos mainframes centralizados. Atualmente, o gerenciamento do ambiente distribuído pode ser considerado o "Calcanhar de Aquiles" da computação distribuída [Cas93].

Padrões para o gerenciamento de redes, abertos, já existem, e estabelecem procedimentos e protocolos que cobrem as necessidades tanto das redes de longo alcance quanto das redes locais. Falamos mais adiante sobre estes padrões, que compõem o principal tópico deste capítulo.

9.1.2. Os Pontos de Controle

Os novos modelos da computação distribuída fizeram surgir novas questões para o gerenciamento de redes [Dew93]. Em gerações de hardware mais antigas, um computador único comunicava-se com seus usuários por meio de terminais "mudos" (figura 9.2 a). Nesta situação, fica claro que o computador central é o ponto de controle para o gerenciamento. Ele é o único elemento configurável do sistema, e permite que um único administrador tenha controle sobre todas as facilidades que são disponibilizadas para os usuários.

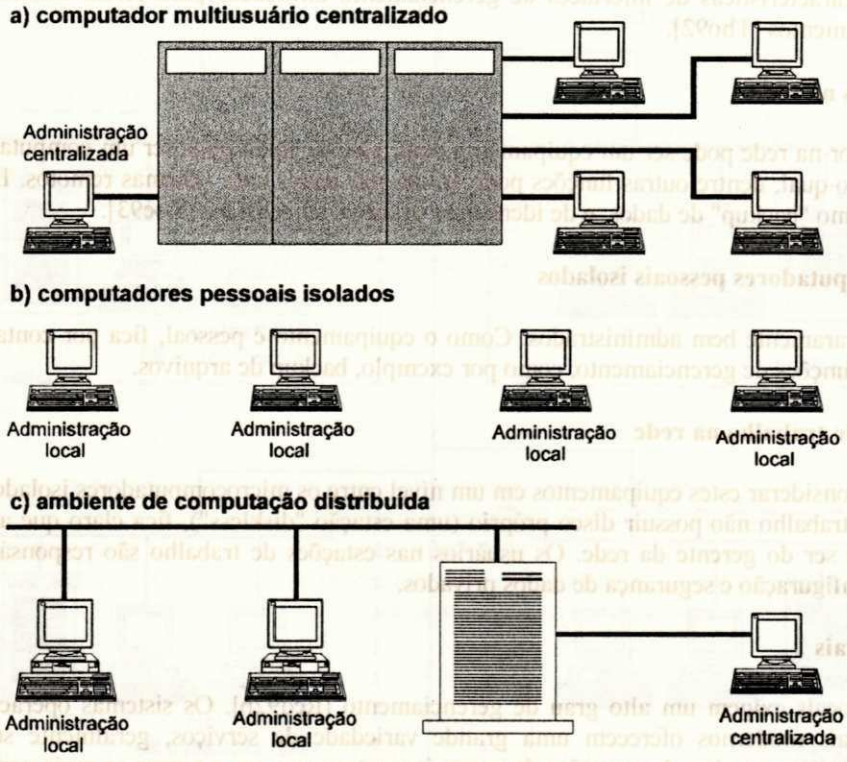


Figura 9.2. Localização dos pontos de controle para o gerenciamento efetivo [Gra91]

As estações de trabalho isoladas, (ou microcomputadores isolados) apresentam uma situação parecida. Existe um único ponto de controle para cada computador (figura 9.2 b).

Somente quando os computadores são interligados em uma rede esta situação muda substancialmente (Figura 9.2. c). Além de equipamentos inteligentes na frente de cada usuário, existem também computadores que agem como recursos partilhados. Cada elemento do ambiente de processamento distribuído exige administração, e as tarefas administrativas envolvidas podem ser divididas em duas categorias: aquelas que afetam a somente um usuário, e aquelas que afetam a mais de um usuário.

É possível, em um ambiente de redes, que se concentrem todas as tarefas administrativas em um único ponto, criando-se uma situação como a da figura 9.2 a) anterior, e proibindo-se que os usuários tenham qualquer tipo de controle sobre seus ambientes. No entanto, esta não é uma situação adequada. É sempre melhor que se permita que os usuários adaptem as configurações de seus sistemas às suas necessidades, o que pode melhorar a produtividade, desde que as regras de integridade do sistema distribuído como um todo não fiquem comprometidas. Desta forma, os usuários são os responsáveis pelos elementos da gerência local, enquanto que um administrador da rede cuida dos recursos partilhados na mesma.

As redes possuem uma tendência natural de se tornarem maior em extensão, e também em seu número de usuários. É conveniente, tanto tecnicamente quanto administrativamente, que se divida uma grande rede, ou domínio, em sub-domínios, como mostrado na figura 9.3 [Ste91].

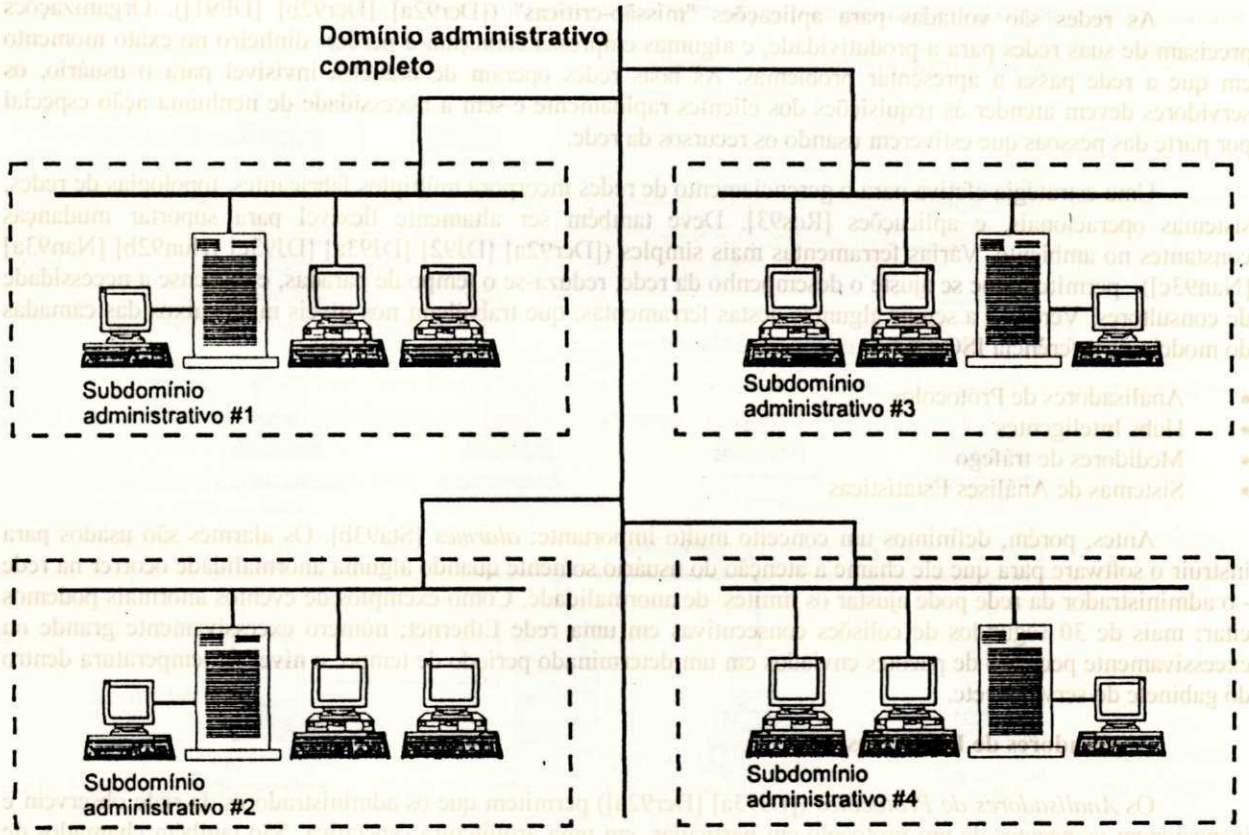


Figura 9.3. O gerenciamento da rede com o enfoque de domínios e subdomínios [Ste91]

9.1.3. O Monitoramento de Sistemas

O monitoramento de sistemas [MSS93] é essencial para o planejamento de necessidades nos ambientes computacionais. Este tipo de planejamento torna-se mais difícil quando as aplicações distribuídas dividem os serviços de processamento entre diferentes computadores, ou quando os dados são distribuídos entre diferentes computadores, geralmente com estes computadores em diferentes redes. Nos mainframes, a tarefa de monitoramento é trivial, e as ferramentas abundam. Nos sistemas distribuídos estas ferramentas ainda estão começando a serem migradas ou desenvolvidas [Hun92b].

O primeiro passo na identificação de gargalos é feito por meio da coleta de dados sobre as operações correntes, nos computadores, e na própria rede. O monitoramento do desempenho [Sen94] está mais relacionado com: servidores e clientes; SGBDs; e, aplicações distribuídas.

Para *servidores e clientes*, o monitoramento do desempenho varia de acordo com o sistema operacional. Em geral, as ferramentas são mais robustas para os sistemas operacionais maduros. Para os *SGBDs*, existem muitas ferramentas de monitoramento para produtos como por exemplo, o DB2 da IBM [Cas93], porém existem poucas destas ferramentas para os servidores típicos do ambiente da computação distribuída, tais como o Oracle ou o SQL Server [Sal93]. Para as *aplicações distribuídas* [Sch92c], seu acompanhamento torna-se ainda mais difícil porque normalmente o processo ocorre na retaguarda, o que não gera informação de monitoramento, apesar de existirem tarefas sendo executadas. O processamento também pode se estender entre vários clientes e servidores. Neste caso, uma ferramenta de monitoramento teria que estar apta a seguir cada tarefa de uma transação, seguir o desempenho em cada passo, e então tabular os resultados.

9.1.3.1. Ferramentas Simples para o Gerenciamento da Rede

Na depuração de problemas na rede, é importante que se avaliem as causas potenciais de um problema, e que se utilize esta avaliação para direcionar as regras para a descoberta de outros problemas relacionados. Deve-se trabalhar em todas as camadas de protocolos, de tal forma que se assegure que não se passe por cima de problemas de camadas mais baixas, quando estes parecem estar ocorrendo em camadas mais altas.

As redes são voltadas para aplicações "missão-críticas" ([Der92a] [Der92b] [DF91]). Organizações precisam de suas redes para a produtividade, e algumas empresas começam a perder dinheiro no exato momento em que a rede passa a apresentar problemas. As boas redes operam de maneira invisível para o usuário, os servidores devem atender às requisições dos clientes rapidamente e sem a necessidade de nenhuma ação especial por parte das pessoas que estiverem usando os recursos da rede.

Uma estratégia efetiva para o gerenciamento de redes incorpora múltiplos fabricantes, topologias de redes, sistemas operacionais, e aplicações [Ros93]. Deve também ser altamente flexível para suportar mudanças constantes no ambiente. Várias ferramentas mais simples ([Der92a] [DJ92] [DJ93a] [DJ93c] [Hun92b] [Nan93a] [Nan93c]) permitem que se ajuste o desempenho da rede, reduza-se o tempo de paradas, e dispense a necessidade de consultores. Veremos a seguir algumas destas ferramentas, que trabalham nos níveis mais baixos das camadas do modelo de referência ISO/OSI.

- Analisadores de Protocolos
- Hubs Inteligentes
- Medidores de tráfego
- Sistemas de Análises Estatísticas

Antes, porém, definimos um conceito muito importante: *alarmes* [Sta93b]. Os alarmes são usados para instruir o software para que ele chame a atenção do usuário somente quando alguma anormalidade ocorrer na rede - o administrador da rede pode ajustar os limites de anormalidade. Como exemplos de eventos anormais podemos citar: mais de 30 segundos de colisões consecutivas em uma rede Ethernet; número excessivamente grande ou excessivamente pequeno de pacotes enviados em um determinado período de tempo; o nível da temperatura dentro do gabinete do servidor; etc.

Analisadores de Protocolos

Os *Analisadores de Protocolos* ([DJ93a] [Der92a]) permitem que os administradores da rede observem e inspecionem os pacotes de um protocolo em particular, em uma arquitetura específica. São também chamados de farejadores ("sniffers").

Uma aplicação distribuída que deseje comunicar-se com outra em um sistema remoto deve empacotar a mensagem a ser enviada entre campos de controle à esquerda e à direita dos dados, de acordo com o formato determinado pelo protocolo que está sendo usado. Estes campos de controle são como um envelope para a mensagem enquanto ela transita nos meios de comunicações. Desde que os sistemas que enviam e recebem a mensagem utilizem o mesmo protocolo, então eles saberão como ler o endereço no envelope, roteá-lo, entregá-lo e até mesmo fornecer um recibo de entrega (acknowledgement). Se a comunicação sobre o meio de ligação cair , a leitura dos campos de controle, e mesmo a abertura do envelope e decodificação do seu conteúdo, pode fornecer pistas para o problema que estiver ocorrendo na rede. Isto é feito pelos analisadores de protocolos.

Analisadores de Protocolos são normalmente fornecidos na forma de pequenos gabinetes portáteis (ou notebooks) com display em vídeo e software sofisticado para emissão de gráficos e relatórios. Os analisadores de protocolos de rede capturam pacotes de dados que estejam transitando na rede e utilizam um software especial para decodificá-los. Permitem que se filtrem e classifiquem os dados capturados, para facilidade de processamento. Pode-se usar o analisador de protocolos para mostrar os pacotes seletivamente em tempo real, ou então capturá-los para estudos posteriores.

Alguns produtos possuem facilidade para a identificação do protocolo em uso. Os dados capturados após analisados fornecem dicas sobre irregularidades ou falhas na rede.

Hubs Inteligentes

Os *hubs* [Day92] representam um forte ponto de pulsação na rede. Todo o tráfego na rede passa pelos hubs, mesmo o tráfego que sobrepassa o servidor de arquivos e vai diretamente de uma estação cliente para outra.

Um microprocessador à parte pode ser colocado à disposição dos mecanismos de administração da rede no hub. Estes processadores trabalham juntamente com software em uma estação na rede, para reportar a atividade em todos os nós da rede, e controlá-los quando necessário (principalmente desconectando-os).

Sistemas de Análises Estatísticas

Os *Sistemas de Análises Estatísticas* [DJ92] trabalham com software (não é necessário hardware especial) que faz medições dos seguintes fatores:

- Quantidade de espaço em disco usado por uma aplicação em particular, pessoas ou centros de custos;
- Quantidade de atividade em programas ou arquivos específicos;
- Tempo de conexão de pessoas específicas ou estações clientes;
- No. de jobs de impressão;
- Carga do servidor, durante um período predeterminado.

Estes sistemas baseiam-se em facilidades similares ofertadas desde muito tempo pelos ambientes de computadores mainframes e minis. Os administradores podem utilizar as medições para: planejar o crescimento da rede; determinar bases de comparações; detectar problemas com antecedência; e, justificar orçamentos.

Medidores de Tráfego

No caso de indisponibilidade de um hub, pode-se monitorar o volume de atividade na rede ou receber alerta quando certos tipos de pacotes "errantes" se movimentam na rede, por meio de produtos alternativos e fáceis de se instalar e operar - com um custo bem reduzido. *Medidores de tráfego* ([DJ93c] [Der92a]) são programas que podem rodar em qualquer estação da rede. Fornecem excelentes displays tipo fotografias ("snapshots") da atividade da rede, estatísticas, e medem a qualidade da transmissão nos cabos. Porém, não são ativos, ou seja, não podem desconectar estações com problemas.

9.1.4. Gerenciamento de Redes: A Necessidade de Padrões

As redes e os sistemas de processamento distribuído estão crescendo em importância, e por conseguinte, tornaram-se críticos no mundo dos negócios. Nos ambientes empresariais, a tendência é por redes maiores e mais complexas, que suportem um maior número de usuários e aplicações. À medida que estas redes crescem em escala, dois fatos vão ficando mais evidentes:

- As redes, juntamente com seus recursos e aplicações distribuídas tornam-se cada vez mais indispensáveis para as organizações;
- Com o crescimento das redes, existe conseqüentemente uma maior possibilidade de ocorrerem problemas, o que pode levá-la a um estado de inoperância ou a níveis inaceitáveis de desempenho.

Uma grande rede não pode ser montada e gerenciada somente por esforços humanos. A complexidade dos sistemas impõe o uso de ferramentas de gerenciamento automatizadas ([Ros93] [Tho92]). Para um controle de custos efetivo, exigem-se ferramentas de gerenciamento padronizadas para atenderem a um grande leque de produtos, incluindo servidores, pontes, roteadores, e equipamentos de telecomunicações em geral. A necessidade de protocolos e programas para monitoramento e controle de redes de computadores torna-se mais crítica em grandes redes interligadas [Qua90]. Para atender a estas necessidades, dois esforços de padronização têm sido especificados: o esforço da ISO/OSI e o da Internet [Sta93b].

As arquiteturas básicas de gerenciamento usadas pela ISO (International Standards Organization) e a Internet são bastante similares. Esta similaridade não é gratuita; seus esforços de desenvolvimento influenciaram-se mutuamente. Ambas arquiteturas gerenciam a rede por meio do uso de uma base de dados de informações gerenciais chamada **MIB** (*Management Information Base*) [SOIEC91b] que contém dados no formato definido em uma estrutura de informações gerenciais chamada de **SMI** (*Structure of Management Information*) [SOIEC91b]. Entretanto, a MIB e a SMI são bastante diferentes nas duas pilhas de protocolos, e diferentes protocolos são utilizados no gerenciamento da MIB: estes protocolos de gerenciamento são o **SNMP** (*Simple Network Management Protocol*) [CFSD90] no mundo da Internet e o **CMIP** (*Common Management Information Protocol*) [SOIEC90] no mundo ISO/OSI.

Observe que as idéias básicas para SMI e MIB são também apropriadas para o gerenciamento de sistemas operacionais em ambientes com redes [Qua90].

Falamos a seguir a respeito dos principais padrões para gerenciamento de redes existentes. Particularmente, descrevemos o SNMP e o CMIP com maior riqueza de detalhes nas seções adiante.

HEMS

O sistema *HEMS (High-Level Entity Management System)* ([Qua90] [Hal93b] [AP94a]) foi uma proposta para o gerenciamento de nós remotos para redes TCP/IP. Apesar de ser tecnicamente interessante (e muitas de suas idéias foram implementadas em outros protocolos), foi deixado de lado pelos próprios autores, que queriam evitar debates demorados sobre as definições de padrões para protocolos de gerenciamento de redes.

SGMP

O sistema *SGMP (Simple Gateway Monitoring Protocol)* ([Qua90] [AP94a]) foi desenvolvido nos idos de abril de 1984, porém jamais chegou a ser implementado. Seu propósito era monitorar a rede Internet (que àquela época estava crescendo desenfreadamente). A maioria de suas idéias foram aproveitadas pelo projeto que o sucedeu: o SNMP.

SNMP

O padrão *SNMP (Simple Network Management Protocol)* [CFSD90] evoluiu a partir das idéias do SGMP à medida que necessidades adicionais de gerenciamento foram sendo identificadas na Internet, tais como:

- A necessidade de se gerenciar de fato uma internet, e não de somente monitorá-la;
- A necessidade para se monitorar entidades que não somente redes e gateways (como por exemplo: servidores, pontes, hubs);
- A necessidade de se facilitar a transição para o mundo ISO/OSI.

O SNMP foi especificado em sua forma inicial em junho de 1988, e em sua forma final em agosto deste mesmo ano. A empresa NYSERNet, Inc., formada a partir de esforços de colaboração de várias universidades, do governo do estado de Nova Iorque, e de várias organizações comerciais, foi a responsável pelo desenvolvimento e implementação do SGMP, e de seu sucessor, o SNMP [Yem94].

Apesar de apresentar muitas carcaterísticas interessantes, podemos destacar algumas limitações para o SNMP, incluindo: falta de características de segurança; impossibilidade de se efetuar coleta de dados gerenciados em grandes volumes (bulk); impossibilidade de se conectar diferentes aplicações de gerenciamento de redes; qualidade da documentação inconsistente; permite que se criem configurações não-padrão (MIBs com extensões).

SNMP Versão 2

Para suprir as deficiências de projeto da versão 1 do SNMP, foi proposta pelo IETF (Internet Engineering Task Force), o grupo de padronização para o TCP/IP, uma nova versão: ***o SNMP versão 2*** [SS94] (ou melhor, **SNMPv2**). A nova versão suporta o gerenciamento de aplicações, e fornece mecanismos para comunicação segura entre os dois sistemas gerentes, e entre os periféricos gerenciados na rede. Permite que se efetue coleta de dados gerenciados em grandes volumes, e os mecanismos para tratamento de erros foram bastante melhores. A grande vantagem que merece destaque é a possibilidade de uso desta versão conjugada com uma série de serviços de transporte, incluindo OSI, IPX, AppleTalk, e o próprio TCP/IP. SNMP versão 1 só suporta TCP/IP.

A versão 2 do SNMP permite a comunicação de sistemas gerentes com outros sistemas gerentes, o que possibilita que uma estação gerente centralizada delegue tarefas para sistemas gerentes em sub-redes. O suporte aos padrões OSI também é de fundamental importância para a versão 2 do SNMP .

Infelizmente, a versão SNMPv2 não é compatível com versões antigas do SNMP, o que torna a transição de uma versão para a outra mais difícil [Dew94].

CMIP

O padrão *CMIP (Common Management Information Protocol)* [SOIEC90] é o protocolo da ISO/OSI para o gerenciamento da MIB. CMIP já está adiantado no processo de formalização de sua padronização, tendo sido aprovado formalmente em 1990. No entanto, muitas outras partes da arquitetura de gerenciamento ISO/OSI ainda estão para ser padronizadas, e algumas encontram-se no início de sua especificação, como é o caso da própria MIB.

Baseado nos padrões OSI, o CMIP busca suprir as limitações do SNMP. No entanto, produtos que utilizam o padrão CMIP são muito recentes, muito poucos em número, mais caros que seus correspondentes SNMP (ou SNMPv2), e, por oferecerem funcionalidades adicionais, são mais complexos, pois exigem uma maior quantidade de recursos como memória e capacidade de processamento para rodarem a contento.

9.1.4.1. A Implementação dos Padrões

Mesmo que as informações de gerenciamento sejam padronizadas, cada fabricante ainda terá que fornecer suas próprias aplicações de gerenciamento da rede, projetadas especificamente para suportar os equipamentos vendidos por eles. Se considerarmos uma grande rede com muitos equipamentos de diferentes fabricantes, gerenciada a partir de um ponto de controle central, os operadores neste centro de operações terão que monitorar muitas consoles. Adicionalmente, alguns dos dados capturados nas MIBs só terão algum significado para os sistemas de gerenciamento de redes fornecidos por determinado fabricante. O resultado final é uma MIB proprietária, o que vai totalmente contra os preceitos dos sistemas abertos [Mal92b].

Podemos citar como exemplo, o caso da arquitetura *SNA (Systems Network Architecture)* da IBM, que suporta a família de produtos NetView [BRI93b] para o gerenciamento de redes. Porém, esta é uma solução proprietária típica, e não abrange os ambientes multifabricantes ou fornece soluções para os sistemas abertos (Na realidade, parte dos produtos NetView já começam a ser migrados para as plataformas Unix da IBM).

O DME enfoca a falta de padronização para a MIB. Uma de suas primeiras partes disponíveis é justamente a *CM/API (Consolidated Management Application Programming Interface)* [Dew94] que usada em conjunto com os mecanismos de coleta de dados, suporta o gerenciamento integrado entre ambientes CMIP e SNMP, incluindo o suporte a periféricos e circuitos de telecomunicações e também a equipamentos do mundo das redes locais, como pontes e roteadores.

9.2. Conceitos Básicos sobre o Gerenciamento de Redes

As funções de gerenciamento de redes podem ser agrupadas em duas categorias [Sta93b]: o *monitoramento da rede* e o *controle da rede*.

O *monitoramento da rede* [MSS93] é uma função de "leitura", e está relacionado com a observação e análise do estado e do comportamento da configuração e de seus componentes. O monitoramento da rede envolve três aspectos de projeto:

- *Acesso às informações monitoradas*: define as informações a serem monitoradas e a maneira como se obtém esta informação, ou seja, como ela é passada de um recurso para o gerente;
- *Projeto dos mecanismos de monitoramento*: define as melhores estratégias de se obter informações dos recursos gerenciados;
- *Utilização das informações de monitoramento*: define como a informação de monitoramento pode ser utilizada para permitir a análise e diagnóstico de problemas nas várias áreas funcionais de gerenciamento.

Por sua vez, o *controle da rede* [Sta93b] é uma função de "alteração" e está relacionada com a modificação de parâmetros em vários componentes da configuração, o que faz com que estes componentes executem ações pré-definidas.

9.2.1. As Informações de Monitoramento da Rede

As informações que devem ser avaliadas para o monitoramento da rede podem ser classificadas da seguinte maneira [SOIEC91b]: estáticas, dinâmicas e estatísticas.

As informações estáticas caracterizam a configuração atual da rede e seus componentes. Estas informações mudam com pouca frequência. São informações usualmente geradas pelo componente envolvido. Por exemplo, um roteador mantém suas próprias informações de roteamento (na forma de tabelas) [BG93].

As informações dinâmicas estão relacionadas com eventos na rede, como por exemplo a transmissão de um pacote pela rede. São também geralmente coletadas e armazenadas pelo componente da rede responsável pelos eventos ligados a ele.

As informações estatísticas são derivadas das informações dinâmicas. Indicam por exemplos a média de pacotes transmitidos em um intervalo de tempo. Podem ser geradas por qualquer sistema que tenha acesso às informações dinâmicas.

9.2.2. O Modelo Funcional de Monitoramento de Redes

A Figura 9.4 descreve um modelo em termos funcionais para o monitoramento de redes.

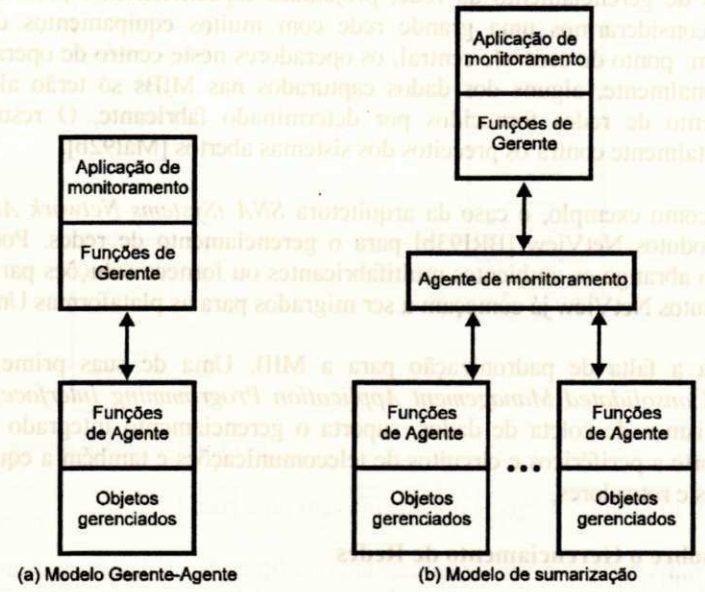


Figura 9.4. O modelo funcional de monitoramento de redes [Sta93b]

Os principais componentes de um sistema de gerenciamento são os seguintes [SOIEC89]:

O *monitor*: constitui a estação que irá fornecer a console de monitoramento, e define o centro de controle de toda a rede

Aplicação de monitoramento: são os módulos de software que implementam as funções visíveis para os usuários, como por exemplo o monitoramento de desempenho e o monitoramento de falhas.

Sistema gerente: é o módulo de software no monitor (estação de monitoramento) que executa as funções básicas de gerenciamento, como por exemplo a recuperação de informações em outros componentes da configuração.

Sistema agente: é o módulo de software que obtém e armazena a informação de gerenciamento para um ou mais elementos da rede, e passa estas informações para o monitor.

Objetos gerenciados: são as informações de gerenciamento que representam os recursos e suas atividades.

Agente de monitoramento: é um módulo que gera sumários e análises estatísticas das informações de gerenciamento. Quando um elemento possui um sistema agente disponível, as informações de gerenciamento coletadas localmente devem ser passadas para o monitor, e lá deve-se proceder com a sumarização e análise dos dados. Porém, se o monitor não precisar acessar todos os dados coletados (e evitando-se processamento desnecessário no monitor e congestionamento do meio de comunicações) pode-se disponibilizar neste elemento um agente de monitoramento, que fará toda a sumarização e enviará o resultado para o monitor.

9.2.2.1. Configurações Possíveis para os Módulos de Gerenciamento

Os módulos funcionais descritos na seção anterior podem ser configurados de várias maneiras.

Uma configuração interessante é a do monitor (estação de gerenciamento). Geralmente incluem-se para o monitor, software agente e um conjunto de objetos gerenciados, como mostrado na figura 9.5 (a) . Como o monitor é um ponto crucial na rede, é da maior importância que ele possa monitorar-se a sí mesmo e aos recursos a ele conectados, de forma que se possa fornecer maior confiabilidade ao gerenciamento da rede como um todo.

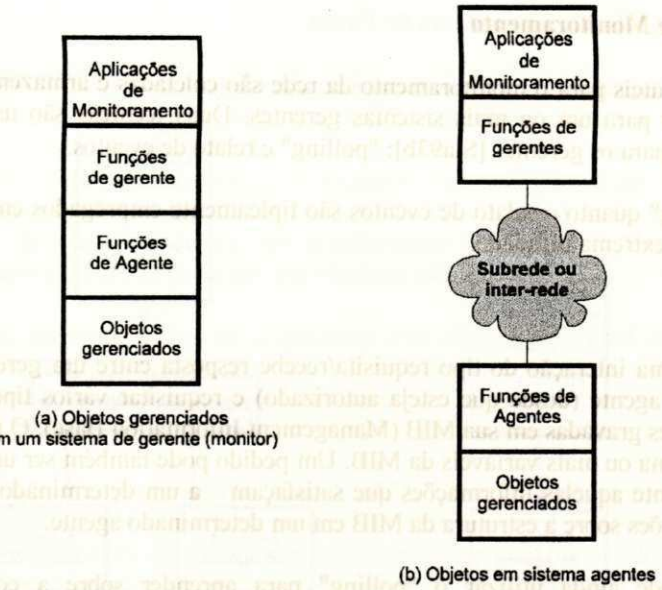


Figura 9.5. Objetos gerenciados em uma rede [Sta93b]

A configuração mais comum é aquela apresentada na figura 9.5 (b). Aproveitamos para lembrar conceitos vistos na seção 9.1.4 anterior: esta configuração exige que os sistemas agentes e gerentes partilhem os mesmos protocolos de gerenciamento (SNMP ou CMIP) e a mesma MIB (Management Information Base).

9.2.2.2. Agentes Procuradores

Existem porém, muitos casos onde os sistemas agentes não partilham os mesmo protocolos de gerenciamento que os sistemas gerentes, e em casos práticos, isto pode até não ser possível. Por exemplo, a configuração da rede pode incluir sistemas mais antigos e proprietários ("legacy") [Ges93c], que não suportam os protocolos padrões para gerenciamento de redes que costumam ser usados nos ambientes abertos. Existem ainda situações que incluem pequenos sistemas operacionais para redes locais, que podem ficar saturados com a implementação de uma solução completa de gerenciamento, e existem ainda componentes como modens ou multiplexadores que simplesmente não suportam software adicional.

Para se tratar estes casos, exige-se um *agente procurador ("proxy")* ([Sta93b] [AP94a] [BRI93b] [Com91b]), ilustrado na figura 9.6 . É comum, portanto, que exista na configuração um dos agentes atuando como um procurador para um ou mais nós.

Quando um agente executa a função de procurador, ele age pelos outros nós que ele representa. Um monitor que deseje obter informações de um nó, ou deseje controlá-lo, comunica-se como o agente procurador, através de um pedido por serviço. O agente procurador então traduz o pedido do monitor para uma forma apropriada para o componente desejado, e desta forma pode fazer uso de qualquer protocolo de gerenciamento de rede que esteja disponível no ambiente deste componente. As respostas do componente de volta para o agente procurador são traduzidas de maneira similar e passadas para o monitor. Na prática, um agente procurador é instalado em um componente que implementa os protocolos de transporte tanto do ambiente original quanto do ambiente proprietário, como por exemplo, em um roteador.

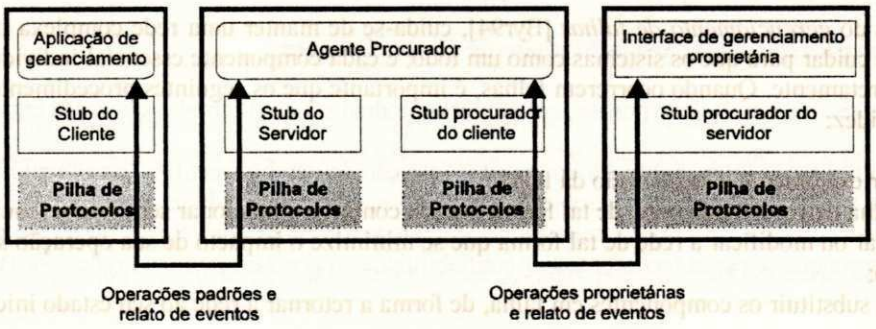


Figura 9.6. Os agentes procuradores ("Proxies") [Yem92]

9.2.3. Os Mecanismos de Monitoramento

As informações úteis para o monitoramento da rede são coletadas e armazenadas por sistemas agentes e disponibilizadas a seguir para um ou mais sistemas gerentes. Duas técnicas são usadas para disponibilizar as informações dos agentes para os gerentes [Sta93b]: "polling" e relato de eventos.

Tanto o "polling" quanto o relato de eventos são tipicamente empregados em sistemas de gerenciamento de redes, e ambos são de extrema utilidade.

9.2.3.1. "Polling"

O "polling" é uma interação do tipo requisita/recebe resposta entre um gerente e um agente. O gerente pode consultar qualquer agente (desde que esteja autorizado) e requisitar vários tipos de informações; o agente responde com informações gravadas em sua MIB (Management Information Base). O pedido por informações pode ser específico, listando uma ou mais variáveis da MIB. Um pedido pode também ser uma pesquisa, requisitando ao agente para passar somente aquelas informações que satisfaçam a um determinado critério de seleção. Pode-se ainda requisitar informações sobre a estrutura da MIB em um determinado agente.

Um gerente pode ainda utilizar o "polling" para aprender sobre a configuração que ele estiver gerenciando, para obter periodicamente informações sobre condições, ou para investigar uma determinada área com maior nível de detalhamento, depois de ser alertado sobre um problema.

No entanto, o mecanismo de "polling" gera "overhead" no uso da rede e da CPU do gerente.

9.2.3.2. Relato de Eventos

No relato de eventos, a iniciativa parte do agente, e o sistema gerente se comporta somente como um ouvinte, aguardando que cheguem informações. Um agente pode gerar um relatório periodicamente para fornecer ao gerente seu estado atual. O período de relatos pode ser preconfigurado, ou pode ser estabelecido pelo próprio gerente. Um agente pode gerar um evento quando uma condição significativa (por exemplo, uma mudança de estado) ou anormal (por exemplo, uma falha) ocorrer.

O relato de eventos é muito útil para que se detectem problemas assim que eles aparecem. É também mais eficiente que o "polling" em termos de uso de recursos físicos.

9.2.4. As Áreas Funcionais de Gerenciamento da ISO/OSI

Para que se possa estabelecer as linhas mestras de um projeto voltado para o desenvolvimento de uma facilidade de gerenciamento de redes, uma divisão das necessidades dos usuários em áreas funcionais é exigida, de tal forma que se possa estruturar todo o processo de desenvolvimento. Esta divisão foi proposta pela ISO como parte das especificações para os sistemas de gerenciamento OSI, a qual tornou-se largamente utilizada no sentido de descrever de uma maneira muito didática e útil as exigências de qualquer sistema de gerenciamento de redes. As áreas funcionais definidas são cinco [SOIEC89], e falamos sobre cada uma delas a seguir: gerenciamento de falhas, gerenciamento de desempenho; gerenciamento da configuração, gerenciamento da segurança e gerenciamento da contabilização do uso de recursos na rede.

Gerenciamento de Falhas

Através do *gerenciamento de falhas* [Byr94], cuida-se de manter uma rede complexa operacional. Neste contexto deve-se cuidar para que os sistemas como um todo, e cada componente essencial individualmente, estejam funcionando corretamente. Quando ocorrerem falhas, é importante que os seguintes procedimentos sejam seguidos com a maior rapidez:

- determinar exatamente a localização da falha;
- isolar a falha do restante da rede, de tal forma que ela continue a funcionar sem interferências;
- reconfigurar ou modificar a rede de tal forma que se minimize o impacto de sua operação sem o componente danificado;
- reparar ou substituir os componentes em falha, de forma a retornar a rede ao seu estado inicial.

As falhas devem ser diferenciadas dos erros [Yem92]. Uma falha é uma condição de anormalidade que requer uma ação. Ela é normalmente indicada pela operação incorreta de um componente ou por erros em excesso. Por exemplo, se uma linha de comunicação for cortada fisicamente, os sinais não podem alcançar o seu destino. Uma rachadura no cabo pode causar distorções pesadas que ocasionarão taxas de erros de bits altíssimas. No caso dos erros, eles ocorrem ocasionalmente, e podem ser geralmente tratados e recuperados por meio de mecanismos de controle de erros que existem em vários protocolos (por exemplo, erros de paridade).

O impacto e a duração das falhas podem ser minimizados através do uso de componentes redundantes e rotas de comunicações alternativas, de tal forma que se forneça à rede um certo nível de tolerância a falhas [Cri91].

Vale observarmos que, como em todas as outras áreas funcionais que veremos a seguir, o gerenciamento de falhas deve afetar minimamente o desempenho da rede.

Gerenciamento de Desempenho

O gerenciamento de desempenho [Sen94] deve monitorar muitos recursos para poder fornecer informações que determinem o nível de operacionalidade da rede. Através da coleta de informações, de sua análise, e pelo uso dos resultados desta análise para o estabelecimento de valores adequados, o administrador da rede pode reconhecer situações que indiquem degradações que estejam ocorrendo ou que estejam na iminência de ocorrerem. Alguns dos aspectos que são relevantes para o administrador da rede são:

- qual o nível de utilização possível para a rede?
- existe tráfego em excesso?
- a vazão degradou-se para níveis inaceitáveis?
- existem gargalos?
- o tempo de resposta está aumentando?

Os administradores da rede precisam de estatísticas que lhes permitam planejar, gerenciar e manter as grandes redes. Estatísticas de desempenho podem ser utilizadas para se descobrir os gargalos potenciais, antes que eles venham a causar problemas para os usuários finais, de tal forma que ações apropriadas possam ser tomadas.

Em planejamento de longo prazo, a questão do planejamento de necessidades baseado em informações estatísticas de desempenho podem indicar as decisões apropriadas que devem ser tomadas. Por exemplo, pode-se ter a certeza da necessidade de se expandir as linhas de comunicações em uma determinada área, ou de se adquirir unidades de discos adicionais para o armazenamento de arquivos ou bancos de dados.

Gerenciamento da Configuração

O gerenciamento da configuração [Ray94] está envolvido com a inicialização da rede e com o seu término de maneira suave. Envolve também a manutenção, adição e atualização dos relacionamentos entre os componentes da rede, e o monitoramento do estado deste componentes. Outra das funções do gerenciamento de configuração é a distribuição e atualização de software na rede - uma função particularmente importante para os ambientes da computação distribuída.

A reconfiguração da rede é geralmente requisitada para atender às medições de desempenho - que podem ter detectado um possível gargalo - ou para implementar uma atualização na rede, ou uma recuperação de falha, ou verificações na segurança.

Gerenciamento da Segurança

O gerenciamento da segurança ([Sch94] [Kay94a]) está relacionado com o monitoramento e o controle de acessos a redes de computadores. Cuida também da geração, distribuição, e armazenamento de chaves de segurança. Senhas de acesso e outras informações de controle ou de autorização também devem ser mantidas e distribuídas.

O gerenciamento da segurança fornece mecanismos para a proteção dos recursos da rede e de informações dos usuários.

Os arquivos de "log" são uma importante ferramenta de segurança, e portanto, o gerenciamento da segurança esta envolvido com a coleta, armazenamento, e análise de registros de auditoria e logs de segurança, assim como da ativação/desativação destas facilidades de "logging".

Gerenciamento da Contabilização

O gerenciamento de contabilização está relacionado com a cobrança pelo uso dos serviços na rede. O administrador da rede deve poder acompanhar o uso de recursos na rede por um usuário, ou por um grupo de usuários, por várias razões, dentre outras:

- Um usuário, ou um grupo de usuários podem estar abusando de seus privilégios de acessos e atrapalhando o uso da rede por outros usuários;
- Os usuários podem estar fazendo um uso ineficiente da rede, e o administrador da rede pode calibrar estes níveis de ineficiência por meio do uso de procedimentos que melhorem o seu desempenho;
- O administrador da rede estará em uma posição mais confortável para planejar o crescimento da rede, se ele conhecer o nível de atividades dos usuários em detalhes.

9.3. Sistemas de Gerenciamento de Redes

Nas seções anteriores já falamos de sistemas agentes, sistemas gerentes, e falamos até de um modelo funcional para o gerenciamento de redes. Nesta seção definimos em termos mais formais os sistemas de gerenciamento de redes.

Um *sistema de gerenciamento de redes* ([Tho92] [BRI93b] [Kle88] [Ros93]) é composto por ferramentas para o monitoramento e controle da rede. Ele consiste de módulos de software, e também de hardware, implementados nos componentes de uma rede. O software utilizado para a execução das tarefas de gerenciamento reside tanto nos computadores em sí (como mainframes, servidores, estações de trabalho, etc.) quanto nos processadores especializados de comunicações (como processadores frontais -"front-ends"-, controladores de terminais, hubs para redes locais, pontes e roteadores). Os elementos ativos na rede fornecem informações regulares para um centro de controle da rede (o monitor).

Uma arquitetura para um sistema de gerenciamento de redes está sugerido na figura 9.7 . O conjunto de softwares que implementam as tarefas de gerenciamento são denominados, quando instalados em um dos nós da rede, de *entidade de gerenciamento da rede* (EGR) ([Sta93b] [AP94a] [Yem93]). Na nossa figura, todos os nós na rede estão representados com uma EGR correspondente, porém, na prática, as EGRs só serão disponibilizadas para aqueles nós que se deseje monitorar explicitamente.

As tarefas de uma EGR são as seguintes:

- coletar estatísticas sobre as atividades da rede;
- armazenar estas estatísticas localmente;
- atender aos comandos do centro de controle, incluindo:
  - transmitir as estatísticas coletadas para o centro de controle da rede;
  - modificar parâmetros;
  - fornecer informações de estado;
  - gerar tráfego artificial para teste.

Pelo menos um dos computadores na rede será designado para executar o papel de *gerente* ([Sta93b] [AP94a] [Yem93]). Além de suas EGRs, o gerente possui também uma série de programas chamados de *aplicações de gerenciamento de redes (AGRs)* ([Sta93b] [AP94a] [Yem93]). Estes programas cuidam de funções como o monitoramento de desempenho da rede, o monitoramento de falhas e o controle da configuração. A AGR inclui também uma interface gráfica de usuário (GUI) para permitir que operadores autorizados gerenciem a rede a partir de um terminal de gerenciamento, também chamado de *monitor* ([Sta93b] [AP94a] [Yem93]) (que pode ser um terminal assíncrono de um mainframe, e neste caso a GUI é orientada a caracteres, ou então uma estação de trabalho na rede, ou ainda um terminal X).

A AGR atende aos comandos dos operadores mostrando respostas na tela ou emitindo comandos para as EGRs espalhadas pela rede. Observe que esta comunicação é efetuada utilizando-se um protocolo padrão de gerenciamento de redes na camada de aplicação (por exemplo, SNMP ou CMIP).

Cada nó da rede que é parte do sistema de gerenciamento e inclui uma EGR, que é referenciada nos ambientes de gerenciamento de redes como um *agente* ([Sta93b] [AP94a] [Yem93]). Agentes incluem os computadores que suportam as aplicações dos usuários e os nós que fornecem serviços de comunicações tais como pontes, roteadores.

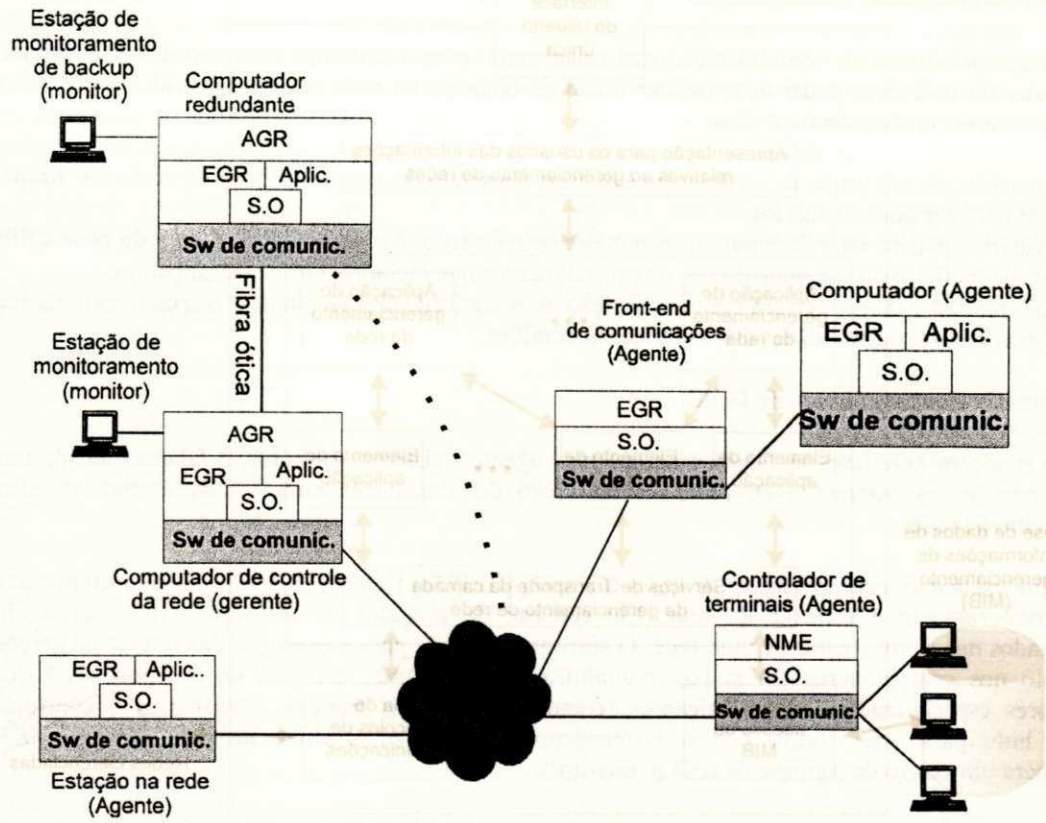


Figura 9.7 Elementos de um sistema de gerenciamento de redes [Sta93b]

Para que se mantenha uma alta disponibilidade para a função de gerenciamento da rede, dois ou mais computadores são instalados para os gerentes, de forma a se garantir a tolerância a falhas por meio da redundância destes equipamentos ([Byr94] [Cri91]). Consideramos o computador gerente como o coração do sistema de gerenciamento da rede, e como tal deve-se fornecer atenção redobrada (literalmente) ao mesmo. Estes computadores redundantes devem ser ligados por canais de alta velocidade ou por fibra-ótica, e no caso do computador primário falhar, o computador secundário deve assumir imediatamente o controle da rede.

9.3.1. A Arquitetura do Software de Gerenciamento

Não existe uma arquitetura formal para o software de gerenciamento da rede, portanto apresentamos uma visão genérica do que se poderia dizer ser uma arquitetura deste tipo ([JM90] [Her92] [Sch92c]). Podemos dividir tal software em três grandes categorias:

- software de apresentação para o usuário;
- software de gerenciamento da rede propriamente dito;
- software de suporte a comunicações e a banco de dados.

Na figura 9.8, descrevemos esta arquitetura.

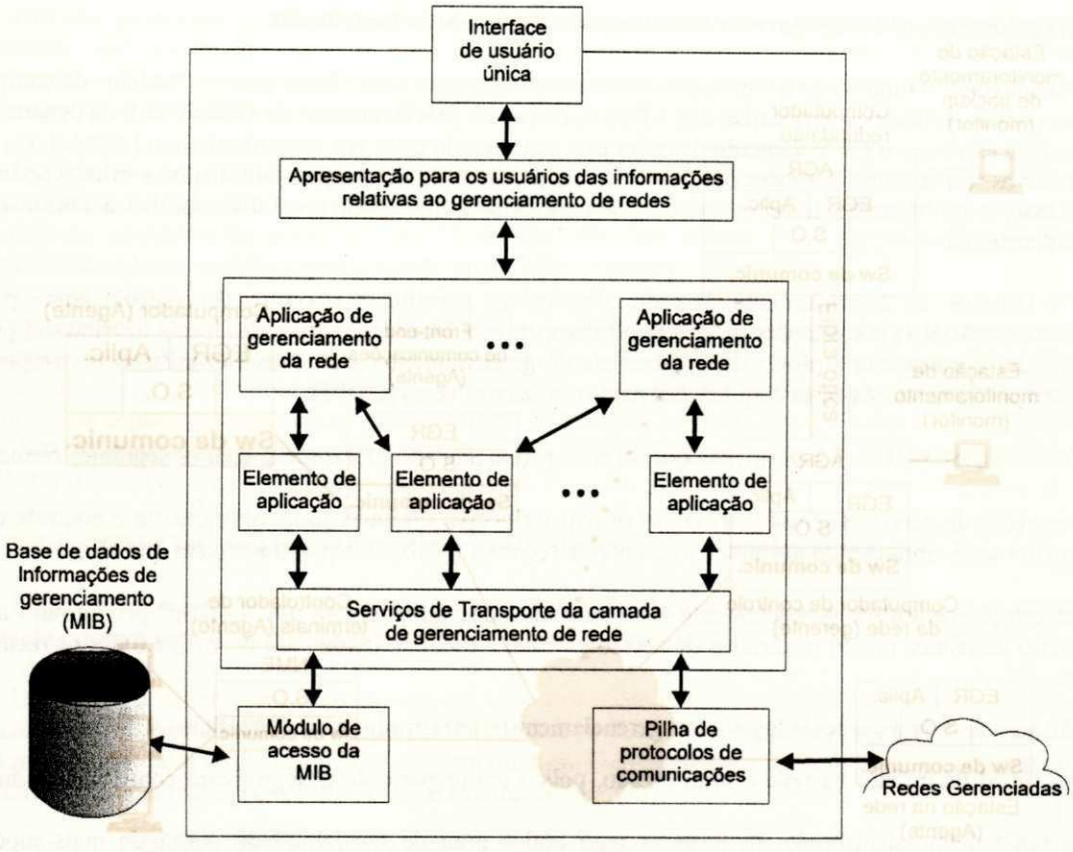


Figura 9.8. Visão genérica para uma arquitetura de gerenciamento de redes [JM90]

9.3.1.1. Software de Apresentação para o Usuário

As interações entre o operador da rede e o monitor ocorrem através de uma interface gráfica de usuário [Sey89] (ou GUI - Graphical User Interface). Podem eventualmente existir GUIs nos agentes, somente para testes ou estabelecimento de valores iniciais. A GUI deve ser unificada [JM90], para permitir o gerenciamento de ambientes heterogêneos com uma necessidade mínima para treinamento de pessoal. O software de apresentação para o usuário deve fornecer facilidades para organização, sumarização e simplificação das informações de gerenciamento, privilegiando apresentações gráficas ou em Tabelas.

9.3.1.2. O Software de Gerenciamento da Rede

Situa-se a este nível o software para a implementação de protocolos padrões como SNMP ou CMIP [Yem94]. Normalmente está dividido em três níveis: no primeiro nível temos as aplicações de gerenciamento de rede, que fornecem as funcionalidades necessárias para o gerenciamento das áreas funcionais: desempenho, falhas, configuração, segurança e contabilização. No segundo nível temos as aplicações elementares, que cuidam das rotinas mais triviais como providenciar alarmes ou sumarizar dados. No terceiro nível ficam os serviços de transporte de dados do protocolo, que fornecem comandos básicos como: busca de informações (get), estabelecimento de parâmetros (set) e outros.

9.3.1.3. Software de Suporte a Comunicações e a Banco de Dados

O software de gerenciamento de redes precisa acessar informações armazenadas nas MIBs locais e remotas. Ele o faz por meio do Software de Suporte a Comunicações e a Banco de Dados [JM90]. A MIB local em um agente contém informações que refletem a configuração e o comportamento do nó sob a responsabilidade deste agente. O gerente possui uma MIB local, que contém informações a seu próprio respeito e também informações sumarizadas a respeito dos agentes por ele controlados. O software de acesso à MIB inclui métodos de acesso a arquivos e controles de concorrência. No caso de um agente procurador, o software de acesso pode necessitar de um conversor de formatos (para converter dados em formatos proprietários para formatos padronizados, por exemplo). A comunicação com outros nós é suportada por famílias de protocolos de comunicações como TCP/IP ou OSI.

9.4. O Gerenciamento de Redes para os Ambientes da Computação Distribuída

Assim como o modelo de computação centralizado começa a ceder lugar para os modelos da computação distribuída, com as aplicações deslocadas dos CPDs (Centros de Processamento de Dados) para os departamentos espalhados pela empresa, o gerenciamento de redes está se tornando cada vez mais distribuído [AP94a]. Os fatores de sempre são os responsáveis por estes deslocamentos: a proliferação de estações de trabalho e micros poderosos e de baixo custo, a proliferação de redes locais, e a necessidade clara para o controle local e a otimização das aplicações distribuídas.

Os sistemas de gerenciamento de redes distribuídos substituem o centro de controle único proposto anteriormente (seção 9.3) por estações de trabalho interoperáveis localizadas em redes locais distribuídas por toda a empresa. Esta estratégia cria níveis departamentais de gerenciamento, com seus administradores responsáveis pela manutenção de redes, sistemas e aplicações para os usuários locais [Cas93].

Nestes casos, podemos sugerir uma estrutura de gerenciamento hierárquica, com os seguintes elementos:

- As estações de gerenciamento distribuídas devem ter acesso limitado ao monitoramento e controle da rede como um todo, abrangendo normalmente somente recursos locais ao departamento em questão;

- Uma estação central de gerenciamento (ou monitor), normalmente com um "backup" pronto para assumir em caso de falhas, possui os direitos de acesso globais, e está habilitada para gerenciar todos os recursos na rede.

Ao ser mantida a capacidade para um gerenciamento centralizado, vários benefícios se destacam:

- O overhead de tráfego na rede é minimizado, pois a maior parte do tráfego ficará confinado ao ambiente local;
- O gerenciamento distribuído ira fornecer uma maior grau de escalabilidade, tornando mais modular e simples a adição de novos elementos de gerenciamento;
- Elimina-se com este enfoque a fragilidade de um centro de controle único.

Na figura 9.9 , podemos examinar uma estrutura básica que vem sendo seguida pela maioria das implementações de sistemas de gerenciamento de redes disponíveis no mercado [Her92]. No topo encontramos os clientes de gerenciamento, que fornecem aos usuários acesso aos serviços de gerenciamento por meio de uma GUI.

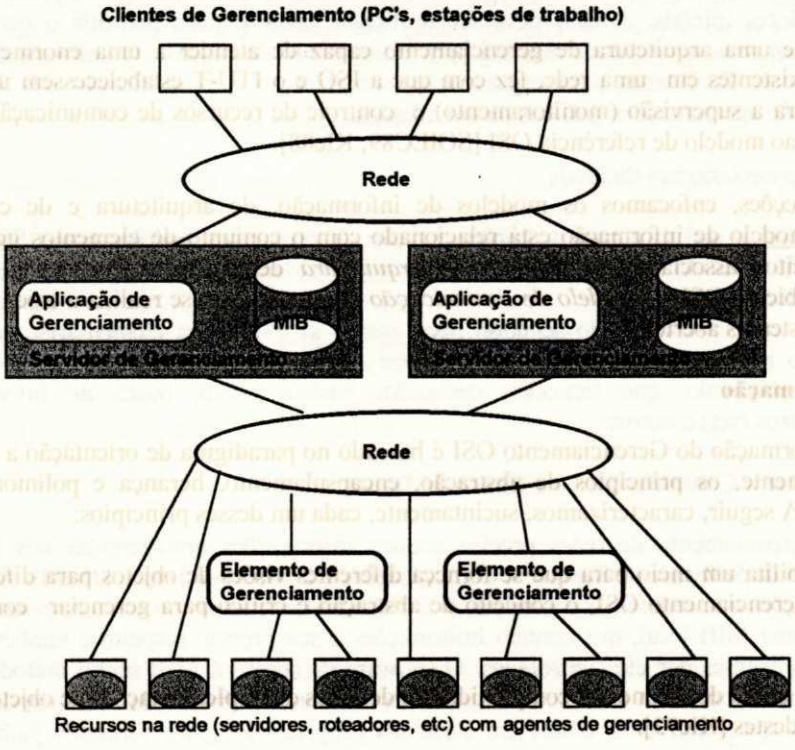


Figura 9.9 Uma visão do gerenciamento distribuído [Her92]

Um dos principais componentes de um sistema de gerenciamento distribuído são os servidores de gerenciamento, que cuidam de armazenar as informações de gerenciamento e executam a maioria do software de gerenciamento, e são sem sombra de dúvida o coração de todo o sistema de gerenciamento de redes. Cada servidor suporta uma série de aplicações e uma MIB [BRI93b].

Os agentes e os gerentes elementares comunicam-se diretamente com os recursos na rede. Os agentes - módulos de software que residem no recurso sendo gerenciado - coletam informações de gerenciamento e as enviam para os servidores de gerenciamento. O diálogo entre um agente e um servidor de gerenciamento é normalmente baseado em padrões como o SNMP ou o CMIP [Yem94].

Os agentes são fornecidos por fabricantes de roteadores, pontes, hubs, servidores de arquivos, sistemas operacionais e aplicações, ou em alguns casos podem ser desenvolvidos pelos prórpios usuários [cita]. O software para os sistemas de gerenciamento distribuído são fornecidos por grandes fabricantes como a IBM, DEC e HP. Alguns dos mais conhecidos sistemas de gerenciamento de redes no mercado são o SunNet Manager [Mal92a], da empresa Sun Microsystems, o Open View [BRI93b], da Hewlett-packard, e o EMA (Enterprise Management Architecture) [Cas93] da Digital Equipment Corporation.

9.5. Os Padrões OSI e Internet para Gerenciamento de Redes

Com a tendência de utilização de sistemas abertos [HM91], organismos de pesquisa e de padronização têm efetuado esforços para definir padrões para gerenciamento de redes.

Utilizamos a denominação *Gerenciamento OSI* para o conjunto de padrões para gerenciamento de redes estabelecido pela ISO e pelo ITU-T (criado em 1o. de março de 1993 substituindo o CCITT, o qual deixou de existir a partir de 28 de fevereiro de 1993); e a denominação *Gerenciamento Internet* para o conjunto de padrões para gerenciamento de redes estabelecido pela comunidade Internet.

O Gerenciamento OSI e o Gerenciamento Internet correspondem, respectivamente, ao padrão oficial (*de jure*) e ao padrão de indústria (*de facto*) para gerenciamento de redes.

Nas seções seguintes, caracterizamos o Gerenciamento OSI e o Gerenciamento Internet. Posteriormente, apresentamos algumas diferenças entre os mesmos.

9.6. O Gerenciamento OSI

A necessidade de uma arquitetura de gerenciamento capaz de atender a uma enorme diversidade de elementos gerenciáveis existentes em uma rede, fez com que a ISO e o ITU-T estabelecessem um conjunto de ferramentas e serviços para a supervisão (monitoramento) e controle de recursos de comunicação dentro de um ambiente OSI, em adição ao modelo de referência OSI [SOIEC89, Kle88].

Nas próximas seções, enfocamos os modelos de informação, de arquitetura e de comunicação do Gerenciamento OSI. O modelo de informação está relacionado com o conjunto de elementos gerenciados, seus relacionamentos e conceitos associados. O *modelo de arquitetura* destaca os componentes envolvidos no gerenciamento de um ambiente OSI. O *modelo de comunicação* descreve como se realiza a troca de informações de gerenciamento entre sistemas abertos.

9.6.1. O Modelo de Informação

O modelo de informação do Gerenciamento OSI é baseado no paradigma de orientação a objetos (OO), o qual utiliza, intrinsecamente, os princípios de abstração, encapsulamento, herança e polimorfismo [Sta93b, Yem93, Yem94, Pyl94]. A seguir, caracterizamos, sucintamente, cada um desses princípios:

Abstração: possibilita um meio para que se forneça diferentes visões de objetos para diferentes usuários [Pyl94]. No mundo do gerenciamento OSI, o conceito de abstração é crítico para gerenciar complexidades do ambiente OSI.

Encapsulamento: cuida de esconder a complexidade e detalhes de implementações de objetos, assegurando a integridade operacional destes [Kle93].

Herança: está associada ao conceito de classes de objetos. Objetos que possuem propriedades similares podem ser agrupados em classes. Uma classe de objetos é um modelo ou *"template"*, que define os métodos e as variáveis a serem incluídas em um tipo particular de objeto. Uma instância de um objeto é um objeto de fato (real) que inclui as características da classe que o define. Uma instância contém valores para as variáveis definidas na classe de objeto. Uma classe pode ser uma subclasse de outra classe (sua superclasse). Uma subclasse herda todas as propriedades de sua superclasse, de maneira irrestrita, independentemente da necessidade ou não destas propriedades. A estas subclasses podem ser adicionadas novas propriedades. A Herança possibilita, então, que uma nova classe de objetos seja definida em termos de uma classe previamente existente [Sta93b].

Polimorfismo: permite a reutilização de instâncias de objetos na criação de novas classes de objetos. No gerenciamento OSI, polimorfismo refere-se à habilidade de uma instância de objeto, que é membro de uma classe, ser gerenciada como uma instância de objeto de uma ou várias classes. Assim, polimorfismo é uma propriedade inerente a uma instância de objeto e não a uma classe de objetos [Kle93]. No gerenciamento OSI, polimorfismo é um conceito muito importante para o controle de versões.

Na concepção do Gerenciamento OSI, elementos gerenciados correspondem a *objetos gerenciados*. Neste trabalho, o termo objeto gerenciado é empregado no mesmo sentido de uma *instância de objeto gerenciado*. Assim, objetos gerenciados correspondem a visões conceituais de recursos físicos ou lógicos que são gerenciados ou que podem ser submetidos a atividades de gerenciamento. Objetos gerenciados são definidos em termos de seus atributos, operações a que podem ser submetidos, notificações que podem emitir, e suas relações com outros objetos gerenciados [Sta93b, Pyl94]. Os elementos de dados reais contidos em um objeto gerenciado são chamados atributos. Cada atributo corresponde a uma característica do recurso que o objeto gerenciado representa, tais como suas características operacionais e estado corrente.

Operações a que podem ser submetidos os objetos gerenciados definem as ações de gerenciamento que podem ser executadas nos atributos de um objeto (operações orientadas a atributos) ou ações que se aplicam a um objeto como um todo (operações orientadas a objetos). Exemplos de operações orientadas a atributos são: recuperar o valor de um atributo (*Get*); substituir o valor de um atributo (*Replace*) e estabelecer o valor de um atributo (*Set*). Exemplos e operações orientadas a objetos são: criar (*Create*), remover (*Delete*) e acionar (*Action*) objetos gerenciados. A operação Action requer que o objeto gerenciado execute uma ação especificada e indique o seu resultado.

Notificações são informações emitidas assincronamente pelos objetos gerenciados. Os objetos gerenciados podem emitir notificações quando algum evento (condição de exceção) externo ou interno ocorrer.

O modelo de informação do Gerenciamento OSI inclui um número de representações de relações genéricos (está-contido-em, é-parte-de, é-cópia-de) que possibilita a definição explícita de relacionamentos entre objetos gerenciados. O uso de atributos de relacionamento na modelagem de objetos gerenciados é similar aos mecanismos usados em modelos de banco de dados em rede [Yem94, Ull88].

Objetos gerenciados estende o conceito de classe do modelo de dados OO por incluir notificações de eventos [Yem94]. Em modelos de dados OO tradicionais, existem interações síncronas entre um objeto e seus usuários (programas). No Gerenciamento OSI, eventos podem ocorrer independentemente e assincronamente.

9.6.1.1. MIB e o Conceito de MIT

Informações sobre objetos gerenciados são armazenadas na MIB. Como dissemos anteriormente, o termo MIB é conceitual, ou seja, não importa que tipo de armazenamento físico (memória principal, arquivos, base de dados, etc) é empregado no armazenamento das informações de gerenciamento. Tanto o armazenamento como a representação de informações na MIB não estão sujeitas à padronização.

As informações na MIB podem ser vistas como uma coleção de entradas (objetos), sendo cada uma destas entradas formada por uma lista de atributos com seus valores associados para um objeto gerenciado em particular. Estas entradas estão estruturadas hierarquicamente em uma árvore de informação de gerenciamento, chamada **MIT** (*Management Information Tree*) [Kle88]. Objetos gerenciados podem ser adicionados ou removidos desta árvore dinamicamente [Yem93, Yem94]. A Figura 9.10 exemplifica a estrutura de uma MIT. Cada retângulo representa uma entrada na MIB.

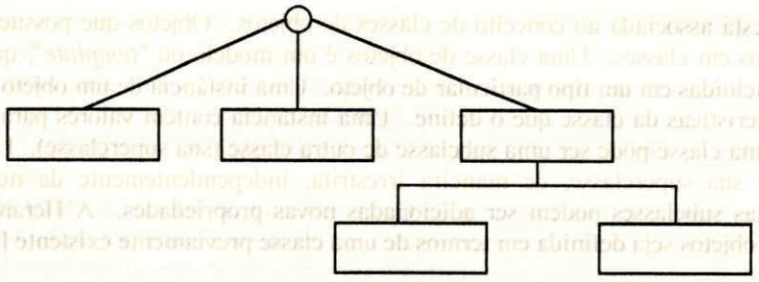


Figura 9.10 Estrutura de uma MIT [Yem94]

A MIT organiza objetos segundo uma hierarquia de *containment* (hierarquia de nomeação) [Yem94]. Esta hierarquia referência de forma única (não ambígua) um objeto gerenciado e representa também a estrutura da MIB. Se a MIB for, por exemplo, um banco de dados, a MIT representa o esquema deste banco de dados.

Objetos gerenciados na MIT são identificadas por atributos de nomeação denominados **RDN** (*nome característico relativo - Relative Distinguished Name*) e **DN** (*nome característico - Distinguished Name*). Cada objeto gerenciado inclui atributos que servem como um RDN. RDNs identificam unicamente um determinado objeto gerenciado dentre objetos irmãos, em um mesmo nível da MIT, e filhos de um mesmo pai, o qual é um objeto em um nível superior na MIT. A concatenação de RDNs do topo da hierarquia da árvore (nó raiz) a um determinado nó é chamado DN. Sendo assim, um DN identifica um nome único e completo para um objeto gerenciado. A Figura 9.11 e a tabela 9.1 exemplificam, respectivamente, uma hierarquia de nomeação, e os conceitos de RDN e DN.

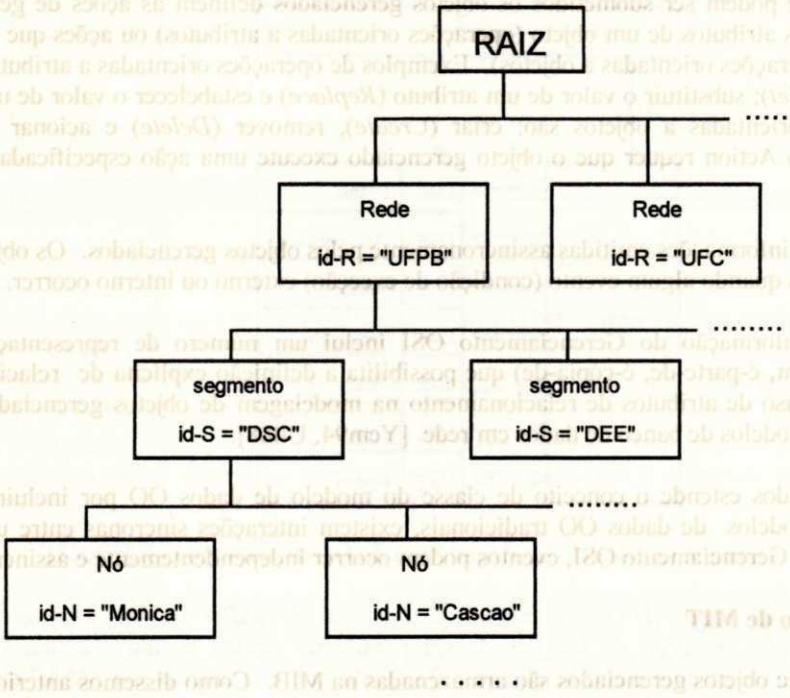


Figura 9.11 Exemplo de uma Hierarquia de Nomeação

Classe de Objeto	RDN	DN
Raiz	{}	{}
Rede	{id-R="UFPB"}	{id-R="UFC"}
Segmento	{id-S="DSC"}	{id-S="DEE"}
Nó	{id-N="Monica"}	{id-N="Cascao"}

Tabela 9.1. Nomes de Instâncias de Objetos Gerenciados referenciados no exemplo da figura 9.11

9.6.1.2. O Conceito de SMI

Conceitos do modelo de informação do Gerenciamento OSI, vistos anteriormente, são introduzidos no documento ISO10165-1, denominado *Management Information Model*, e detalhados no documento ISO10165-4, denominado *Guidelines for Definition of Managed Objects* **(GDMO)** [SOIEC91b, SOIEC91c].

Estes documentos tratam da Estrutura de Gerenciamento de Informação **SMI** (*Structure of Management Information*), que descreve o cenário no qual objetos gerenciados podem ser definidos. Este cenário inclui a definição do conjunto de operações que pode ser realizado sobre os objetos gerenciados e o comportamento destes objetos mediante a execução destas operações.

Na SMI, objetos gerenciados são definidos como estruturas de dados usando a notação de sintaxe abstrata **ASN.1** (*Abstract Syntax Notation One*). O GDMO apresenta extensões à linguagem ASN.1 no tratamento da sintaxe das definições de informações gerenciadas. Uma nova estrutura de linguagem - *template* - é introduzida para combinar definições [Yem94]. Como dissemos anteriormente, uma classe de objetos gerenciados é um modelo ou uma *template* para objetos gerenciados que partilham os mesmos atributos, as mesmas notificações, e as mesmas operações de gerenciamento [Sta93b]. As *templates* resumem os elementos que devem ser incluídos em uma definição de objeto gerenciado e definem também as ferramentas de notação que são recomendadas para uso com esta definição.

9.6.2. O Modelo de Arquitetura

O modelo de arquitetura do Gerenciamento OSI [SOIEC89], ilustrado na figura 9.12 , é composto pelos seguintes elementos: SMAP, SMAE, LME e MIB.

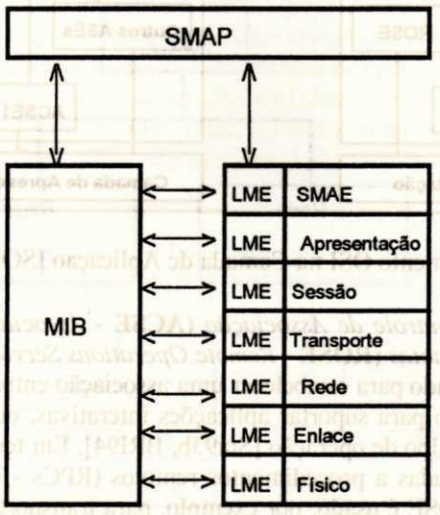


Figura 9.12 Modelo de Arquitetura do Gerenciamento OSI [Kle88]

Um *Processo de Aplicação de Gerenciamento de Sistemas* (**SMAP** - *Systems Management Application Process*) é responsável pela execução de funções de gerenciamento dentro de um sistema. Um SMAP pode desempenhar tanto o papel de gerente como de agente. O sistema que implementa um SMAP que desempenha o papel de gerente (SMAP-gerente) é denominado **sistema gerente**, e o sistema que implementa um SMAP que desempenha o papel de agente (SMAP-agente) é denominado **sistema gerenciado**. Um SMAP-agente pode fornecer a um SMAP-gerente uma visão de seus objetos gerenciados. O SMAP-gerente pode, então, emitir operações de gerenciamento sobre tais objetos. No caso do SMAP-gerente emitir operações de gerenciamento, o SMAP-agente as recebe e as aplica aos seus objetos gerenciados. O SMAP-agente identifica seus objetos gerenciados de acordo com uma hierarquia de nomeação disposta na MIT.

Uma *Entidade de Aplicação para Gerenciamento de Sistemas* (**SMAE** - *Systems Management Application Entity*) fornece mecanismos para monitoramento e controle dos objetos gerenciados de um sistema onde tais objetos podem pertencer a uma ou várias camadas. Além disso, SMAEs são responsáveis pela troca de informações de gerenciamento entre sistemas gerente e gerenciado.

Uma *Entidade de Gerenciamento de Camada* (**LME** - *Layer Management Entity*) é responsável pela execução de funções de gerenciamento específicas de cada camada.

9.6.3. O Modelo de Comunicação

O modelo de comunicação do Gerenciamento OSI está relacionado com o modelo de arquitetura apresentado na seção anterior.

Como dissemos na seção anterior, SMAEs são responsáveis pela troca de informações de gerenciamento entre sistemas gerente e gerenciado. Diálogos entre SMAEs são realizados com ajuda de um protocolo da camada de aplicação, denominado *Protocolo de Informação de Gerenciamento Comum* (**CMIP** - *Common Management Information Protocol*).

Como qualquer entidade da camada de aplicação, uma SMAE pode ser definida como um conjunto de *Elementos de Serviço de Aplicação* (**ASE** - *Application Service Elements*). A Figura 9.13 ilustra ASEs em SMAEs.

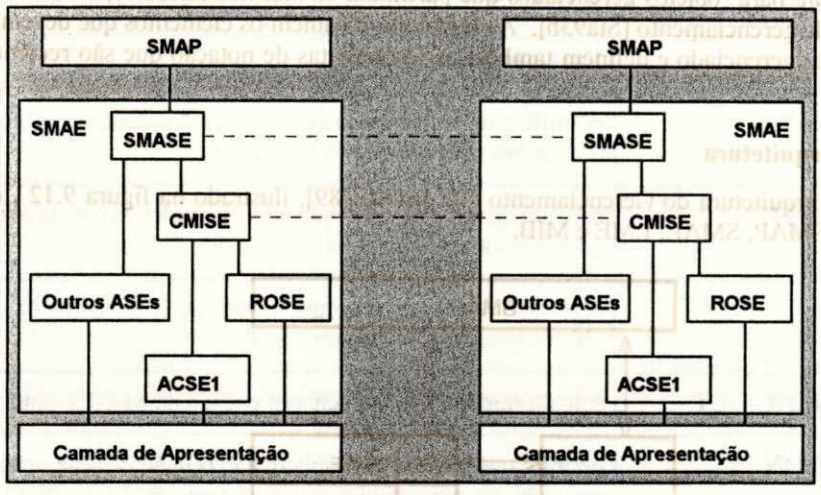


Figura 9.13 Gerenciamento OSI na Camada de Aplicaçao [SOIEC91b]

O *Elemento de Serviço de Controle de Associação* (**ACSE** - *Association Control Service Element*) e o *Elemento de Serviço de Operações Remotas* (**ROSE** - *Remote Operations Service Element*) são utilizados por uma variedade de aplicações. O ACSE é usado para estabelecer uma associação entre duas entidades de aplicação pares ("peers") [Ram93]. O ROSE é voltado para suportar aplicações interativas, ou seja, aplicações que requisitam a outras aplicações a execução de algum tipo de operação [Sta93b, BRI94]. Em termos de programação, um exemplo comum deste mecanismo são as chamadas a procedimentos remotos (RPCs - Remote Procedure Calls). No caso particular do gerenciamento OSI, o ROSE é usado, por exemplo, para transportar operações de gerenciamento e o resultado de operações realizadas entre sistemas agentes e gerentes [BRI94].

O *Elemento de Serviço de Aplicação de Gerenciamento de Sistema* (**SMASE** - *Systems Management Application Service Element*) e o *Elemento de Serviço de Informação de Gerenciamento Comum* (**CMISE** - *Commom Management Information Service Element*) são específicos para o gerenciamento de redes.

O SMASE implementa funções de gerenciamento de sistemas (**SMF** - *Systems Management Functions*) que são comuns a várias áreas funcionais de gerenciamento [Sta93b]. Além disso, o SMASE especifica a informação de gerenciamento a ser trocada entre SMAEs. Em particular, o SMASE define a semântica e a sintaxe abstrata das informações transferidas nas *Unidades de Dados do Protocolo de Aplicação de Gerenciamento* (**MAPDUs** - *Management Application Protocol Data Unit*).

Os serviços de comunicação usados pelo SMASE podem ser prestados pelo CMISE ou por outros ASEs, como o de **FTAM** (*File Transfer, Access and Management*) e o **TP** (*Transaction Processing*).

O CMISE especifica o serviço e os procedimentos usados para a transferência das *Unidades de Dados do Protocolo de Informação de Gerenciamento Comum* (**CMIPDUs** - *Common Management Information Protocol Data Unit*) e provê um meio de troca de informações para as operações e notificações de gerenciamento. Para realizar suas funções, o CMISE utiliza os serviços dos ASEs ACSE e ROSE.

Unidades de Dados (**DU** - *Data Units*) são trocadas entre SMAEs utilizando um protocolo de transporte orientado a conexão como o TP4 (Transport Protocol class 4) da ISO [Yem93]. Outros protocolos de transporte orientados a conexão, como o TCP (Transmission Control Protocol), também podem ser utilizados [MR91, Yem94].

9.6.3.1. CMIS e CMIP

O CMISE é especificado em duas partes: **CMIS** (*Common Management Information Services*) e **CMIP** (*Common Management Information Protocol*) [Sta93b, Yem93, Ram93, Yem94]. Enquanto o CMIS [SOIEC91d] define os serviços para operações de gerenciamento, o CMIP [SOIEC90] define procedimentos para transmitir informações de gerenciamento e a sintaxe para os serviços de gerenciamento do CMIS.

CMIS (Common Management Information Services)

Os serviços CMIS são especificados em termos de primitivas que podem ser vistas como comandos ou chamadas a procedimentos. A Tabela 9.2 relaciona cada serviço do CMIS juntamente com sua funcionalidade.

Serviço	Funcionalidade
M-CREATE	permite que um *sistema aberto* (**s.a.**) ***A*** requisite a criação de um (e somente um) objeto em um s.a. ***B***
M-DELETE	permite que um s.a. ***A*** requisite a remoção de um ou mais objetos gerenciados em um s.a. ***B***
M-GET	permite que um s.a. ***A*** recupere valores de atributos de um ou mais objetos gerenciados em um s.a. ***B***
M-CANCEL-GET	solicita que um s.a. cancele um pedido de M-GET anterior
M-ACTION	permite que um s.a. ***A*** requisite a execução de uma ação em um ou mais objetos gerenciados de um sistema ***B***
M-EVENT-REPORT	permite que um s.a. ***A*** reporte um evento para um s.a ***B***
M-SET	permite que um s.a. ***A*** requisite a modificação de valores de atributos de um ou mais objetos gerenciados em um s.a. ***B***

Tabela 9.2 Serviços CMIS [SOIEC90]

Os serviços do CMIS podem ser confirmados ou não-confirmados. Se o serviço for no modo não-confirmado nenhuma resposta será gerada pelo receptor da primitiva. Serviços como o M-SET, M-ACTION e o M-EVENT-REPORT podem ser requisitados no modo confirmado ou não-confirmado [Ram93].

As primitivas do CMIS são mapeadas em operações de gerenciamento que acessam a MIB. A Tabela 9.3 ilustra correspondências entre primitivas do CMIS e operações da MIB.

Primitivas CMISE	Operações da MIB
M-CREATE	Create
M-DELETE	Delete
M-CANCEL-GET	Remove
M-ACTION	Action
M-EVENT-REPORT	Notificação
M-GET	Get
M-SET	Set

Tabela 9.3 Primitivas CMIS e operações da MIB [Ram93]

As primitivas M-GET, M-SET, M-ACTION e M-DELETE podem especificar operações em múltiplos objetos. A seleção de múltiplos objetos gerenciados envolvem três conceitos: escopo, filtro e sincronização [Sta93b, Ram93].

O *mecanismo de escopo* é utilizado para identificar objetos gerenciados que são candidatos a executar uma particular operação. O escopo é baseado na especificação de um objeto gerenciado base (**BMO** - *Base Managed Object*). O BMO é o ponto inicial da seleção de um ou mais objetos gerenciados. Após objetos gerenciados serem identificados pelo mecanismo de escopo, estes devem ser submetidos ao mecanismo de filtro.

O mecanismo de aplicação de critérios em objetos selecionados é conhecido como **filtro**. Critérios são especificados usando expressões lógicas que definem asserções sobre atributos.

O conceito de sincronização está relacionado com a execução de uma operação em vários objetos. No Gerenciamento OSI, existe a sincronização atômica e a sincronização de melhor esforço. Na sincronização atômica, uma operação é somente executada caso todos os objetos selecionados possam executar a operação. Na sincronização de melhor esforço, que é o default, uma operação é executada por todos os objetos que podem executar a operação, mesmo que alguns objetos selecionados não sejam capazes de executar tal operação.

CMIP (Common Management Information Protocol)

O *Protocolo de Informação de Gerenciamento Comum* (**CMIP** - *Common Management Information Protocol*) é um protocolo orientado a conexão que é definido em termos de um conjunto de unidades de dados de protocolo (CMIPDUs) no qual as primitivas CMIS são mapeadas. Essencialmente, o CMIP oferece um formato comum para a transferência de informações de gerenciamento entre entidades pares.

Para executar a transferência de CMIPDUs, o CMIP utiliza-se de serviços dos ASEs ROSE e ACSE e de serviços da camada de apresentação. A sintaxe utilizada para especificar elementos do protocolo CMIP é a sintaxe abstrata padronizada ASN.1 [Bou90]. Detalhes sobre o protocolo CMIP podem ser encontrados no documento ISO9596 ([SOIEC90]) da ISO.

9.7. O Gerenciamento Internet

O Gerenciamento Internet foi desenvolvido pela comunidade Internet em decorrência da necessidade de se gerenciar a rede Internet, composta por diferentes topologias de redes interligadas.

A filosofia de gerenciamento de redes apresentada pela comunidade Internet em 1988 é baseada no modelo conhecido como **SNMP** (*Simple Network Management Protocol*), que teve a sua origem em um protocolo para monitoramento de gateways, o **SGMP** *Simple Gateway Management Protocol*). Obviamente, tal evolução implicou na introdução de mudanças no modelo original, que passou a utilizar alguns conceitos do Gerenciamento OSI [Ros91, Sta93b, BRI94].

Devido a algumas deficiências existentes no modelo SNMP, a comunidade Internet propôs, no começo de 1993, uma nova versão para este modelo. O modelo SNMP passou a ser referenciado como modelo SNMPv1 (versão 1) e sua nova versão, como modelo SNMPv2 (versão 2).

9.7.1. O Modelo SNMPv1

O modelo SNMPv1 foi projetado para gerenciar o funcionamento de redes baseadas no conjunto de protocolos TCP/IP. A seguir, examinamos aspectos das informações de gerenciamento e da arquitetura do modelo SNMPv1.

9.7.1.1. Informações de Gerenciamento

Conceitos relacionados com as informações de gerenciamento do modelo SNMPv1 são semelhantes aos conceitos do modelo de informação do Gerenciamento OSI.

Como o Gerenciamento OSI, as informações de gerenciamento do modelo SNMPv1 relacionam-se com o conceito de *objetos gerenciados*, o qual corresponde ao conceito de elementos gerenciados, apresentado na seção anterior.

Na concepção do modelo SNMPv1, um objeto gerenciado é uma variável de dados. Cada objeto gerenciado possui um valor e é definido através de um *tipo de objeto*. Ao contrário do Gerenciamento OSI, o modelo SNMPv1 não utiliza os conceitos de classes de objetos e seus respectivos atributos.

Informações sobre objetos gerenciados são também armazenadas na MIB. A comunidade Internet definiu duas MIBs para o modelo SNMPv1: MIB I e MIB II, que serão vistas adiante com maiores detalhes.

Objetos gerenciados estão estruturados hierarquicamente em uma árvore de informações de gerenciamento também denominada MIT. Na identificação de objetos gerenciados, é utilizada a hierarquia de registro, especificada segundo as regras definidas pela notação ASN.1. De acordo com esta hierarquia, cada nó da MIT está associado a um número determinado por uma autoridade de registro (por exemplo, ISO e ITU-T). Desta maneira, cada objeto da MIB é identificado por uma seqüência de números, cada um correspondente a um nó. O modelo SNMPv1 não define nenhuma hierarquia de nomeação, como no Gerenciamento OSI.

A estrutura de informação de gerenciamento (SMI) do modelo SNMPv1 estabelece a forma na qual a MIB pode ser definida e construída, ou seja, ela identifica os tipos de dados que podem ser usados na MIB e como objetos gerenciados podem ser representados e nomeados.

9.7.1.2. As MIBs do Modelo SNMPv1

Como dissemos anteriormente, a comunidade Internet definiu duas MIBs para o modelo SNMPv1: MIB I e MIB II.

Objetos gerenciados na MIB estão subdivididos em grupos. A MIB I contém 114 objetos gerenciados distribuídos em oito grupos, denominados: System, Interface, Address Translation, IP, ICMP, TCP, UDP e EGP. A MIB II criou os grupos Transmission e SNMP, e adicionou outros objetos aos demais grupos, conforme mostrado na tabela 9.4 .

Grupo de Variáveis	MIB I no. objetos	MIB II no. objetos
System	3	7
Interfaces	22	23
Add.Translation	3	3
IP	33	38
ICMP	26	26
TCP	17	19
UDP	4	7
EGP	6	18
Transmission	0	0
SNMP	0	30
Total	**114**	**171**

Tabela 9.4 Quantidade de Objetos da MIB I e MIB II [Kas92]

A seguir caracterizamos a funcionalidade destes grupos:

O grupo **System** é usado para representar informações sobre o sistema onde está instalado o conjunto de protocolos em execução. Estas informações incluem, por exemplo, a identificação do fabricante do sistema e a versão do software de comunicação.

O grupo **Interface** está relacionado a informações sobre interfaces de rede como tipo, taxa de transmissão, endereço, estado operacional, número de pacotes recebidos e transmitidos, dentre outras.

O grupo **Address Translation** corresponde a tabelas de mapeamento, entre o endereço IP (Internet Protocol) e o endereço de sub-rede do sistema.

Os grupos **IP, ICMP, TCP, UDP e EGP** armazenam informações, respectivamente, sobre os protocolos IP (Internet Protocol), ICMP (Internet Control Message Protocol), TCP (Transmission Control Protocol), UDP (User Datagram Protocol) e EGP (Exterior Gateway Protocol).

O grupo **Transmission** relaciona informações sobre os esquemas de transmissão e os protocolos de acesso em cada interface do sistema.

O grupo **SNMP** armazena informações relacionadas com a implementação e a execução do protocolo SNMPv2 no sistema.

A Figura 9.14 ilustra a árvore (MIT) da MIB II estruturada segundo a hierarquia de registro. Vale ressaltar que cada nó da árvore tem associado um valor inteiro positivo. A seqüência desses valores da raiz até um nó desejado identificam um objeto. Desta maneira, todos os objetos da MIB II são registrados com nomes cujo prefixo é **1.3.6.1.2.1.** Um objeto do grupo IP, por exemplo, tem como prefixo a seqüência **1.3.6.1.2.1.4.**

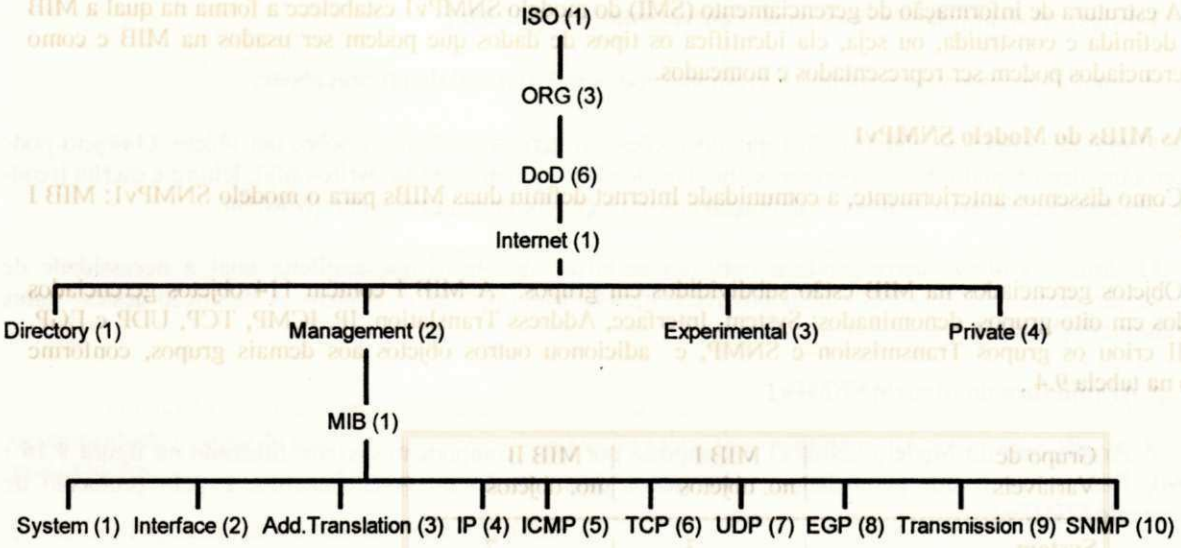


Figura 9.14: A Árvore da MIB II [Wal94]

Em contraste com o Gerenciamento OSI, as MITs das MIB I e II são estáticas, ou seja, não permitem a criação ou remoção de objetos em tempo de execução.

Fornecedores podem acrescentar informações patenteadas, denominadas extensões, na MIB. Entretanto, sempre que definem novos objetos, estes limitam a interoperabilidade de seus produtos. SGRs de outros fornecedores nem sempre conseguem recuperar informações destes objetos.

9.7.1.3. Tipo de Objeto

Um objeto gerenciado no modelo SNMPv1 é definido através do conceito de tipo de objeto. A definição de tipo de objeto contém cinco campos (Figura 9.15) [MR88a]: nome textual com respectivo identificador de objeto (OBJECT IDENTIFIER); uma sintaxe ASN.1; a definição da semântica associada ao objeto; o tipo de acesso do objeto; e o estado (status) do objeto.

OBJECT
sysDescr {system 1}
Syntax:
OCTET STRING
Definition:
Descrição textual para a entidade.
Access:
read-only.
Status:
mandatory.

Figura 9.15 Definição do Objeto sysDescr [MR88]

O *nome textual do objeto* (*sysDescr*, na figura 9.15 acima) corresponde ao nome mnemônico do tipo de objeto.

O *identificador de objeto* corresponde a uma seqüência de inteiros positivos, que percorre a árvore da MIB (MIT), da raiz ao nó desejado, para identificar o objeto. Em outras palavras, o identificador de objeto reflete a hierarquia de objetos gerenciados na MIB. Esta seqüência é decorrente da utilização do conceito de hierarquia de registro especificada pela notação ASN.1. O objeto gerenciado (sysDescr) está inserido na sub-árvore System da figura 9.15. Desta forma, sysDescr tem como prefixo a seqüência 1.3.6.1.2.1.1. Precisamente, o identificador do objeto sysDescr é 1.3.6.1.2.1.1.**1.0**. O último 1 corresponde ao valor inteiro positivo associado ao nó sysDescr e o zero indica que o objeto é terminal, ou que é uma instância única de sysDescr [Kas92, GL92].

A *sintaxe do objeto* especifica o tipo de dado que modela o objeto. São permitidos os seguintes tipos de dados: básicos (INTEGER, OCTET STRING, OBJECT IDENTIFIER, NULL, SEQUENCE e SEQUENCE OF); definidos (*NetworkAddress, IpAddress, Counter, Gauge, TimeTicks e Opaque*); e os construídos ou compostos (listas e Tabelas). A definição dos tipos citados encontramos em [MR88b].

A *semântica associada ao objeto* corresponde à descrição textual do próprio objeto.

O *tipo de acesso do objeto* indica que operações um gerente pode fazer sobre um objeto. O objeto pode pertencer a um dentre quatro tipos: somente leitura (read-only), somente escrita (write-only), leitura e escrita (read-write) e não acessível (not-accessible). A definição dos tipos citados encontramos em [MR88b].

O *status do objeto* corresponde a parte da definição do objeto que explícita qual a necessidade de implementação deste. Existem três status possíveis: mandatório, opcional e obsoleto. A definição destes status encontramos em [MR88b].

9.7.1.4. A Arquitetura do Modelo SNMPv1

A Arquitetura do Modelo SNMPv1 é composta por três componentes, como ilustrado na figura 9.16 : [CFSD90, Ros91] vários nós gerenciados; pelo menos uma estação de gerenciamento; e pelo protocolo de gerenciamento SNMPv1.

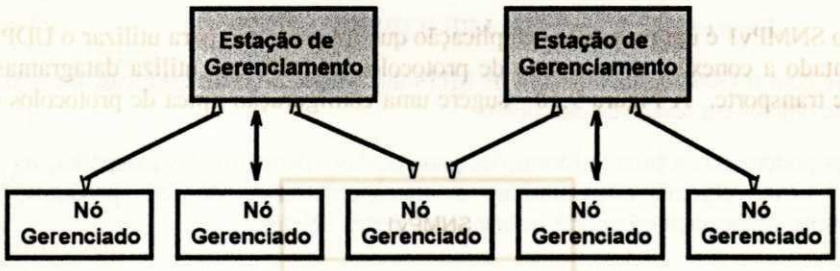


Figura 9.16 A arquitetura do Modelo SNMPv1 [CFSD90]

Nós gerenciados correspondem a sistemas hospedeiros, gateways, dispositivos intermediários (pontes, hubs, modens, multiplexadores) que implementam agentes.

Estações de gerenciamento correspondem a sistemas hospedeiros que monitoram (supervisionam) o estado da rede, coletando, periodicamente, informações de cada nó gerenciado. Para as estações de gerenciamento, cada nó gerenciado é visto como um conjunto de objetos gerenciados. Lendo-se valores de variáveis, um nó gerenciado é monitorado; mudando-se valores de variáveis, um nó gerenciado é controlado.

Todo SGR baseado no modelo de gerenciamento SNMPv1 deve verificar o seguinte axioma [CFSD90, Ros91]: "o impacto da adição de agentes em nós gerenciados deve ser mínimo". Neste contexto, os nós gerenciados devem executar um gerenciamento simples, como estabelecer e obter valores de variáveis, que utilize poucos recursos de sistema (CPU, memória, disco). A maior parte do processamento de gerenciamento da rede deve ser executado pelas estações de gerenciamento.

O relacionamento entre estações de gerenciamento e nós gerenciadas não é fixo [Ros91]. Uma estação de gerenciamento pode gerenciar vários nós gerenciados; um nó gerenciado pode ser gerenciado por várias estações de gerenciamento.

Nós gerenciados podem implementar ou não o protocolo de gerenciamento SNMPv1. Para gerenciar um nó que não implementa o protocolo de gerenciamento SNMPv1, é necessário um processo intermediário, chamado **agente procurador** (*proxy agent*), que traduza as operações da estação de gerenciamento em operações reconhecíveis pelo nó e vice-versa. Na prática, um agente procurador é instalado em um componente que implementa os protocolos de transporte da estação de gerenciamento e do nó que não implementa o protocolo SNMPv1.

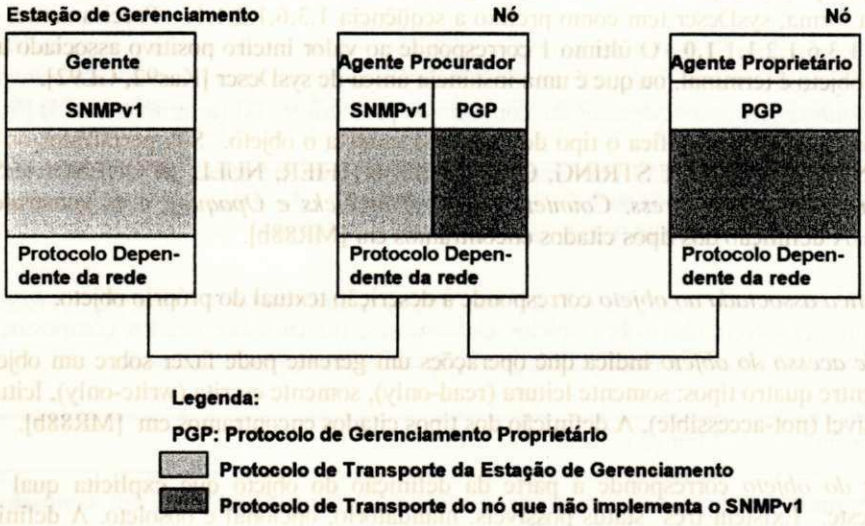


Figura 9.17. O Conceito de "Proxy" (Procurador) revisitado [Ros91]

O modelo SNMPv1 consiste, tipicamente, em um esquema centralizado, onde estações de gerenciamento desempenham o papel de gerente e os demais componentes da rede desempenham o papel de agente ou de agentes procuradores.

9.7.1.5. O Protocolo de Gerenciamento SNMPv1

O protocolo SNMPv1 é um protocolo de aplicação que foi projetado para utilizar o UDP (um protocolo de transporte não orientado a conexão, do conjunto de protocolos TCP/IP, que utiliza datagramas não confiáveis), como mecanismo de transporte. A Figura 9.18 sugere uma configuração típica de protocolos de suporte para o protocolo SNMPv1.

Figura 9.18 Uma Configuração Típica de Protocolos de Suporte para o Protocolo SNMPv1

Particularmente, o protocolo SNMPv1 caracteriza-se por:

- implementar as técnicas de monitoramento (polling) e relato de eventos. No primeiro caso, um gerente envia uma mensagem de protocolo referente a uma operação de gerenciamento a um agente e recebe dele outra mensagem contendo o resultado da operação executada. No segundo caso, o próprio agente envia mensagens assíncronas denominadas *traps;*

- O modelo SNMPv2 admite a existência de um gerenciamento distribuído, com estações de gerenciamento configuradas para exercer o papel de gerentes e agentes, e com a possibilidade de comunicação entre gerentes para a troca de informações de gerenciamento. Cada gerente pode administrar diretamente um conjunto de agentes, e quando o número de agentes cresce ao ponto de causar problemas relativos a seu gerenciamento, uma estação de gerenciamento pode delegar a tarefa de gerenciamento a gerenciadores intermediários. O gerenciador intermediário exerce o papel de gerente para monitorar e controlar os agentes sob sua responsabilidade e exerce o papel de agente para enviar e receber informações de controle de uma estação de gerenciamento hierarquicamente superior [Sta93b, BRI94].
- Para o modelo SNMPv2, a comunidade Internet definiu três MIBs que oferecem suporte a um esquema distribuído de gerenciamento. A MIB SNMPv2 contém informações relativas ao próprio protocolo SNMPv2. A *MIB Gerente-Gerente* (**M2M** - *Manager-to-Manager*) foi projetada, especialmente, para suportar a arquitetura de gerenciamento distribuída. Em essência, ela contém informações relacionadas com as trocas de informações entre gerentes. A *MIB de Segurança* contém informações relativas à segurança, identificando e definindo autorizações para cada relacionamento entre um gerente e um agente.
- O protocolo SNMPv1 só permite a comunicação entre um gerente e um agente. No modelo SNMPv2, o protocolo SNMPv2 permite a comunicação entre gerentes.
- A funcionalidade do protocolo SNMPv2 é alcançada através das operações *get-request, get-next-request, get-response, set-request e trap*, citadas no protocolo SNMPv1, e das operações *get-bulk-request e inform-request*. A Tabela 9.6 ilustra a funcionalidade das operações de gerenciamento próprias do SNMPv2.

Operação	Função
get-bulk-request	permite a recuperação de um conjunto de variáveis da MIB através de uma única troca de PDU
inform-request	possibilita que um gerente encaminhe informações de gerenciamento para outro gerente, suportando assim, um esquema distribuído

Tabela 9.6 Operações de Gerenciamento Próprias do SNMPv2 [SS94]

- Na recuperação de um conjunto de variáveis, a operação *get-request* do protocolo SNMPv1 necessita de várias trocas de PDUs. Por conseguinte, a operação *get-bulk-resquest* cobre uma deficiência do protocolo SNMPv1 ao permitir a recuperação eficiente de grandes blocos de informações através de uma única troca de PDU.
- No protocolo SNMPv1, caso um gerente emita um *get-request* incluindo uma lista de variáveis, onde nem todas destas variáveis estão disponíveis, a operação é abortada. No protocolo SNMPv2, esta mesma operação não é atômica e fornece resultados parciais.
- A grande modificação introduzida pelo modelo SNMPv2 está centrada, principalmente, na parte relativa à **segurança** que inclui as seguintes facilidades:

. um *mecanismo de autenticação* utilizado para assegurar que uma mensagem recebida foi realmente transmitida pelo emissor que aparece como fonte no cabeçalho da mensagem. A autenticação de mensagem habilita as partes em comunicação a verificar se alguém alterou as mensagens recebidas e se suas fontes são autênticas;

. um *mecanismo de privacidade*, baseado em criptografia, usado nas mensagens trocadas entre gerentes e entre agentes e gerentes. Privacidade é a proteção dos dados transmitidos contra escutas ou espionagens. A privacidade exige que o conteúdo de qualquer mensagem seja oculto, de modo que somente o receptor pretendido possa recuperá-lo;

. um *mecanismo de controle de acesso* que determina o tipo de acesso que uma determinada fonte pode ter sobre as informações de gerenciamento. O controle de acesso assegura que somente usuários autorizados tenham acesso a um banco de informações de gerenciamento em particular.

No modelo SNMPv1 não é possível verificar a autenticidade da fonte de uma mensagem de gerenciamento ou impedir escutas. Sem a capacidade de autenticação, o modelo SNMPv1 fica vulnerável a ataques que podem modificar ou desabilitar configurações da rede. Como resultado, muitos fabricantes de equipamentos SNMPv1 preferem não implementar o comando *set*, que permite alterar a configuração de um agente. Isto reduz a capacidade de gerenciamento dos equipamentos destes fabricantes somente à monitoração [SS94].

9.8. Gerenciamento OSI versus Gerenciamento Internet

Após termos apresentado os conceitos de Gerenciamento OSI e Gerenciamento Internet, enumeramos, a seguir, algumas diferenças entre eles:

1. Com relação à classificação de padrões:

- O Gerenciamento OSI é um padrão oficial (de jure) para gerenciamento de redes, em razão de ter sido elaborado por comissões da ISO e do ITU-T constituídas por representantes técnicos da maioria dos países com experiência em gerenciamento de redes.
- O Gerenciamento Internet é um padrão de fato (de facto) para gerenciamento de redes, em decorrência de ter sido elaborado no ambiente da rede Internet de acordo com a necessidade de gerenciá-la.

2. Com relação ao conceito de objetos gerenciados:

No Gerenciamento OSI:

- Objetos gerenciados são definidos em termos de seus atributos, operações a que podem ser submetidos, notificações que podem emitir, e suas relações com outros objetos gerenciados;
- Objetos gerenciados incluem ações que podem ser, explicitamente, invocadas como parte de acessos de gerenciamento.
- Objetos gerenciados podem gerar notificações de eventos.
- Objetos gerenciados podem incluir atributos de relacionamento com ponteiros para objetos gerenciados relacionados. Por exemplo, um objeto gerenciado (porta) pode conter um ponteiro para um objeto gerenciado (interface) onde está contido.
- Objetos gerenciados podem ser agrupados em classes de objetos.
- Objetos gerenciados podem herdar propriedades de outros objetos.
- Objetos gerenciados podem ser reutilizados.

No Gerenciamento Internet:

- Objetos gerenciados são definidos através do conceito de tipo de objeto, desvinculado do conceito de atributo.
- Objetos gerenciados não incluem ações.
- Objetos gerenciados não podem gerar notificações de eventos. Eventos estão associados a agentes.
- Não existe o conceito de classe de objeto.
- Objetos gerenciados não podem herdar propriedades de outros objetos.
- Objetos gerenciados não podem ser reutilizados.

3. Com relação à MIT:

No Gerenciamento OSI:

- Informações de gerenciamento estão armazenadas na MIT tanto em nós internos como em folhas [Yem94].
- A estrutura da MIT pode mudar dinamicamente.
- A MIT organiza objetos de acordo com o conceito de hierarquia de nomeação.

No Gerenciamento Internet:

- Informações de gerenciamento estão armazenadas na MIT somente em folhas [Sta93, Yem94].
- A estrutura da MIT é estática.
- A MIT organiza objetos de acordo com o conceito de hierarquia de registro.

4. Com relação às comunicações de gerenciamento:

No Gerenciamento OSI:

- Comunicações de gerenciamento estão embutidas no ambiente de aplicação OSI, utilizando elementos da pilha OSI, como por exemplo o ACSE e ROSE.
- Comunicações de gerenciamento utilizam, tipicamente, protocolos de transporte orientado a conexão, embora possam utilizar protocolos de transporte baseado em conexão como o TCP.
- Comunicações entre gerentes e agentes envolvem na maioria das vezes confirmação.

No Gerenciamento Internet:

- Comunicações de gerenciamento utilizam protocolos de transporte não orientados a conexão. Dentro desse contexto, o Gerenciamento OSI apresenta um nível de confiabilidade maior em relação ao Gerenciamento Internet.

5. Com relação aos protocolos de gerenciamento:

No Gerenciamento OSI:

- Existem diversos operações de gerenciamento definidas no CMIP. Algumas delas apresentam facilidades adicionais de escopo, filtro e sincronização.
- O Protocolo CMIP dispõe das funções *bulk retrieval* e *selective retrieval* que são importantes, respectivamente, para recuperar grandes blocos de informações e informações específicas respectivamente.

No Gerenciamento Internet:

- As operações de gerenciamento definidas para o protocolo SNMPv1 restringem-se às operações do tipo *get e set*, com uma sensível melhoria na definição do protocolo SNMPv2.
- Somente o protocolo de gerenciamento SNMPv2 dispõe da função *bulk retrieval* através da operação *get-bulk-request*. Nem o protocolo SNMPv1 e nem o protocolo SNMPv2 dispõem da função *selective retrieval*.

9.8.1. Considerações Finais

Apesar do Gerenciamento OSI e do Gerenciamento Internet apresentarem conceitos similares, eles apresentam características que os diferenciam.

Em termos gerais, pode-se dizer que o Gerenciamento Internet é mais simples. Conceitos no Gerenciamento OSI são complexos, de difícil implementação. Muitos destes conceitos ainda estão para ser padronizados. Alguns encontram-se no início de sua especificação, como é o caso da MIB.

Produtos que utilizam o Gerenciamento OSI são muito recentes, poucos em número e mais caros que os seus correspondentes no Gerenciamento Internet. Em troca da funcionalidade adicional fornecida, eles exigem uma maior quantidade de recursos como, por exemplo, mais memória e uma maior capacidade de processamento.

Nos últimos anos, o Gerenciamento Internet tem dominado o mercado de SGRs padronizados (de jure e de facto) devido à simplicidade de sua implementação. No entanto, a tendência é a utilização do Gerenciamento OSI porque as companhias de telecomunicações estão desenvolvendo SGRs baseados nele. Adicionalmente, existe uma grande pressão por parte do governo americano, da Comunidade Europeia e do Japão para forçar fabricantes a desenvolverem sistemas de gerenciamento de redes (SGRs) baseados no Gerenciamento OSI.

Capítulo 10: Segurança em Ambientes Distribuídos

10.1. Introdução

Um dos aspectos mais importante dos ambientes de computação distribuída é a sua ***segurança.*** Disponibilizar mecanismos de segurança em ambientes distribuídos é muito diferente de se disponibilizar estes mecanismos em ambientes centralizados. A principal diferença fica por conta da facilidade em definir-se o perímetro de proteção nos ambientes tradicionais, o que é bem mais complicado em ambientes distribuídos, devido à própria estrutura da rede que os suporta [Sch94].

Para conceituarmos segurança nos ambientes de computação, vamos primeiramente responder a algumas questões básicas [CB94]:

1) O *que queremos proteger*? A resposta para esta questão certamente depende do tipo de ambiente de computação que estivermos falando. De maneira geral desejamos proteger recursos como UCPs (Unidades Centrais de Processamento) rodando software para tratamento de arquivos especiais sobre a configuração do sistema, a partir dos quais pode-se acessar outros recursos como: arquivo com dados secretos de uma empresa ou de órgãos do governo, periféricos de armazenamento, linhas telefônicas, etc. Outro ponto que merece ser protegido é a disponibilidade de conectividade com outros sistemas. Queremos proteger todos estes recursos, e a melhor maneira de fazê-lo, é barrando os intrusos na porta da frente, evitando em primeiro lugar que eles acessem o sistema de computação.

2) *Contra quem devemos proteger o sistema*? Técnicas de espionagem como a escuta nos fios ou a análise criptográfica, o monitoramento de emissões eletromagnéticas, e outras, tornam os sistemas de computação altamente vulneráveis aos especialistas de serviços de inteligência. Contra um adolescente que tenha um modem em casa, um mecanismo de senhas sofisticado é suficiente para detê-lo. Portanto, a proteção fornecida pelos mecanismos de segurança em uma instalação deve ser proporcional às ameaças vindas de fora. Vale lembrarmos que, quanto maior a rede, maiores tornam-se estas ameaças.

3) *Que nível de segurança é necessário*? Um sistema de computação deve estar comprometido com o custo envolvido no fornecimento de segurança, tanto em termos de dólares para aquisição de roteadores e computadores para construir gateways do tipo "*portas corta fogo*" ("*firewalls*") [CB94] quanto da contratação de pessoal experiente para administrá-los. É importante que estes custos sejam bem estudados com relação ao valor da informação a ser protegida ([Sch94] [CB94]). Custos mais sutís podem surgir ligados à questão de queda da produtividade. Níveis elevadíssimos de segurança podem prejudicar a produtividade, e certamente níveis mínimos também o farão, deve-se portanto buscar chegar a um ponto de equilíbrio.

10.1.1. A Política de Segurança

Uma *política de segurança* [Bor93], consiste de uma série de decisões que irão em conjunto determinar a postura de uma organização com relação à segurança. Esta política de segurança deve determinar os limites de tolerância e os níveis de respostas às violações que possam ocorrer. Certamente, as políticas de segurança irão diferir de uma organização para outra. Por exemplo, em um departamento em uma Universidade, as necessidades de segurança são bem diferentes daquelas encontradas em grandes empresas, como por exemplo em uma administradora de cartões de crédito, a qual por sua vez difere consideravelmente de uma instituição militar. O importante a ser observado é que toda organização, independentemente de seu tipo, deve ter uma política de segurança bem definida, mesmo que seja somente para estabelcer ações a serem tomadas quando da ocorrência de eventos inaceitáveis.

Deve-se em primeiro lugar decidir aquilo que é, e o que não é permitido na instalação [Hur93c]. Este é um processo dirigido pelas necessidades estruturais de cada organização, de tal forma que é possível que podem existir diferentes políticas estabelecidas para diferentes ambientes. Por exemplo, uma empresa pode limitar o uso dos recursos em computadores de grande porte a partir dos microcomputadores rodando software para emulação de terminais; outras empresas podem restringir o tráfego de dados para fora de suas instalações, de tal forma que possam proteger-se contra a exportação de dados valiosos para empregados em viagens de negócios. Outros aspectos podem ser dirigidos por considerações tecnológicas. Por exemplo, pode-se estabelecer o uso restrito de um protocolo específico apesar de o mesmo ser extremamente útil, pois este protocolo pode eventualmente ser pouco confiável. Outras empresas podem ainda preocupar-se com o fato de seus empregados importarem software sem a permissão devida: estas empresas não querem ser acusadas de infringir os direitos autorais de terceiros.

10.1.2. Problemas Iniciais

Os especialistas em computação têm anotado casos famosos de ataques a sistemas de computação que causaram sérios problemas às organizações envolvidas. Falamos a seguir sobre alguns destes casos, onde são mostrados problemas de segurança muito interessantes.

10.1.2.1. O Ataque do Cavalo de Tróia

Um dos casos de penetração em sistemas de computação mais famosos é o ***"Ataque do Cavalo de Tróia"*** ([Tan92] [Den90] [dH93a]). Este caso ocorreu no sistema operacional Multics, no início do uso de sistemas operacionais de partilhamento de tempo. A idéia básica deste ataque era a de aproveitar-se de uma vulnerabilidade de projeto: o Multics não previa segurança para o processamento em lote, pois como ele era um sistema multi-usuário com partilhamento de tempo, privilegiava-se a segurança no processamento em-linha. Programas normais podiam ser alterados por rotinas em lote, de forma que eles passassem a fazer algum tipo de sujeira. Quando um usuário rodava a versão alterada o ataque ocorria. Estes tipos de programas eram considerados benignos, porém com instruções destrutivas escondidas em seu código, como por exemplo, a reformatação de um disco.

10.1.2.2. Vírus

Um tipo especial de ataque é conhecido como *vírus* de computador ([dH93a] [dH93b] [KW93] [Den90] [Tan92] [Bor93] [Sch94] [Kay94a]). Um vírus é um fragmento de programa que é acoplado a um programa normal com a intenção de infectar outros programas.

Um vírus típico age da seguinte maneira: o programador que cria o vírus escreve um programa útil e interessante que irá acomodar o vírus escondido em seu código. A seguir este programa, geralmente um jogo, será divulgado e fornecido de graça, por exemplo via serviços de BBSs (Bulletin Board Systems). Quando o usuário (a vítima) inicializa o programa em seu computador, este programa começa imediatamente a examinar todos os programas no disco rígido, para verificar se estes já estão infectados ou não. Quando um programa não infectado é encontrado, ele é infectado pelo vírus, que acopla seu código ao final do arquivo executável, e substitui a primeira instrução neste programa por um desvio para o código virulento. Assim, toda vez que um programa infectado é executado, ele tenta infectar outros programas.

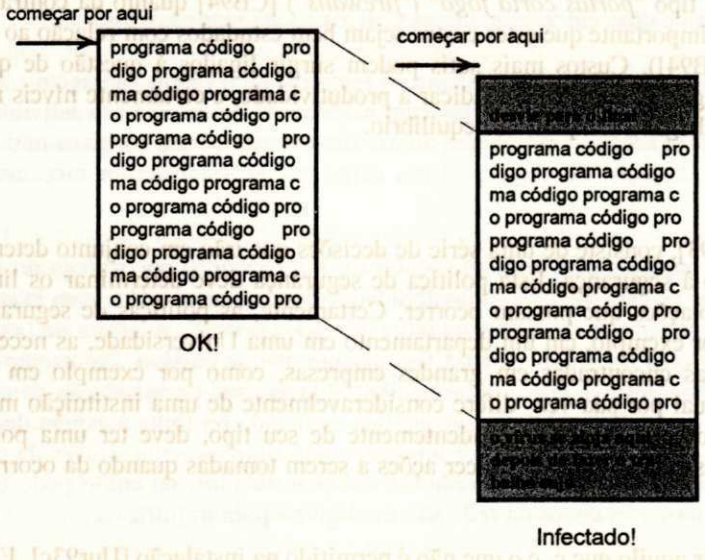


Figura 10.1 Anatomia de um vírus [Den90]

Adicionalmente os vírus podem fazer outros estragos como por exemplo apagar, modificar ou criptografar arquivos. Também é possível para um vírus infectar o setor de inicialização ("*boot*") de um disco rígido, tornando-o inadequado para inicializar o sistema [KW93]. Os vírus podem ser combatidos por programas específicos chamados de anti-vírus, que fornecem vacinas para tornar os sistemas imunes a tipos específicos de vírus.

10.1.2.3. Os Vermes na Internet

Os Vermes representam um dos maiores ataques à segurança em sistemas de computação já ocorridos. Um estudante de uma universidade americana descobriu dois *bugs* no sistema operacional Unix que permitiram que ele ganhasse acesso não autorizado a máquinas em toda a Internet. Este estudante, escreveu um programa que se auto replicava, chamado de "*verme*" ([dH93a] [Sto93a] [Tan92] [Den90]), para explorar estes bugs. Em novembro de 1988 o programa foi executado pela primeira vez, auto-replicando-se em segundos por todas as máquinas às quais ganhava acesso. Este ataque tirou do ar milhares de computadores em universidades, grandes empresas, e laboratórios do governo em todo o mundo, antes que fosse descoberto e removido. Os vermes são também conhecidos por vírus ***cruise***, assim batizados em homenagem aos mísseis do mesmo nome, usados na guerra do Iraque contra Israel em 1991. Um vírus é diferente de um verme por se aninhar em outro programa, enquanto o verme é ele mesmo um programa completo. Tanto vírus quanto vermes tentam se espalhar, podendo causar sérios danos.

10.2. Conceitos Básicos

As definições a seguir são básicas para a compreensão dos aspectos de segurança em ambientes de computação. É importante que se tenha uma visão clara das mesmas para se possa entender os problemas específicos relacionados com as questões da segurança em ambientes distribuídos.

10.2.1. Vulnerabilidades

Uma *vulnerabilidade* ([Bor93] [Sch94]) em termos de segurança é um ponto fraco de projeto de sistema que pode ser explorado com más intenções. Por exemplo, a liberação de linhas de comunicações através de modens para a conexão com um determinado sistema torna-o vulnerável a acessos por pessoas não autorizadas.

10.2.2. Ameaças e Ataques

Uma *ameaça* ([Bor93] [Sch94]) a um sistema consiste de uma pessoa ou de uma organização que podem querer atentar contra a segurança deste sistema.

Um *ataque* ([Bor93] [Sch94] [Tan92]) é o que acontece quando uma ameaça tenta levar vantagem sobre uma vulnerabilidade. Os ataques podem ser de duas categorias: ataques ativos ou ataques passivos.

Nos *ataques ativos* [Sch94], as ameaças alteraram o sistema na tentativa de levarem vantagem sobre suas vulnerabilidades. Por exemplo, uma pessoa que tenta se conectar com a senha de outra ou tenta adivinhar uma senha pelo método da tentativa e erro, esta praticando um ataque ativo. São ataques contra os quais não se pode prevenir, pode-se somente detectá-los. A detecção de ataques ativos exige que se estabeleçam mecanismos de contra-ataque.

Nos *ataques passivos* [Sch94] as ameaças simplesmente observam a informação que trafega no sistema. Não são introduzidas informações para que se explorem as vulnerabilidades deste sistema . Um bom exemplo de ataque passivo é o monitoramento do tráfego em um dos cabos na rede, que pode permitir a leitura de uma senha de acesso. Um intruso passivo pode operar com meios ativos para facilitar um ataque passivo. Por exemplo, pode instalar ativamente uma escuta em um cabo. Observe que a natureza deste ataque, após a instalação da escuta, é fundamentalmente passiva. O intruso passivo tradicional é aquele que fica escutando na linha à espera de senhas. Os ataques passivos podem ser prevenidos. A prevenção pode ser implementada simplesmente através da proteção física de todos os locais onde os componentes e os cabos da rede estiverem instalados, ou então pode envolver esquemas de criptografia sofisticados para tornar os dados na rede não-inteligíveis para os intrusos.

10.2.2.1. Tipos de Ataques mais Comuns

A seguir apresentamos uma lista dos ataques mais comuns [Tan92] aos sistemas operacionais em computadores hospedeiros, e que são quase sempre bem sucedidos:

- Tentar executar chamadas ilegais ao sistema, ou tentar executar chamadas legais ao sistema com parâmetros ilegais, ou ainda, tentar executar chamadas legais ao sistema com parâmetros também legais, porém inadequados. Muitos sistemas confundem-se com estas chamadas, e podem deixar claras para os intrusos as suas vulnerabilidades.

- Iniciar o processo de login e digitar alguma tecla de escape (DEL, ESC, BREAK, etc) durante a seqüência de conexão. Em muitos sistemas a validação da senha será abortada, e o login poderá ser bem sucedido. Neste caso, o intruso poderá ter acesso a um contexto operacional privilegiado.
- Tentar modificar estruturas de dados de controle do sistema operacional que são mantidas na área de espaço de usuários. Em muitos sistemas operacionais, principalmente naqueles que implementam o conceito de micro-núcleos, para abrir um arquivo o programa de usuário precisa criar uma estrutura de dados que contenha o nome do arquivo juntamente com outros parâmetros, e em seguida envia esta estrutura para o sistema operacional. À medida que o arquivo vai sendo lido e gravado, o próprio sistema operacional se encarrega de atualizar a estrutura de controle. Ao modificar campos nesta estrutura o intruso pode eventualmente quebrar a segurança do sistema.
- Enganar os usuários com um programa parecido com a tela de login. Os usuários irão se conectar com suas identificações e senhas, que serão guardadas para uso indevido.
- Procurar nos manuais instruções do tipo "Não faça KKK". Tentar quantas variações forem possiveis para KKK.
- Requisitar páginas de memória, buffers para arquivos em discos ou fitas, e ler as informações alí contidas para uma área de trabalho de usuário. Muitos sistemas não apagam as informações contidas nestas áreas depois de as alocarem para um novo usuário. Elas podem estar cheias de informações interessantes que tenham sido gravadas pelo usuário anterior.

10.2.3. Mecanismos de Contra-ataque

Um *mecanismo de contra-ataque* [Sch94] é uma ação tomada com o intuito de fornecer proteção contra um ataque específico (ou contra uma classe de ataques) a uma vulnerabilidade em particular. Por exemplo, em alguns sistemas de linhas discadas, se um usuário tenta se conectar diversas vezes com uma senha incorreta, o sistema irá recusar novas tentativas de conexão após um número intolerável de tentativas. Este comportamento é um contra-ataque projetado para tornar o processo de adivinhação da senha mais difícil.

A *reutilização de objetos* [Bor93] é um mecanismo de contra-ataque bem conhecido pelos projetistas de sistemas operacionais. É uma técnica usada para garantir que áreas de memória principal ou secundária sejam reformatadas após o seu uso por um determinado usuário ou processo de aplicação. Assim, novos usuários ou processos de aplicação não poderão aproveitar-se de informações que tenham sido inadvertidamente deixadas nestas áreas pelos usuários anteriores. Para que esta técnica funcione, o sistema operacional precisa ser informado sobre o término de um serviço de usuário, de tal forma que ele possa limpar estas áreas antes de as mesmas serem realocadas.

Um conceito relacionado com a reutilização de objetos é o protocolo de *auto-bloqueio* [Mae87], uma facilidade disponível há muito tempo no mundo dos mainframes, que no entanto é raramente encontrada nos ambientes cliente/servidor da computação distribuída. Neste tipo de contra-ataque, um programa de auto-bloqueio desconecta automaticamente usuários que tenham esquecido suas estações de trabalho conectadas. O programa roda na retaguarda, e procura por estações de trabalho inativas. O administrador da rede pode estabelecer um tempo limite de tolerância para uma conexão inativa. Se este tempo estabelecido for ultrapassado, a conexão é terminada imediatamente.

O mais comum tipo de ataque a computadores ocorre com as quebras de *senhas protegidas* ([Sch94] [Tan92] Lam91]). Uma grande parte das violações em sistemas de computadores deve-se a falhas no sistema de senhas. Estas falhas podem ser várias, e a mais comum é a *adivinhação* de senhas.

Senhas são extremamente vulneráveis [Mal92b]. Se olharmos em baixo de teclados em muitos terminais, ou tentarmos nomes de amigos ou parentes de um usuário (ou cães, ou marcas de carros), ou se simplesmente perguntarmos a este usuário qual a sua senha, temos uma boa chance de descobri-la.

Os ataques por adivinhação de senhas podem ser de duas formas: a primeira envolve atentados utilizando-se nomes de usuários conhecidos e tentando-se adivinhar suas senhas. Geralmente esta estratégia funciona muito bem. Para se proteger destes tipos de atentados, pode-se tomar algumas precauções, como por exemplo, evitar que um número infinito de tentativas de login com senhas inválidas seja permitido. Nestes caso, o mais interessante é que se estabeleçam arquivos de log onde sejam notificados estes atentados.

A segunda forma de atentado a senhas é pelo roubo de arquivos de senhas (por exemplo, o arquivo /etc/passwd no sistema operacional Unix). Estes roubos podem ocorrer em sistemas que tenham sido invadidos, de tal forma que os intrusos tentarão usar as senhas roubadas em outras máquinas (os usuários tendem a reutilizar a mesma senha em vários sítios). Estes ataques são chamados de *ataques de dicionário* [Ste91], e geralmente obtêm alto grau de sucesso.

Os contra-ataques que podem ser implementados para que sejam evitados estes tipos de problemas, requer que os usuários sejam educados para escolherem senhas adequadas, e que os arquivos de senhas sejam muito bem protegidos.

No entanto, o contra-ataque mais radical implementado para proteger uma senha é implementado por meio de um enfoque conhecido como *senha descartável* (*"one time password"*) ([Tan92] [Bor93] [Dew93]). Quando este método é empregado, os usuários recebem um livreto contendo uma lista de senhas válidas. A cada login o usuário utiliza uma nova senha da lista. Mesmo que um intruso descubra uma das senhas, ela não lhe será útil, pois na próxima vez uma senha diferente será requisitada. Certamente, o usuário não poderá jamais perder o livreto de senhas.

10.3. O Problema da Segurança em Redes de Computadores

A maioria dos problemas de segurança começam a surgir quando um ambiente tradicionalmente centralizado começa a tornar-se aberto pelo estabelecimento de conexões com redes no mundo exterior [CB94].

Por que uma empresa estaria interessada em colocar os seus segredos em risco em troca dos benefícios conseguidos com estas conexões? Esta questão pode ser apreciada de dua maneiras. Em primeiro lugar, é imprescindível que grandes empresas se interliguem em todo o mundo, de tal forma que sejam atingidos seus objetivos de negócios agora definidos em níveis globais. Em segundo lugar existe o outro extremo da questão: por exemplo, as instituições militares americanas são bastante desconfiadas com relação às técnicas de segurança empregadas em sistemas de computação - em muitos órgãos, proibe-se terminantemente que seus computadores com informações confidenciais se conectem a redes não seguras.

As redes e inter-redes fornecem muitas vantagens; desconectar um computador da rede pode inibir estas vantagens. A desconexão pode até ser uma boa solução, que deve ser avaliada de forma critériosa, julgando-se os prós e os contras da questão cuidadosamente.

Os ambientes da computação distribuída operam com infra-estrutura básica fornecida por redes de computadores. A seguir discutimos os problemas de segurança que podem impactar estes ambientes, levando em consideração as vulnerabilidades que podem surgir nas redes que os suportam.

10.3.1. As Redes Locais Isoladas

As redes locais isoladas são caracterizadas por estarem fisicamente disponíveis para um conjunto limitado de pessoas, como por exemplo, para os empregados de uma empresa. As ameaças potenciais em redes locais isoladas são pequenas se comparadas àquelas em redes interligadas (inter-redes) [Dew93].

A principal ameaça nestes segmentos isolados de redes são os próprios empregados da empresa. Por estranho que pareça, o pessoal interno é quem mais preocupa. Normalmente estas pessoas estão familiarizadas com o sistema e sabem onde estão suas vulnerabilidades.

10.3.2. As Redes Locais Interligadas (Inter-redes)

No caso das inter-redes, existem conexões com outras redes que estão fora do escopo de controle do domínio local, o que aumenta consideravelmente a lista de potenciais ameaças. [CB94]. Nestes casos, além do próprio pessoal interno deve-se preocupar também com o pessoal externo que pode querer acessar a rede local através de suas conexões como o mundo exterior.

As redes interligadas podem ser categorizadas como [Kay94a]: redes locais totalmente conectadas e redes locais parcialmente conectadas.

As redes locais *totalmente conectadas* encaixam-se sem remendos nas redes maiores com as quais elas se conectam. Um bom exemplo são segmentos de redes locais ligados à Internet.

As redes locais *parcialmente conectadas* são aquelas disponíveis fora dos limites da organização, e que utilizam tecnologias de comunicação diferentes daquelas usadas nos segmentos da rede local corporativa. Por exemplo, ao invés de Ethernet ou TokenRing, pode-se utilizar um protocolo de comunicação que forneça uma conexão em linha discada através de um modem. Uma conexão como esta é parcial porque a rede telefônica só fornece um acesso limitado à rede.

Praticamente qualquer usuário no mundo pode ameaçar uma rede totalmente conectada caso esta não estabeleça políticas adequadas com relação ao seu acesso. É comum ouvirmos falar de ataques que ocorreram em países onde nem ao menos se considera estes tipos de ataques como um caso fora da lei. Portanto, um importante conselho para a segurança nos sistemas distribuídos é: "proteja-se por sua conta!". Não espere que o mundo exterior tenha regras bem definidas para facilitar sua vida.

10.3.3. Redes Seguras

Na maior parte dos casos de ataques, a rede não é propriamente o recurso que está correndo perigo; o objetivo dos intrusos é chegar a um computador hospedeiro (ou a um servidor principal) da rede. Portanto, estes computadores hospedeiros devem ser configurados e armados para resistir aos ataques [CB94]. O mais difícil, no entanto, é conseguir montar um esquema de segurança completo, pois com toda certeza ainda existirão bugs nos programas de segurança da rede, ou existirão falhas humanas na administração do sistema. Se um intruso encontrar um ponto fraco sequer, o sistema será penetrado.

Nossa visão é pessimista, mas não sem motivos. Mesmo os sistemas mais seguros, garantidos por critérios **A1** do *livro laranja* do Departamento de Defesa (**DoD**) americano [Bla91], não são totalmente garantidos, pois existem casos detectados de penetração em uma de suas poucas vulnerabilidades. Na Internet a situação é muito pior, pois poucos sistemas chegam a pelo menos o nível **C2** do livro laranja. Portanto, a segurança na Internet é pouco adequada. Adiante, na seção 10.9, abordamos as questões de **padronização** para segurança.

Com relação à dificuldade de se administrar sistemas seguros, não importa com quanta certeza o código do software de segurança tenha sido escrito, ou o quanto o projeto tenha sido bem elaborado, o fato é que se houverem erros humanos, todas as proteções podem ficar comprometidas. Vejamos o seguinte exemplo:

- Uma estação com a função de gateway em uma rede apresentou uma falha em um feriado de final de semana, quando nenhum dos administradores de sistemas da instalação estavam disponíveis. Um operador mais experiente tentou resolver o problema a partir de seu microcomputador em casa, através de uma conexão de linha discada, e para tanto precisou criar uma conta de nome "*convidado*". Esta conta foi criada sem senha de acesso. Terminado o serviço o operador local esqueceu de apagar a conta. Em um belo dia, estudantes universitários descobriram a tal conta e contaram a seus colegas.

Este é um caso verídico acontecido em um dos laboratórios da AT&T nos Estados Unidos. Este fato só foi descoberto meses depois, quando tentaram burlar um recurso protegido por alarme, o qual advertiu o administrador de sistemas da instalação. Portanto, a segurança das redes é imprescindível para a confiabilidade e privacidade das informações armazenadas nos computadores hospedeiros.

10.4. Serviços Básicos

Os sistemas de segurança fornecem cinco serviços básicos: identificação, autenticação, autorização, criptografia e auditoria. Falamos a seguir sobre cada um destes serviços.

10.4.1. Identificação

Nos ambientes da computação distribuída, o conceito de "usuário" estende-se do domínio de uma única máquina para todos os domínios envolvidos em uma rede dispersa e heterogênea. Um usuário na rede é uma entidade que existe em uma única ou em várias localizações, e que possui uma ou várias identidades associadas, cada uma com suas próprias características. Estas identidades e suas características são usadas para determinar que recursos na rede este usuário tem acesso liberado.

Em sistemas centralizados, o controle de acesso a recursos por usuários é uma tarefa trivial de se gerenciar. Nos sistemas distribuídos, esta tarefa torna-se dificílima: o usuário pode estar em qualquer parte. Como estes usuários são meras entidades na rede, eles podem estar logados em diferentes plataformas, como em Macintoshes, PCs, estações Unix, terminais de mainframes, terminais X, e outros. É comum que muitos destes ambientes sejam usados em conjunto para que se consiga executar determinadas tarefas.

Os geradores de senhas aleatórias são chamados de *passes* (*"tokens"*) e fornecem uma palavra alfanumérica ou um número pseudo-randômico que muda a cada minuto e é sincronizado com uma base de dados armazenada em um servidor na rede. Esta estratégia resulta em uma senha descartável, que é útil somente em um momento particular e para somente um único login.

Os dispositivos desafio-resposta são um outro tipo de passe que consistem de uma pequena calculadora que permite que o usuário digite um valor fornecido pelo servidor quando o usuário inicia o processo de login. Este valor gerado pelo servidor é chamado de "desafio". A calculadora (o passe) depois que recebe o desafio gera uma "resposta" que o usuário pode usar para digitar na sua estação, de tal forma que o processo de login se completa. A não ser que o usuário esteja de posse de seu passe (a calculadora), nenhum login será possível. Geralmente a própria calculadora possui uma senha para ser usada.

Estes são exemplos de produtos bastante interessantes, porém ainda um pouco distante de nossa realidade cotidiana devido ao seu preço elevado.

Prova por Conhecimento (O que você sabe?)

A terceira maneira de se identificar um indivíduo é por meio de algo que este indivíduo (e somente ele) saiba. As tradicionais senhas de login são um bom exemplo deste tipo de tecnologia de autenticação. Este é o tipo mais freqüente de autenticação em sistemas de computação. É também um dos mais problemáticos, pois as pessoas tendem a escolher senhas extremamente simples e fáceis de serem descobertas.

A combinação das técnicas

Uma maneira de se reforçar as técnicas de autenticação é por meio da combinação das técnicas apresentadas acima. A combinação mais comum encontrada mistura *o que você tem* com *o que você sabe*. Um cartão de saque do Banco 24 Horas é um bom exemplo. Ele só é útil se uma senha de identificação for fornecida depois que o cartão é lido pela leitora do caixa eletrônico.

10.4.3. Autorização

Depois de feita a identificação de um indivíduo (ou de um processo), e um pedido for dirigido para um servidor, o próximo passo com relação à segurança diz respeito à autorização deste pedido. Observe que autorização não é a mesma coisa que autenticação, porém, a maioria dos sistemas que executam algum tipo de autenticação também possuem um componente de autorização implícito.

A forma mais comum de autorização em sistemas de computação é fornecida pelas ***listas de controle de acesso*** (**ACLs** - *Access Control Lists*) ([Tan92] [Mae87] [Mil87] [OSF90c] [OSF92a]). As ACLs residem geralmente em arquivos em disco e contêm os nomes das entidades autenticadas juntamente com uma lista de operações que são permitidas sobre os objetos disponíveis em uma determinada instalação.

Como um exemplo de uso das ACLs, vamos imaginar que um determinado sistema operacional as implemente, de tal maneira que tenhamos grupos de usuários (GUs) e usuários (USUs) que irão compor a tupla (USU, GU) que chamaremos de *domínio*. Os domínios contêm as entidades autenticadas. Assim as ACLs irão associar a cada objeto (arquivos, diretórios, etc) uma lista de domínios que podem acessar aquele objeto, e como podem ser feitos estes acesso. As ACLs são o mecanismo mais comumente empregado pelos sistemas de catálogos em computadores tipo mainframes.

Suponhamos que existam quatro usuários em nosso sistema (isto é, USUs): Suzana, Pascal, Robério e Jácques, que pertencem aos grupos (GUs): *sistema, pessoal, estudante* e *estudante*, respectivamente. Suponhamos também que alguns objetos possuam as seguintes ACLs:

Objeto0: (Suzana, *, RWX)
Objeto1: (Suzana, sistema, RWX)
Objeto2: (Suzana, *, RW-), (Pascal, pessoal, R--), (Jácques, *, RW-)
Objeto3: (*, estudante, R--)
Objeto4: (Robério, *, ---), (*, estudante, R--)

Cada entrada na ACL (em parêntesis) especifica um USU, um GU e os tipos de acessos permitidos (Read, Write e eXecute). Um asterisco significa todos USUs ou GUs. O objeto **Objeto0** pode ser lido, gravado ou executado por qualquer processo que tenha USU=Suzana, em qualquer GU. **Objeto1** pode ser acessado somente pelos processos com USU=Suzana, e GU= sistema. Um processo que tenha USU=Suzana, e GU=pessoal, poderá acessar o objeto **Objeto0**, mas não Objeto1. **Objeto2** pode ser lido ou gravado por processos com USU=Suzana em qualquer GU, poderá também ser lido por processos com USU=Pascal e GU=pessoal, ou ainda por processos com USU=Jácques em qualquer GU. **Objeto3** pode ser lido por qualquer estudante. **Objeto4** é especialmente interessante: diz que qualquer USU=Robério, em qualquer grupo, não possui nenhum tipo de acesso permitido, porém todos os outros estudantes podem ler este objeto. Assim, podemos ver que as ACLs representam um poderoso mecanismo de proteção para os sistemas de autorização a serem implementados nos ambientes da computação distribuída.

10.4.4. Criptografia

Os ***sistemas de criptografia*** ([ACM92a] [Sch94] [Kay94a] [Way93a] [Way93b]) permitem que informações sejam enviadas através de um hardware não confiável, e sejam recebidas de tal forma que se possa detectar, dependendo do tipo de serviço de criptografia usado, se a informação foi ou não destorcida ou espionada em sua jornada. É praticamente impossível implementar-se a segurança em ambientes da computação distribuída sem os sistemas de criptografia.

Os serviços de criptografia podem ser divididos em três tipos básicos [Sch94]: (1) *confidencialidade*: é o serviço que assegura que as informações não estarão disponíveis para pessoas (ou processos) não autorizados enquanto elas atravessam o sistema de informações; (2) *autenticação*: é o serviço que permite que se saiba se uma parte específica de informação foi gerada de uma entidade em particular; (3) *integridade*: é o serviço que permite que se determine se uma parte da informação não foi alterada enquanto atravessava o sistema de informação. Na seção 10.5 apresentamos os conceitos básicos e alguns detalhes a respeito de sistemas de criptografia.

10.4.5. Auditoria

Os objetivos dos sistemas de autenticação e autorização são de assegurar que somente ações autorizadas sejam executadas. A ***auditoria*** [Bor93] é o processo que cuida de manter registros detalhados sobre quem fez o que com qual objeto, para que se possa determinar como um ataque foi efetuado e que tipo de informação ficou comprometida. Exemplos de eventos que são registrados podem ser: que programas foram usados, que arquivos foram abertos, quantas leituras/gravações foram executadas, quantas vezes um usuário tentou se conectar, etc. Certamente, não é suficiente que sejam guardados os registros de auditoria. É necessário que estes registros possam ser analisados para que se assegure que nenhum tipo de atividade ilícita tenha ocorrido. Portanto, um bom sistema de auditoria deve não somente guardar registros os mais detalhados possiveis, mas deve também fornecer facilidades que ajudem o usuário a rever estes registros para que se detectem possiveis atividades ilícitas, permitindo inclusive que sejam filtradas somente as informações de interesse para o administrador de sistemas.

10.5. Introdução à Criptografia

Uma parte importante dos *sistemas de criptografia* são os algoritmos e métodos usados pelos mesmos. Desenvolver algoritmos de criptografia é uma tarefa complicada, onde não existem regras de projeto claras para facilitar a vida dos projetistas. Os algoritmos de criptografia (codificação) podem ser classificados em dois tipos [Sch94]: ***codificadores simétricos*** e ***codificadores assimétricos.***

10.5.1. Codificadores Simétricos

Nos codificadores simétricos, também chamados de codificadores convencionais, as informações são transformadas algoritmicamente a partir de uma forma legível, chamada de *texto convencional*, para uma forma não legível, chamada de *texto codificado*. Esta transformação é função de uma ***chave***, que é geralmente mantida em segredo. Na criptografia simétrica a mesma chave que é usada para codificar informações, é também usada para decodificar texto codificado para sua forma original de texto convencional.

O mais largamente usado sistema de criptografia é chamado de *Data Encryption Standard* (**DES**), um padrão do governo americano. O DES foi desenvolvido pela IBM em 1975, e sofreu algumas modificações antes de ser publicado como padrão *de jure* pelo **NIST** (*National Institute of Standards Technology*), e suportado em hardware e software por uma grande variedade de fornecedores. É um sistema robusto que não demonstrou nenhum tipo de vulnerabilidade desde que foi lançado disponibilizado para uso geral [Way93b]. Uma das críticas ao DES é com relação à sua natureza consumidora de tempo de CPU quando implementado em software para ambientes distribuídos, pois o mesmo foi projetado para ser implementado originalmente em hardware.

Alguns fornecedores de software implementam criptografia em seus produtos utilizando algoritmos proprietários, que são significativamente mais rápidos que o DES. Por exemplo, a empresa americana RSA Data Security, fornece um clone proprietário similar ao DES, chamado de RC-4, e que implementa um algoritmo de nome *Message Digest 4* (**MD4**) bastante eficiente. O problema que surge com alternativas deste tipo é que o uso de algoritmos proprietários torna seus usuários dependentes da competência (ou incompetência) destes fornecedores. A recomendação é que se use o DES sempre que possível. O Kerberos, do qual falaremos adiante, também implementa um algoritmo de codificação simétrico [Mal92b].

A Administração de Chaves de Codificação

Um dos problemas mais sérios em sistemas de segurança que implementam criptografia é o controle da distribuição de chaves secretas de codificação. Os sistemas baseados em criptografia dependem de um procedimento confiável e incorruptível para a distribuição destas chaves. Em muitos sistemas que usam codificação simétrica, as chaves precisam ser distribuídas manualmente, de tal forma que os usuários são obrigados muitas vezes a transportarem estas chaves consigo para que as mesmas possam ser instaladas. Nestas situações, se um destes usuários não for confiável todo o sistema pode ficar comprometido.

10.5.2. Codificadores Assimétricos

Os *codificadores assimétricos*, ou *codificadores de chaves públicas* diferem dos codificadores simétricos porque ao invés de terem uma única chave para controlar tanto a operação de codificação quanto a operação de descodificação, são empregadas duas chaves. Estas chaves são conhecidas como *chave pública* e *chave privada* ([Mal92b] [Den90]). Uma destas chaves, a chave pública, pode ser divulgada abertamente na rede. A grande vantagem desta estratégia é que o problema de distribuição das chaves entre clientes e servidores fica mais fácil de ser solucionado.

Vamos considerar uma situação simples onde duas pessoas desejam comunicar-se de maneira segura. Com um algoritmo de codificação simétrica, estas pessoas terão que trocar uma chave secreta a ser partilhada, de tal maneira que ninguém mais tenha conhecimento da mesma. Provavelmente, a única maneira de se efetuar esta troca será pessoalmente (ou através de um portador de extrema confiança).

Consideremos agora a mesma situação com a codificação assimétrica. Cada pessoa informa à outra a sua chave pública, abertamente, sem se preocupar que outras pessoas a conheçam. Depois de trocadas as chaves públicas, estas pessoas poderão comunicar-se secretamente por meio de mensagens codificadas com a chave pública um do outro.

A chave privada é secreta e conhecida somente pelo seu dono. Já a chave pública é conhecida por todos. Se a chave pública for usada para codificar os dados, então somente a chave privada poderá decodificá-los. Da mesma forma, se a chave privada for usada para codificar os dados, então a chave pública será usada para decodificá-los.

Que sentido existe por detrás de se codificar dados com uma chave privada se todos podem decodificá-los com uma chave pública? A codificação com chave privada, e posterior decodificação com chave pública serve para se verificar a identidade do remetente. Dados codificados com uma chave pública não estão seguros contra decodificação, porém o destinatário assegura-se da identidade do remetente. Esta funcionalidade recebe o nome de "assinatura digital" [Den90].

Resumindo, cada usuário possui duas chaves, uma pública e uma privada. Um usuário remetente utiliza a chave pública do destinatário para enviar uma mensagem, e o usuário destinatário usa uma chave privada para decodificar a mensagem. Como somente o destinatário possui a chave secreta a mensagem pode ser enviada confidencialmente. Se a mensagem for interceptada, ou roteada incorretamente, ela não poderá ser decodificada e lida. Adicionalmente, a mensagem pode conduzir informações extras para assegurar ao destinatário a identidade do remetente.

O mais conhecido dos codificadores assimétricos é o RSA [Way93b] desenvolvido no MIT (Massachuessetts Institute of Technology) em 1978, e patenteado nos Estados Unidos pela empresa RSA Data Security, a qual cobra taxas de licenciamento para o seu uso. A Novell implementa o RSA no seu produto NDS (Netware Directory Service) [Ude93c] que implementa serviços de nomes e segurança em suas redes locais NetWare. O RSA é baseado em uma dificuldade matemática de derivação de números primos [Mal92b].

Uma das principais desvantagens dos codificadores assimétricos, incluindo o RSA, é que eles são ordens de magnitude mais lentos do que os codificadores simétricos. Estes problema é contornado por uma estratégia simples: os codificadores assimétricos são usados somente para permitir a troca de chaves em codificadores simétricos. Recentemente, a Internet Activities Board propôs um padrão baseado na combinação dos algoritmos RSA e DES. Esta proposta prevê um sistema criptográfico híbrido que pretende melhorar a privacidade de mensagens eletrônicas trocadas na Internet e que também permita que se autentique a identidade do remetente [Mal92b].

Por exemplo, deseja-se enviar uma mensagem secreta para um usuário, de tal forma que se conheça sua chave RSA pública (para a codificação assimétrica). Pode-se usar esta chave RSA pública para se codificar uma outra chave, escolhida aleatoriamente. Esta outra chave pode ser uma chave randômica DES (para a codificação simétrica), que será usada para se codificar a própria mensagem a ser transmitida. Agora codifica-se esta chave DES com a chave RSA pública do destinatário. A seguir envia-se a chave DES codificada (pelo RSA) juntamente com a mensagem codificada (pela chave DES) para o destinatário.

O destinatário decodifica a chave DES (por meio de sua chave RSA privada) e a seguir decodifica a mensagem propriamente dita (com a chave DES decodificada). Um intruso não pode interceptar e decodificar a mensagem porque ele não possui a chave RSA privada do destinatário, e não pode desta forma recuperar a chave DES que foi usada originalmente para codificar a mensagem. Assim, o RSA foi usado como um portador de extrema confiança para entregar a chave do codificador simétrico.

O Chip de Codificação Clipper

Uma nova era nas comunicações criptografadas está por surgir com a disponibilização para uso comercial do chip de criptografia Clipper, desenvolvido com o apoio da pela Agência Nacional de Segurança americana (NSA- National Security Agency). O chip será usado para codificar tanto dados quanto voz. O chip Clipper [Way93a] consiste de um coprocessador de 12 Mbps projetado pela empresa Mykotronk e fabricado pela empresa VLSI, ambas na California. É um chip protegido contra esforços da engenharia reversa, o que garante a segurança do algoritmo que ele implementa. Seu custo ainda é elevado, porém espera-se que com a produção em massa ele já possa ser usado na *"Super Highway"* de informações americana [BG93].

O chip implementa uma "trapdoor" que permitirá que órgão da justiça dos Estados Unidos possam escutar legalmente o tráfego em uma conexão. O algoritmo implementado foi desenvolvido por um professor do MIT, e chama-se de "fair crypto system". Neste algoritmo a chave de decodificação pode ser dividida em várias partes e atribuída a diferentes donos. A chave só pode ser reconstruída se todos os portadores dos pedaços concordarem com a sua montagem.

Por exemplo, o governo pode fornecer à polícia local autorização para que sejam fornecidas as partes da chave para um determinado chip, de tal forma que a mensagem possa ser escutada via a "trapdoor". Este tipo de permissão tem gerado muitas controvérsias a respeito da privacidade dos cidadãos, porém a argumentação é que ela só será usada para rastrear o uso criminoso das redes de computadores.

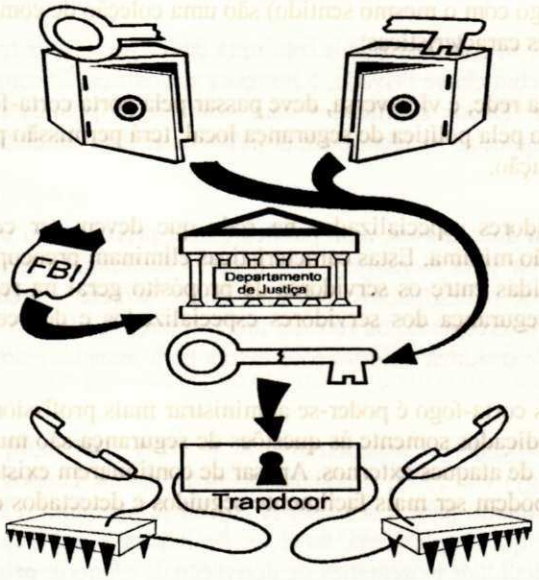


figura 10.2. A solução oferecida pela pastilha ("*chip*") ***Clipper*** [Way93a]

10.5.3. Tipos de Ataques em Sistemas de Codificação

Os ataques a sistemas de criptografia [Sch94] são paralelos aos ataques convencionais vistos até agora. Um sítio que enfrente ameaça de um intruso altamente competente deve estabelecer defesas tanto contra ataques à criptografia quanto contra os ataques mais convencionais. São vários os ataques que os sistemas de criptografia podem sofrer. É impossível fornecer uma lista completa, porém discutimos alguns dos ataques mais importantes:

- *Análise criptográfica*: consiste da leitura de tráfego criptografado sem o conhecimento da chave de codificação usada.
- *Análise criptográfica "prática"*: é o contrário da análise criptográfica pura. Força o roubo de uma chave por quaisquer que sejam os meios.
- *Ataque a texto convencional conhecido*: geralmente um intruso irá possuir um ou mais pares de textos codificados com a mesma chave. Estes pares, conhecidos como *berços ("cribs")*, podem ser usados para ajudar na análise criptográfica.
- *Busca exaustiva*: tenta-se todas as chaves possiveis. Também conhecido como ataque de força bruta.
- *Escuta passiva*: Um intruso passivo simplesmente escuta o tráfego na linha.
- *Ataque ativo*: o intruso pode inserir mensagens ou deletar ou modificar mensagens originais.
- *Homem-no-meio*: o inimigo fica entre a vítima e o destinatário da mensagem, e escuta, possívelmente alterando, as mensagens de um para o outro.
- *Corta-e-cola*: dadas duas mensagens codificadas com a mesma chave, é às vezes possível combinar porções de duas ou mais mensagens para se produzir uma nova mensagem. Pode-se não entender o que ela significa, porém pode-se utilizá-la para que a vítima faça algo que ajude na análise criptográfica.
- *Restabelecimento-da-hora*: em protocolos que dependem da hora corrente, pode-se tentar confundir a vítima com relação à hora correta.
- *Ataque de aniversário*: um ataque nas funções de hash, onde o objetivo é encontrar duas mensagens que levem a um mesmo valor.

10.6. Mecanismos de Proteção para os Ambientes Distribuídos

Existem muitos enfoques para o fornecimento de mecanismos de segurança. Abaixo apresentamos alguns dos mecanismo de proteção mais interessantes encontrados em ambientes da computação distribuída.

10.6.1. Portas Corta-Fogo ("Firewalls")

As ***portas corta-fogo*** (*"firewalls"*) [CB94] (literalmente, a tradução seria "parede corta-fogo", porém aqui no Brasil nos referimos a portas corta-fogo com o mesmo sentido) são uma coleção de componentes colocados entre duas redes, e que apresentam as seguintes características:

- Todo o tráfego de dentro para fora da rede, e vice-versa, deve passar pela porta corta-fogo;
- Somente tráfego autorizado, definido pela política de segurança local, terá permissão para passar adiante;
- A porta corta-fogo é imune a penetração.

As portas corta-fogo são servidores especializados na rede que devem ter como características uma administração simplificada e configuração mínima. Estas características eliminam preocupações adicionais, pois as portas corta-fogo não podem ser escolhidas entre os servidores de propósito geral na rede. As portas corta-fogo representam um ponto importante na segurança dos servidores especializados e dos computadores hospedeiros ("hosts") em uma rede.

Uma vantagem no uso de portas corta-fogo é poder-se administrar mais profissionalmente as estações que as implementam. Os administradores dedicados somente às questões de segurança são muito mais conscientes dos problemas que podem surgir decorrentes de ataques externos. Apesar de continuarem existindo os riscos de ataques do próprio pessoal interno, estes riscos podem ser mais facilmente seguidos e detectados do que aqueles vindos de fora.

Portas Corta-fogo na Internet

A principal vantagem no uso de portas corta-fogo é que com elas, o administrador da rede pode aumentar a segurança de toda uma organização, evitando que direitos de acesso a servidores e estações de trabalho na rede sejam burlados [CB94]. Podemos considerar, de maneira radical, que as melhores portas corta-fogo são aquelas que simplesmente tornam as redes internas totalmente isoladas. Porém, a conexão com outras redes é primordial para as organizações, de tal forma que esta solução é muito fora da realidade.

Uma das principais razões para se ter uma conexão de rede com o mundo exterior é facilitar a troca de informações. Assim, as portas corta-fogo são usadas para barrar estas trocas de informações, porém, o nível de proteção que será imposto pelo administrador da rede terá que ser muito bem calculado.

Construir uma porta corta-fogo na Internet consiste da instalação de software de filtragem nos roteadores que conectam uma organização ao mundo exterior, de modo que somente determinados tipos de pacotes tenham permissão para trafegar de fora para dentro de um segmento de rede específico.

Sabemos que na Internet, os protocolos TCP e UDP cuidam dos serviços de transporte na rede. Ambos protocolos fazem uso de números de *portas* [Com91a]. Uma porta é um número único que é assinalado para um sistema final na rede. Uma conexão (TCP ou UDP) é completamente especificada pelos endereços IP dos dois sistemas finais e pelas portas nestes endereços.

Os serviços são requisitados a partir de *portas bem conhecidas* [Com91a]. Por exemplo, Telnet sempre *escuta* na porta 23 enquanto que o FTP escuta na porta 21. A chave para a construção das portas corta-fogo na Internet é controlar o acesso entre estes pares de portas. Por exemplo, um roteador pode ser instruído para barrar todos os pacotes que venham do mundo exterior que estiverem tentando alcançar uma estação interna via a porta 23. Desta forma, comandos de emulação de terminais Telnet que venham de fora serão proibidos.

A maioria dos fabricantes de roteadores suportam características que permitem que administradores da rede estabeleçam uma lista de portas permitidas e proibidas. Outros ainda permitem que se estabeleçam filtros de portas em uma base estação a estação, permitindo desta forma que o administrador da rede controle o acesso com uma granularidade mais fina, e não baseada em toda a organização.

10.6.2. Correio Eletrônico Seguro - PEM

O **Correio Eletrônico Seguro PEM (*Privacy Enhanced Mail)*** ([Mal92b] [Bor93]) é um sistema que fornece criptografia e autenticação para mensagens de correio eletrônico. É um serviço fornecido pela Internet, na forma de um pre-processador. Depois que uma mensagem é preparada pelo PEM, ela é passada adiante para o sistema de tratamento de mensagens SMTP.

Uma questão interessante com relação ao PEM diz respeito ao fato de muitas tecnologias de criptografia desenvolvidas nos Estados Unidos serem consideradas pela Agência de Segurança Nacional (National Security Agency - NSA) americana como tecnologias controladas [Far94a], e que portanto não podem ser exportadas para outros países. O protocolo RSA e o sistema de autenticação Kerberos são bons exemplos de tais tecnologias.

O PEM utiliza o algoritmo de codificação RSA. Assim, nos deparamos com dois problemas de ordem prática com relação ao PEM: em primeiro lugar, como a Internet é uma rede patrocinada por órgãos de pesquisa públicos em todo o mundo, ela é considerada de domínio público, de tal forma que tudo que é usado ou desenvolvido nela pode ser livremente distribuído. Como o RSA é um produto que não está disponível livremente, problemas legais podem surgir com relação às patentes estabelecidas. Em segundo lugar, como a Internet é uma rede global além dos limites territoriais dos Estados Unidos, as mensagens codificadas que forem enviadas para outros países usando tecnologia patenteada, e de uso exclusivo dentro dos Estados Unidos, serão recebidas por pesquisadores e cientistas estrangeiros que certamente não poderão adquirir um software de codificação compatível com o RSA, e portanto não conseguirão decodificar estas mensagens.

A Estrutura do PEM

O PEM apresenta uma estrutura bastante elaborada ([Sch94] [Mal92b] [Den90]). São suportados algoritmos múltiplos de codificação, hash e chaves públicas. Para a codificação do corpo das mensagens de correio eletrônico utiliza-se o DES. RSA é usado para codificar as chaves DES. A mensagem é randomizada por algoritmos de *hash*.

Uma mensagem a ser enviada precisa ser montada seguindo a seguinte ordem: (1) a mensagem em seu formato interno original é representada em um formato adequado para o transporte SMTP baseada em bytes ASCII de 7 bits [Dav89]. (2) A seguir aplicam-se os processos de autenticação e criptografia. A criptografia não precisa ser feita para toda a mensagem, só para partes dela, porém a autenticação é sempre para a mensagem inteira. (3) Finalmente a mensagem é empacotada em uma mensagem normal SMTP, com campos de cabeçalho do tipo "de:/para:" incluídos.

Para que um usuário possa comunicar-se com um colega remoto ele precisa obter um *certificado*, o qual contém campos como: número de versão deste certificado, número serial, assinatura do certificado, nome do emitente, nome do órgão autorizador, data de expiração do certificado, algoritmo de chave pública, etc. Todas estas informações dizem respeito às questões de criptografia, e principalmente, servem para forçar o pagamento das taxas cobradas pela RSA.

A estrutura de certificação é uma floresta (um grupo de árvores). Cada raiz é conhecida como *Autoridade de Certificação de Topo (Top-Level Certifying Authority)*, as quais certificam todas as outras raízes. Exigências de confiabilidade e inspeções rigorosas e elaboradas são impostas às autoridades de certificação de níveis mais baixos. O próprio governo americano é um exemplo de uma autoridade de certificação de topo, e cuida da certificação de organizações governamentais e seus empregados. Outra autoridade de certificação é a empresa RSA Data Security, que age como raiz para usuários Internet interessados no uso do PEM

10.6.3. Pretty Good Privacy - PGP

PGP [Gar95] é um programa de criptografia escrito pelo consultor americano Phil Zimmermann para fornecer mecanismos sofisticados de criptografia para usuários convencionais de informática. Este programa pode ser conseguido facilmente através da Internet ou de BBSs, sem quaisquer custos. O PGP pode ser usado para codificar arquivos ou mensagens de correio-eletrônico. Pode-se também utilizá-lo para fornecer assinaturas digitais que garantem a origem de documentos de uso privado.

Phil Zimmermann começou a escrever o PGP em 1981, e completou a sua primeira versão em 1991. Nesta época, ele forneceu a um amigo uma cópia de seu programa, que o disponibilizou na Internet. O PGP passou então a ser fornecido como software gratuito, porém problemas começaram a ocorrer. Sabemos que programas de criptografia como o RSA ou DES são tecnologia patenteada pela governo americano, e para serem usados precisam de uma autorização expressa para tal. Normalmente paga-se uma alta quantia para se adquirir tal autorização. Como o PGP foi distribuído sem as devidas autorizações para uso do RSA, ele foi considerado software ilegal. Assim, quem quer que utilizasse o PGP nos EUA estaria passível de uma multa por infringir as leis de patentes americanas, e pior ainda, poderia ser preso por violar as leis de tráfico de armas pela exportação do PGP sem a devida autorização do Departamento de Defesa americano.

Apesar da restrições formais para seu uso, o PGP foi disseminado rapidamente por todo os EUA e pelo resto do mundo tornando-se um padrão de criptografia "de facto". No final do ano de 1993, a empresa americana ViaCrypt, dona de uma licença válida para o uso do RSA, negociou com Phil Zimmermann para distribuir uma versão comercial do PGP. Em maio de 1994, uma nova versão do PGP foi liberada, baseada em uma versão do RSA que disponibilizava uma licença válida para o mesmo, liberando seu uso somente para fins não comerciais. Assim, depois de muitos problemas, o PGP é considerado um programa perfeitamente legal nos dias de hoje. Existem até o momento versões do PGP para DOS-Windows, OS/2, Macintosh, Amiga, mainframes Digital com VMS e para a maioria das plataformas Unix.

A seguir apresentamos as principais funcionalidade do PGP:

- Codificação de arquivos. O algoritmo usado para codificar arquivos via PGP chama-se IDEA, e fornece um poderoso mecanismo de criptografia pelo uso de chaves privadas.
- Criação de chaves secretas e de chaves públicas. Estas chaves são usadas para criptografar e sinalizar as mensagens de correio-eletrônico que se deseje enviar pela rede ou que sejam recebidas da rede.
- Gerenciamento de chaves. Pode-se usar o PGP para a criação e manutenção de uma base de dados que contenha as chaves públicas daqueles usuários com os quais um usuário qualquer deseje corresponder-se pela rede.

- Enviar/receber correio criptografado. Pode-se usar o PGP para codificar correio a ser enviado pela rede, ou para decodificar correio recebido pela rede.

- Uso de assinaturas digitais. Pode-se usar o PGP para fornecer assinaturas digitais de documentos. Pode-se também verificar as assinaturas de outros usuários.

- Certificar chaves. A certificação e distribuição de chaves públicas por meio de um sinal eletrônico é possível pelo uso do PGP.

- Habilitar, desabilitar e apagar chaves.

- Configurar o PGP. O programa do PGP permite que se alterem os parâmetros de configuração do mesmo, de tal forma que as opções mais próximas de um usuário em particular sejam escolhidas.

- Permite o uso dos servidores PGP da Internet. Pode-se adicionar chaves à base de dados dos servidores, ou pode-se recuperar chaves de outras pessoas nestes servidores.

A Relação entre o PEM e o PGP

Dentre os mais conhecidos programas de criptografia disponíveis comercialmente podemos citar o SECURE/32, largamente usado na indústria e em grandes corporações, e o MailSafe da empresa RSA Data Security. O PGP tem algumas similaridades com ambos, porém seu grande concorrente é padrão de correio eletrônico da Internet visto na seção anterior, o PEM. As principais diferenças entre o PGP e o PEM são as seguintes:

- Diferentemente do PGP, o PEM é um padrão formal, endossado pela IETF, e não um simples programa. Para que se possa usar o PEM, é necessário que exista um programa de aplicação que "fale" o padrão PEM. Existem vários programas para tal, principalmente em plataformas Unix.

- O PEM não permite que sejam enviadas mensagens de correio anônimas através de seus domínios. No PEM, as mensagens podem ser codificadas e sinalizadas, porém a assinatura deve sempre aparecer fora do envelope de codificação. Qualquer pessoa pode verificar uma assinatura em uma mensagem que tenha sido codificada via PEM, mesmo aquelas pessoas que não conseguem decodificar tais mensagens.

- No PEM, os certificados de chaves públicas precisam ser sinalizados por um sinal de autorização. Por exemplo, cada estudante em uma universidade pode possuir uma chave que tenha sido sinalizada pela universidade. A universidade, por sua vez, deve ter sua própria chave e certificado que devem ser sinalizados por uma autoridade de sinalização de chaves da própria Internet. No PGP, o modelo de certificação de chaves não é centralizado como no PEM, é distribuído; qualquer usuário pode construir sua chave pública e distribuí-la.

Praticamente, todas as diferenças que existem entre o PGP e o PEM decorrem do fato de o PEM ter sido desenhado por um comitê durante vários anos, enquanto o PGP foi construído por um único programador e melhorado por usuários em geral interessados em seu uso. O PEM foi desenhado levado em consideração muitas questões teóricas, enquanto PGP foi projetado para as questões mais triviais do dia a dia.

10.7. Modelos para Protocolos de Autenticação

A autenticação em sistemas distribuídos ([WL92] [Lam91) é sempre viabilizada pelo uso de protocolos. Um protocolo neste contexto define uma seqüência precisa de passos de comunicação e de computação. Um *passo de comunicação* transfere mensagens de um usuário (ou processo de aplicação) para um destinatário. Um *passo de computação* atualiza o estado interno de um usuário/processo. Dois estados distintos podem ser identificados ao término do protocolo: uma autenticação com sucesso ou uma falha.

O formato usado para a apresentação dos os protocolos será o seguinte: Um passo de comunicação onde o usuário/processo ***P*** envia uma mensagem ***M*** para o destinatário ***Q*** será representado por:

P –> Q:M

Um passo de computação de ***P*** será representado por:

P: "especificação do passo de computação"

Por exemplo, o protocolo de login típico entre uma estação ***E*** e um usuário ***c*** está mostrado a seguir:

c --> E: nome do usuário
E --> c: "Digite sua senha:"
c --> E: senha-digitada
E: compute y=f(senha-digitada)
: recupere o registro (c,f(senha-original) a partir do arquivo de senhas
: se y = f(senha-original) then ok; else rejeitar.

Neste exemplo, um usuário informa seu nome, e o sistema requisita que ele digite sua senha. Esta senha é codificada e comparada com a senha original (também codificada) armazenada no arquivo de senhas.

10.7.1. O Sistema de Autenticação Kerberos

Deixamos para falar do Kerberos por último, neste capítulo, porque se o leitor conseguiu chegar até esta seção entendendo todos os conceitos apresentados anteriormente, o que eu espero que tenha ocorrido, ficará bem mais fácil compreender as vicissitudes do Kerberos.

O ***Sistema de Autenticação Kerberos*** ([CB94] [Kay94a] [Sch94] [Mal92b] [Den90] [WL92] [Lam91] [BM90]) foi projetado no MIT como parte do projeto Athena [Gee94] desenvolvido nos anos 80. O sistema recebeu este nome baseado no cão de três cabeças da mitologia grega, que montava guarda na entrada da ilha de Rodes. [304d] O Kerberos tem duas funções principais: autenticação e distribuição de chaves. Ele é um sistema baseado em criptografia, e foi projetado para autenticar usuários e conexões na rede. O algoritmo de criptografia é baseado em chaves partilhadas, e portanto constitui-se como um sistema codificador simétrico [Mal92b]. Seu principal objetivo é fornecer mecanismos de autenticação para plataformas cliente/servidor, permitindo que programas clientes e programas servidores, que operam em uma rede, possam autenticar um ao outro, sem que nenhuma informação seja passada *em claro* pela rede, de forma que elas não possam ser interceptadas.

O Kerberos pode ser considerado *quase* um padrão *de facto* [Kay94a]. Ele prevê sua atuação em um ambiente distribuído baseado em estações de trabalho inseguras, servidores moderadamente seguros e máquinas para a distribuição de chaves de segurança altamente seguras.

O Kerberos é um sistema usado cotidianamente no MIT desde 1987. Ele é também o sistema de autenticação usado pelo sistema de arquivos distribuídos AFS [Coh93] da empresa Transarc Co. Uma importante consideração a respeito do Kerberos é que ele é um dos componentes mais importantes dos servidores de segurança no DCE da OSF ([OSF92d] [Cha94] Mil94c]), que o define como o mecanismo a ser usado na autenticação de usuários e serviços. Muitas plataformas já disponibilizam implementações do Kerberos, seja via DCE, seja via produtos de outros fornecedores como o da empresa *OSCG (Open Computing Security Group) (*[OSF92d] [OSF90c] [OSF91b]). Dentre as plataformas que o implementam podemos citar o Windows NT da Microsoft, Macintosh da Apple, Solaris e SunOs da Sun Microsystems, HP-UX da Hewlett-Packard, NextStep da Next Co., Ultrix e VMS da Digital, AIX, MVS e OS/2 da IBM.

10.7.2. O Ambiente do Kerberos

Na sua concepção original, o principal objetivo do Kerberos era permitir que todos os estudantes do MIT tivessem acesso a todas as estações de trabalho no campus, podendo conectar-se a qualquer destas estações e tendo a mesma visão em qualquer departamento ou laboratório [Gee94].

Neste ambiente de computação distribuída, as estações clientes ficam localizadas em locais públicos, de acesso livre, e são usadas serialmente por diferentes usuários, com diferentes interesses. As estações clientes não são configuradas para conter dados privados ou quaisquer tipos de informações que não possam ser regeradas rapidamente a partir da rede, e portanto, são na sua maioria estações sem disco *("diskless")*. Em essência, estas estações são usadas como front-ends de acesso para que os usuários possam obter serviços da rede.

Todos os arquivos pessoais, assim como todos os programas, residem espalhados pela rede e só podem ser acessados via as transações autenticadas pelo Kerberos.

Os servidores convencionais são gerenciados de maneira centralizada, e ficam localizados em instalações constituídas por salas de computação com segurança moderada. Como os servidores de recursos convencionais ficam localizados em salas com níveis de segurança moderados, um objetivo importante do Kerberos é garantir que um ataque a um servidor não comprometa os outros servidores.

Os *centros de distribuição Kerberos* (**KDCs** - *Kerberos Distribution Centers*) [WL92] ficam localizados em salas com altos níveis de segurança física, e são guardados criteriosamente. Os KDCs acomodam dois tipos de servidores para o Kerberos: os *servidores de autenticação* e os *servidores de tickets* **TGSs** ("Ticket Granting Servers").

As estações de trabalho ficam localizadas em locais públicos, disponíveis para quaisquer tipos de usos, pelos mais diferentes tipos de usuários: professores, estudantes, funcionários, visitantes, etc. Portanto estas estações são completamente inseguras, de tal forma que uma estação de trabalho sozinha não pode garantir a identidade de um usuário para um servidor.

Como os usuários são em sua grande maioria pouco sofisticados com relação às preocupações com questões de segurança em computadores, não existem arquivos com senhas codificadas armazenados localmente em nenhuma estação pública. O principal objetivo do Kerberos é fornecer um ambiente altamente seguro, mesmo que a rede esteja sofrendo constantes ameaças por parte dos intrusos potenciais.

10.7.3. A Estrutura do Kerberos

Cada servidor e cada usuário Kerberos são registrados em uma base de dados no centro distribuição KDC ([Sch94] [WL92] [CB94]). Tanto servidores de autenticação quanto servidores TGS irão ler as informações nesta base de dados durante o processo de autenticação. Neste processo são usados dois protocolos:

Protocolo 1: Protocolo de inicialização de credenciais;
Protocolo 2: Protocolo de autenticação cliente/servidor.

O *protocolo 1* usa os servidores de autenticação. O *protocolo 2* usa os servidores TGS. Vejamos cada um destes protocolos a seguir.

Protocolo 1

Quando um usuário se conecta a uma estação, ele fornece sua identificação (seu nome) e sua senha secreta. O nome do usuário é então usado para requisitar um ticket a um servidor de tickets (falamos mais detalhes sobre tickets daqui a pouco) e sua senha é convertida, por meio de um algoritmo DES, em um número randômico de 64 bits, que será a chave secreta (***Kc***) deste usuário para requisitar serviços ao Kerberos.

Para cada usuário armazena-se seu nome, juntamente com sua chave secreta devidamente criptografada na base de dados do KDC. Como é difícil para um usuário humano guardar um número de 64 bits, ele faz uso de sua senha de acesso convencional (não codificada) que somente será conhecida por ele (e pela base de dados). Os servidores armazenam suas senhas em arquivos protegidos contra leituras não autorizadas. A figura 10.3 mostra os campos de uma base de dados KDC.

nome	instância	atributos	chave	versão da chave	data de expiração	última modif	autor da última modif
Suzana		0	0xf4672389 2056	2	31/12/99	10/04/94	Helvécio

Figura 10.3 Uma entrada na base de dados Kerberos [Sch94]

Os dois protocolos estão mostrados graficamente na figura 10.4. A idéia básica é que quando um usuário ***c*** deseja acessar um serviço ***s***, este usuário deve adquirir um *ticket fornecedor de tickets* por meio de um pedido enviado para o servidor de autenticação Kerberos localizado no KDC, detalhando seu nome e o nome do serviço desejado. O servidor de autenticação Kerberos constrói o *ticket fornecedor de tickets* e uma chave de sessão DES (***Kc,s***). Ambos são codificados com a chave DES do usuário (***Kc***), e retornados para a estação requisitante.

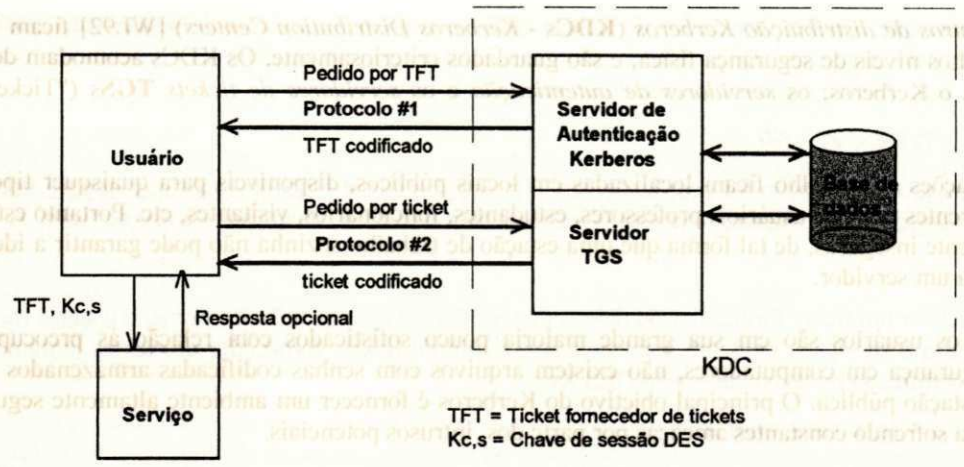


Figura 10.4. O protocolo Kerberos [CB94]

O protocolo 1 usa portanto o servidor de autenticação Kerberos (***saK***), e está especificado abaixo:

```
c --> E    : nome-do-usuário
E --> saK: c,s (nome-do-usuário, serviço)
saK          : recupera Kc e Ks da base de dados
               : gera uma chave de sessão Kc,s
               : cria um ticket fornecedor de tickets T (i)
saK -->E : {T, Kc,s} Kc (ii)
E --> c    : "digite sua senha:"
c --> E    : senha
E            : compute w=f(senha)
               : recupere Kc,s e T pela decodificação de {T, Kc,s} Kc com w
               : se a decodificação falhar, aborte o login
               : senão, retenha T e Kc,s
               : apague senha da memória
```

A estação de trabalho irá perguntar ao usuário por sua senha (na tela), a qual será traduzida em uma chave DES (*Kc*) e usada para decodificar o ticket fornecedor de tickets e a chave de sessão DES (*Kc,s*) retornados. Se a senha for fornecida incorretamente, então uma descrição apropriada não será gerada.

Se considerarmos que o ticket fornecedor de tickets e a chave de sessão DES (*Kc,s*) tenham sido decodificados corretamente com o uso da chave do usuário (*Kc*), então o usuário poderá construir um *Autenticador* para requisitar o serviço *s*, e o enviar juntamente com o ticket fornecedor de tickets para o servidor TGS, para requisitar agora um *ticket servidor* para o serviço *s*. O autenticador é necessário porque o ticket fornecedor de tickets sozinho pode ser copiado ou interceptado, e portanto não constitui prova suficiente de identidade.

Vejamos agora o conteúdo de um ticket:

T = {c, s, end, time, exp; Kc,s} Ks (i)

O Ticket T contém o nome do usuário, *c*, o nome do serviço requisitado, *s*, o endereço da estação, *end*, a hora de criação, *time*, e sua data de expiração, *e*. Esta informação, juntamente com a chave de sessão ***Kc,s***, é codificada a seguir em ***Ks***, uma chave DES secreta para o serviço *s* (não confundir com a chave Kc,s). Este ticket fornecedor de tickets será usado para requisitar tickets de servidores ao TGS.

A notação empregada é muito simples. Colocando-se os termos entre chaves ({}), indica-se que estes termos estão codificados usando-se a chave de codificação que é representa pelo sufixo fora das chaves. Observe que o conteúdo do ticket é completamente ilegível para o usuário que o recebe, pois este usuário não conhece a chave *Ks* para o serviço.

Lembre-se que o ticket fornecedor de tickets também contém uma chave de sessão Kc,s que foi gerada randomicamente pelo algoritmo DES. Ela é retornada para o usuário, codificada, juntamente com o ticket fornecedor de tickets. Assim, o usuário recebe do servidor Kerberos o seguinte:

{T, Kc,s} Kc (ii)

Note que esta informação foi codificada com *Kc*, que é a chave DES do usuário, e que foi originalmente derivada a partir de sua senha. Assim, *Kc,s* está disponível para o usuário, bloqueada dentro do ticket T.

Protocolo 2

Após receber um ticket fornecedor de tickets do servidor de autenticação Kerberos, o usuário irá autenticar uma transação para o serviço *s*, enviando para este uma cópia do ticket fornecedor de tickets juntamente com o autenticador ***A***, codificado na chave de sessão DES Kc,s:

A = {c, s, time, end} Kc,s

Diferentemente do ticket fornecedor de tickets, que é gerado pelo servidor de autenticação Kerberos, o autenticador é gerado pela própria estação de trabalho. Ele contém o nome do usuário, *c*, o nome do serviço desejado, *s* , a hora corrente, *time*, e o endereço da estação, *end.*

Vimos no *protocolo 1* que, para completar uma transação autenticada pelo Kerberos, a chave *Kc* do usuário é requerida para decodificar a primeira resposta que vem do servidor Kerberos. Neste momento, ou o usuário digita sua senha em um *prompt* na tela, ou a estação de trabalho armazena em memória *cache* este valor para usar quando for necessário. Pois bem, nenhuma destas duas situações são recomendáveis, pois geram pontos de vulnerabilidade para o sistema de autenticação.

Para solucionar este problema, foi criado o *serviço de fornecimento de tickets* (**TGS** - *ticket granting service*), o qual roda no mesmo servidor Kerberos e tem acesso permitido às bases de dados no KDC. Sua funcionalidade está descrita a seguir.

Quando um usuário se conecta pela primeira vez a uma estação, ele utiliza o protocolo de autenticação Kerberos *(protocolo 1)* para obter um ticket fornecedor de tickets para o serviço TGS, o que irá requisitar o fornecimento de sua senha de login. No entanto, quando tickets de servidores são requisitados, no *protocolo 2*, ao invés de usar o serviço de autenticação normal do Kerberos, a estação cliente emite um pedido de ticket de servidor para o TGS, requisitando tickets de servidores adicionais (com suas chaves de sessão correspondentes). Estes tickets de servidores retornam para a estação cliente, agora codificados pela chave de sessão do ticket TGS (e não pela chave de usuário *Kc*).

Assim, a única informação secreta que a estação precisa armazenar localmente é a chave de sessão ***Kc,tgs*** que é necessária para ser usada pelo TGS.

Quando um servidor TGS recebe um ticket fornecedor de tickets e um autenticador, ele primeiramente decodifica o ticket fornecedor de tickets usando sua chave DES secreta *Ks*, e certifica-se de que o usuário *c* está fazendo um pedido autenticado. A seguir, ele procura por seu nome *s* no ticket fornecedor de tickets (e gera um erro se ele não o encontrar) e depois verifica se o ticket é válido (ou seja, não expirou sua validade) e se foi despachado do endereço que aparece no ticket. Finalmente, se o ticket estiver correto, a chave de sessão *Kc,s* é removida e usada para decodificar o autenticador. Se o autenticador for decodificado corretamente, então o nome do cliente é comparado com o nome do cliente no ticket. O endereço no ticket também é comparado com o endereço no autenticador. Os dois devem ser os mesmos. Finalmente, o rótulo de tempo (*"timestamp"*) no autenticador precisa estar com somente alguns minutos de diferença para o tempo corrente. Se todas estas condições forem observadas, então assume-se que a transação foi gerada originalmente a partir do usuário c, e protanto, o serviço pedido poderá ser atendido.

Considerações Adicionais

O ponto forte do Kerberos é que se um intruso não conhecer a chave gerada a partir da senha de um usuário (Kc), ele não poderá burlar o sistema. Da mesma maneira, se um intruso gravar uma série de transações, nenhuma dica será fornecida para permitir que este intruso tente executar uma transação similar. Mesmo que dados idênticos sejam enviados pelo intruso, eles serão rejeitados, por causa do rótulo de tempo no autenticador, que certamente já terá expirado. Portanto, o Kerberos cuida de fornecer os mecanismos de contra-ataque necessários para combater os mais sofisticados tipos de ataques em sistemas de codificação.

A Administração do Kerberos

Um domínio administrado em comum e utilizando o mesmo KDC é chamado de *área (realm)* [Sch94]. Operações administrativas típicas, como a adição de novos usuários, ou a modificação de senhas de usuários já existentes é executada por um protocolo autenticado pelo Kerberos através do super usuário da instalação. O servidor KDC deve ser dedicado, e somente um número limitado de indivíduos autorizados devem ter acesso à bases de dados KDC. Administradores de sistemas comuns que desejem executar operações administrativas, como a adição de novos usuários, não precisam ter acesso direto a esta base de dados, mas podem usar um cliente autenticado *admin*. O cuidado é importante porque simplesmente todas as chaves DES para todos os usuários e servidores residem no KDC. Se for feita uma cópia não autorizada da base de dados do KDC, então todas as senhas de todos os usuários estarão comprometidas, e terão que ser modificadas.

10.7.4. Limitações do Kerberos

Vale a pena lembrarmos que o Kerberos não é uma solução de segurança completa [Kay94a]. Mecanismos de autorização (listas de controle de acesso, gerência de senhas) e auditoria não são fornecidos para aplicações ou transações, e portanto devem ser fornecidos por outros sub-sistemas na rede. O Kerberos é somente parte do quebra-cabeças de segurança. É um sistema pesado, consumidor de tempo e gerador de muito tráfego na rede, ideal para plataformas de alto desempenho e conexões de altas velocidades com altas bandas passantes. No MIT o ambiente original é baseado em estações de trabalho de alto desempenho, redes de alta velocidade (FDDI a 100 Mbpss) e servidores de vários tipos [WL92].

Portanto, apesar de sua extrema utilidade, o Kerberos apresenta pontos fracos e limitações ([CB94] [BM90]). Em primeiro lugar, o Kerberos é desenhado para efetuar autenticação do tipo usuário-estação, e não estação-estação. Esta funcionalidade é razoável no ambiente Athena do MIT, porém em ambientes como redes locais convencionais ele é considerado inadequado.

Outro problema são as caches usadas para tickets e chaves de sessão. Qualquer pessoa que possa ler estes valores em memória cache pode usar estas informações para agir como se fosse o usuário legítimo. O ticket pode ser capturado pela escuta na linha, ou pela obtenção de um estado privilegiado na estação. O problema principal, no entanto, fica do conta da maneira de se obter o ticket inicial. O pedido inicial para um ticket fornecedor de tickets não contém informações de autenticação, como por exemplo uma cópia codificada do nome do usuário. A resposta é portanto um bom alvo para os analisadores criptográficos de escuta. Existe ainda outro problema relacionado com o login: como é que o usuário irá garantir que seu próprio login não foi interceptado? A forma usual de se evitar estes tipos de ataques é por meio de equipamentos desafio/resposta, que não são suportados pelo protocolo corrente.

10.8. Resumo dos Pontos Fracos em Ambientes Distribuídos

Existem muitos pontos fracos que podem comprometer a segurança de um ambiente distribuído. Nas seções anteriores citamos vários deles. A seguir apresentamos uma lista contendo um resumo das vulnerabilidades mais comuns que podem comprometer seriamente os sistemas de computação que venham a sofrer ataques terroristas, sejam eles internos ou externos [Bor93]:

. Mecanismo de auto-bloqueio ausente
. Ataques externos
. Portas corta-fogo mal administradas
. Muitas conexões inseguras disponíveis (p. ex.: linhas discadas via modens)
. Transmissão de informações sensiveis não criptografadas
. Indiferença às questões de segurança
. Medo dos custos que podem surgir para se garantir altos níveis de segurança
. Senhas fracas e óbvias
. Usuários não validados ou não autenticados
. Software novo não validado (inspecionado)
. Áreas de RAM/discos não são reformatados após o seu uso
. Ataques violentos de vírus
. Pessoal interno muito hostil e mal intencionado
. Erros de programação (*"bugs"*)
. Exploração de vulnerabilidade em pacotes de software
. Uso de aplicações não autorizadas [Gre94b]
. Equipamentos e mecanismos de *backup* inadequados ([Hay93a] [Koe93])
. Falta de mecanismos de controle de acesso (p.ex: litas de controle de acesso)
. Falta de controles de auditoria e, falta de procedimentos anti-vírus.

10.8.1. Resumo das Dicas para Segurança em Ambientes Distribuídos

Para enfrentar as ameaças listadas na seção anterior, segue um resumo das dicas que podem ser essenciais para a garantia da segurança em ambientes da computação distribuída ([Kay94a] [Sch94] [OSF92d]):

Consciência: deve-se estabelecer uma política organizacional para acesso aos computadores, às informações, e para a distribuição de responsabilidades. Os usuários devem estar conscientes de sua participação na proteção das informações da organização. *Acesso*: precauções adicionais devem ser tomadas com relação às conexões externas. Utilize de preferência portas corta-fogo. *Autenticação*: as senhas devem ser trocadas regularmente. Se possível devem ser utilizados *passes* ou geradores aleatórios de senhas. *Confidencialidade*: os dados a serem transmitidos pela rede devem ser criptografados. Informações armazenadas em disco que sejam sensiveis (por exemplo, arquivos de senhas) ou críticas (por exemplo, arquivos de configuração) devem ser criptografados. *Administração*: deve-se estar apto a administrar a segurança a partir de qualquer ponto na rede. *Disponibilidade*: todos os servidores devem ser equipados com equipamentos do tipo no-breaks. Deve-se utilizar facilidades como RAIDs (Redundant Array of Inexpensive Disks - Arranjo Redundande de Discos Baratos) para que se garanta a recuperação ou o backup de arquivos em-linha. Mantenha arquivos de backup em sítios remotos. Estabeleça mecanismos de tolerância a falhas, como duplicação de discos ou de servidores.

10.9. A Padronização da Segurança

O Centro Nacional de Segurança para Computadores, uma agência do Departamento de Defesa americano (DoD - Department of Defense), publicou em 1985 um documento entitulado "*Department of Defense Trusted Computer System Evaluation Criteria*", também conhecido como "*Orange Book*" (livro laranja, devido à cor alaranjada de sua capa) ([Bor93] [Far94b] [OSF92d] [Sch94]). Este documento tem até hoje fortes influências na indústria de computadores, e estabelece os padrões e exigências para os sistemas seguros que são aprovados pelo DoD. Para que qualquer órgão do governo federal americano possa adquirir um sistema de computação, um nível mínimo de segurança como especificado no livro laranja deve estar implementado para este sistema.

O livro laranja agrupa a segurança em sistemas de computação em quatro divisões: D, C, B, e A, onde esta última divisão (A) representa aqueles sistemas com o maior grau de segurança. As divisões com maior grau de segurança incorporam as atribuições das divisões menores. Assim, a divisão B deve apresentar as proteções discriminatórias da divisão C, adicionalmente aos seus próprios critérios de segurança. As divisões B e C são subdivididas em subseções conhecidas como classes. Estas classes também são ordenadas de maneira hierárquica. A tabela 10.1 sumariza os critérios das divisões e classes do livro laranja [CTR92a].

Divisão	Classe	Descrição
D: Proteção Mínima	-	Qualquer sistema que não atenda às exigências das divisões A, B, ou C.
C: Proteção Discriminatória	**C1**	Proteção de segurança discriminatória: Os sistemas devem incorporar alguma forma de controle capaz de forçar os limites de acesso em uma base individual. O usuário decide quais proteções devem ser forçadas, como acontece por exemplo nas permissões de acesso a arquivos em sistemas Unix.
	C2	Proteção com acessos controlados: os sistemas nesta classe devem forçar controles de acesso mais sofisticados que aqueles de nível C1, tornando os usuários individualmente contabilizáveis pelas suas ações, por meio de procedimentos de login, auditoria de eventos, e isolamento de recursos.
B: **Proteção Obrigatória**	**B1**	Proteção de segurança rotulada: apresenta todas as características da classe C2, e uma declaração informal da política do modelo de segurança, dos rótulos de dados, e do controle de acesso obrigatório aos serviços nomeados e objetos deve estar presente. O sistema deve implementar algum nível de proteção que não está sob o controle do usuário.
	B2	Proteção estruturada: inclui todos os controles de B1, e trata de "canais escondidos".
	B3	Domínios Seguros: deve satisfazer às exigências do monitor de referência que cuida de todos os acessos a serviços por objetos, e deve ser pequeno o suficiente para permitir análises e testes.
A: **Proteção Verificada**	A1	Projeto verificado: é funcionalmente equivalente ao nível B3, exceto que o usuário deve provar que o modelo de segurança e a sua implementação são seguros de fato.

Tabela 10.1. Os critérios das divisões e classes do Livro Laranja [CTR92a]

Capítulo 11: Sistemas de Processamento de Transações Em-Linha (Sistemas OLTP)

11.1. Introdução

Os primeiros sistemas **OLTP (On Line Transaction Processing)** ([Cla92b] [OV91] [Dew93] [Dew94] [Cas93] [CP84] [Gra91]) foram implementados em meados dos anos sessenta pelas empresas aéreas que buscavam disponibilizar aplicações confiáveis para reservas de passagens. Nos anos setenta muitas aplicações OLTP foram implementadas: sistemas para reservas em hotéis, aluguel de carros, automação bancária com terminais ATM (Automatic Teller Machine), transferência eletrônica de fundos, sistemas de cartões de crédito, sistemas de compra e venda de ações em corretoras e bolsas de valores. Atualmente, o uso de aplicações OLTP espalhou-se por: hospitais, que implementam sistemas para controle clínico e administrativo; jornais, com sistemas de suporte à editoração; fábricas, com sistemas de controle da produção baseados em técnicas como "just in time".

Todas as aplicações baseadas em sistemas OLTP possuem duas características em comum: exigem ambientes operacionais que estejam operacionais 100 % do tempo, e não podem permitir que se percam quaisquer tipos de dados. Estas aplicações são o que convencionou-se chamar de *aplicações críticas*. A indisponibilidade de uma delas, mesmo durante pequenos intervalos de tempo, pode representar enormes prejuízos para uma empresa.

11.1.1. Transações

Uma transação consiste da execução de um programa (ou de um segmento de programa) que implementa uma função sobre um ou mais objetos. Uma transação pode ser considerada uma abstração gerenciada por um sistema de gerenciamento de transações, chamado de **sistema TP (Transaction Processing)** ([Cla92b] [OV91] [Dew93] [Dew94] [Cas93] [CP84] [Gra91]). Uma transação é uma unidade atômica de processamento que manipula objetos do sistema. Pode-se modelar uma transação como uma unidade atômica da seguinte forma:

T = a1 a2 a3....an

onde ***ai*** são ações (operações) de *leitura, gravação, remoção, regravação, abertura, fechamento, início-de-tx, fim-de-tx, commit-de-tx, aborto-de-tx,* e muitas outras (*tx* = transação). A ação ***a1*** deve ser sempre início-de-tx obrigatoriamente, assim como a última ação ***an*** deve ser ou commit-de-tx, ou aborto-de-tx. Nenhuma das ações intermediárias ***ai*** poderá ser ou início-de-tx, ou commit-de-tx. Na figura 11.1 apresentamos um exemplo para uma transação simples de débito/crédito.

```
read 100 bytes from terminal;
start_trans();
        /* Atualiza saldo na conta  A  */
    saldo_conta = saldo_conta + delta;
    rewrite novo_saldo_conta to conta A;
        /* grava registro de histórico  */
    log  A, T, B, delta, timestamp;
        /* atualiza saldo no terminal T  */
    saldo_terminal = saldo_terminal + delta;
        /*  atualiza saldo_agência B  */
    saldo_agência = saldo_agência  + delta;
commit_trans();
write 200 bytes to terminal;
```

Figura 11.1. Uma transação simples de débito/crédito

Na transação de débito/crédito da figura 11.1 uma mensagem de entrada é recebida de um terminal de automação bancária (tipo um caixa 24 horas), debita ou credita uma conta, guarda em um arquivo de *log* o número da conta (A), o identificador do terminal (T), o identificador da agência (B), o valor, e um rótulo de tempo ("timestamp") com a data e a hora em que a transação se completou, ajusta o saldo de caixa para a agência e para o terminal, e finalmente, envia uma mensagem de saída para o terminal. Nesta transação, a mensagem de entrada é composta pelo número da conta, identificador do terminal, identificador da agência, e o valor que se deseja debitar/creditar. A mensagem de saída irá conter: número da conta, identificação do terminal, identificação da agência, valor debitado/creditado e o resultado final da operação.

Cada transação em um sistema TP recebe um identificador chamado **TID** (Transaction Identifier), que será usado para referênciá-la enquanto ela existir.

A ação *commit-de-tx* da qual falamos anteriormente indica a finalização com sucesso de uma transação. Quando um *commit* é executado todas as atualizações efetuadas pela transação tornam-se visíveis para as outras transações no sistema. As transações possuem quatro propriedades fundamentais chamadas propriedades ***ACID*** [Cla92b]: Atomicidade, Consistência, Isolamento, e Durabilidade. A seguir, caracterizamos sucintamente cada uma destas propriedades.

- *Atomicidade*: se uma transação alterar um objeto partilhado, esta alteração deverá ser feita obedecendo a uma política de ou fazer tudo, ou então não fazer nada. No caso de uma transação abortar, ela deve desfazer todas as alterações efetuadas antes do aborto. Desta forma, ou todas as ações de uma transação ocorrem com sucesso, ou a em caso de aborto, a transação deve retornar todas os objetos alterados para seu estado original (operação de "rollback").

- *Consistência*: uma transação pode alterar um objeto, por exemplo uma base de dados, de um estado válido para outro estado válido. No caso desta transação abortar, a base de dados deve ser recuperada para o seu estado válido original.

- *Isolamento*: os efeitos de uma determinada transação devem ser transparentes para as demais transações enquanto a tarefa sendo executada por ela não tiver sido concluída (ou seja, enquanto não ocorrer uma ação do commit-de-tx finalizando-a).

- *Durabilidade*: a partir do momento que uma transação tiver concluído seu trabalho (commit), seu efeito sobre os objetos envolvidos permanecerão inalteráveis, mesmo que ocorram falhas posteriores no ciclo de processamento.

11.1.2. O Controle de Concorrência e Recuperação

Duas questões básicas que merecem destaque no processamento de transações em linha são o *controle de concorrência* e a *recuperação de informações.*

O ***controle da concorrência*** é a atividade de coordenação e resolução de conflitos que podem ocorrer quando duas ou mais transações acessam simultaneamente o mesmo objeto partilhado. Os algoritmos para controle de concorrência impõem a execução de transações *serializadamente*. A execução de uma série de transações {Ti|1<i<=n} é serializável se esta execução for computacionalmente equivalente à execução serial de cada uma destas transações. Isto é, um conjunto de transações é serializável se sua execução produzir os mesmos efeitos que seriam produzidos por alguma execução serial das transações uma a uma, e não em conjunto.

A ***recuperação de informações*** é a atividade que cuida de garantir que falhas de hardware ou de software que ocorram durante a execução de uma transação não irão comprometer os dados persistentes, ou seja, aqueles dados que foram "commited" (propriedade de "durabilidade"). Como dissemos há pouco, desde que uma transação seja "commited", os dados atualizados por ela devem ser duradouros (ou persistentes).

11.1.2.1. Problemas com o Processamento de Transações

As aplicações distribuídas envolvem vários processos acessando um mesmo objeto partilhado. Um bom exemplo de tal objeto é um sistema de arquivos em uma aplicação bancária contendo contas correntes de clientes. Geralmente, estas contas são acessadas ao mesmo tempo por vários sistemas clientes interessados em efetuar operações de débito/crédito. Para ilustrar os problemas que podem surgir em tais aplicações, considere a seqüência de operações, mostradas na figura 11.2, executadas por dois sistemas clientes

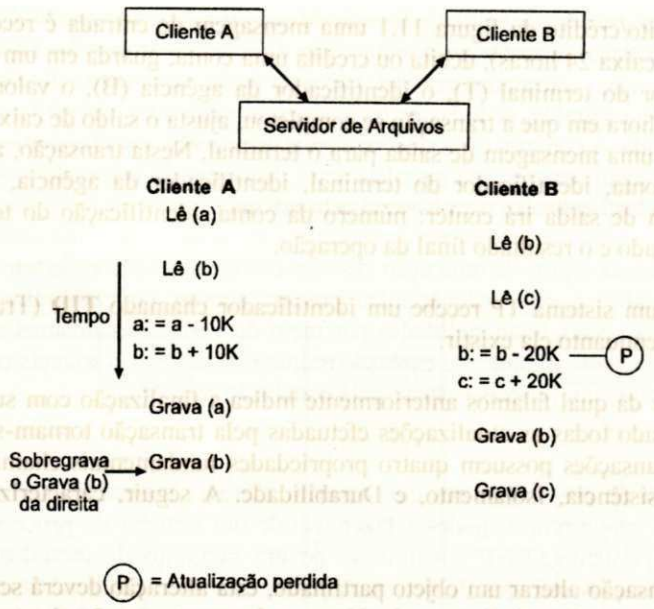


Figura 11.2. Esquema de atualizações perdidas

Neste exemplo, assumimos que o cliente A esteja envolvido com a transferência de 10 K (reais, dólares, etc.) da conta corrente (a) para a conta corrente (b). Simultaneamente, o cliente B irá transferir 20 K da conta (b) para a conta (c). Cada cliente lê as duas contas apropriadas no servidor, executa as transferências correspondentes, e retorna as contas atualizadas para o servidor. Podemos ver que o cliente A sobregrava a atualização efetuada pelo cliente B na conta (b). Isto é conhecido como uma "atualização perdida". Sem os mecanismos apropriados de controle de concorrência, tais perdas podem ocorrer em todas as aplicações que envolvam o acesso concorrente a objetos partilhados.

Outro problema que pode ocorrer quando múltiplas cópias de um arquivo são mantidas em diferentes localizações é que se estas cópias não forem idênticas (consistentes) é possível que em uma aplicação bancária similar àquela do exemplo um cliente possa completar um saque apesar de não ter mais fundos (pois outros saques podem ter sido efetuados anteriormente em outras agências). Este problema é conhecido como "problema de atualização em múltiplas cópias". Estes problemas são comuns em aplicações distribuídas.

Devido a problemas deste tipo, existe a necessidade explícita de que sejam empregados mecanismos que possam evitá-los. Tais mecanismos são normalmente conhecidos como mecanismos para *controle de concorrência* e mecanismos para *recuperação de informações* ([Hal93b] [Cla92b] [OV91] [Dew93] [Dew94] [Cas93]).

Estes mecanismos baseiam-se no conceito de ***"ação atômica"*** ([Cla92b] [OV91] [Cas93]). Essencialmente, uma ação atômica é uma seqüência de operações que são efetuadas por duas ou mais transações cooperantes, de tal forma que:

- A seqüência de operações é efetuada sem que ocorram interferências de transações que não sejam parte da ação atômica;
- As operações de cada transação em uma ação atômica ou são todas completadas com sucesso ou então são todas abortadas; em caso de aborto, quaisquer informações que tenham sido alteradas durante a execução destas operações devem ser restauradas para um estado válido anterior ao início da ação atômica.

A transação que inicia uma ação atômica é conhecida como *transação Mestre* (ou *transação Superior)*, e é ela que está envolvida com o controle de todas as atividades associadas com a ação atômica. De maneira similar, todas as outras transações envolvidas na ação atômica são conhecidas como *escravas* ou *subordinadas*, pois elas são todas controladas pela transação mestre que iniciou a ação atômica.

Todos os dados afetados pelas operações em uma ação atômica são conhecidos como "dados limítrofes" [Cla92b]. Os valores dos dados limítrofes no início de uma ação atômica representam seu *estado inicial* e estes valores após a ação atômica representam seu *estado final*. Quando uma ação atômica termina, seus dados limítrofes ou são aceitos ("commited") para seu estado final, no caso de todas as operações terem sido efetuadas com sucesso, ou então são recuperados para seu estado inicial, no caso de uma ou mais das operações falharem.

Pode-se implementar esta funcionalidade através de um *protocolo de aperto de mãos* ("handshake"): após a execução de todas as operações associadas a uma ação atômica, a transação superior pergunta a todas as transações subordinadas se elas completaram com sucesso; neste momento, diz-se que as transações estão preparadas para o commit. Em seguida, se todas as transações subordinadas responderem positivamente, é dada a ordem para que seja efetivado o commit de todos os dados limítrofes. Se entretanto, uma ou mais transações subordinadas responderem negativamente (ou simplesmente deixarem de responder), será dada uma ordem para que se restaurem todos os dados limítrofes para o seu estado inicial.

Para que se respeitem as regras de uma ação atômica é necessário que cada uma das transações envolvidas evitem que transações estranhas a esta ação atômica possam acessar ou manipular os seus dados limítrofes. Normalmente implementam-se estas funcionalidades por meio de vários mecanismos de sincronização ([Mil87] [Bal89]) como os *semáforos, os* mecanismos de exclusão mútua (*"mutexes"),* variáveis condicionais, ou técnicas de "*handez-vous*" para permitir que um processo espere pela conclusão de outro.

11.2. Sistemas OLTPs

Para facilitar a vida de programadores que desejam escrever transações sofisticadas para atender a um grande número de usuários concorrentes, pode-se fazer uso de um sistema de processamento de transações em linha (*sistema OLTP*). Um sistema OLTP constitui-se de um conjunto de módulos de software que, usados agrupadamente, podem reduzir os esforços de programação exigidos no desenvolvimento de aplicações TP em-linha chamadas **OLTAs** (On Line Transaction Applications) ([Cla92b] [Dew94] [Cas93]). Uma OLTA é uma coleção de programas executáveis que fornecem serviços de transações simultaneamente para centenas ou possívelmente milhares de usuários.

Em ambientes distribuídos, um componente importante para um sistema OLTP completo é o *subsistema de comunicações,* que fornece serviços para permitir o acesso a outros sistemas em máquina remotas. O subsistema de comunicações fornece os serviços de apresentação, entrega de mensagens, e interfaceamento com o software da rede. A figura 11.3 ilustra os principais componentes de um sistema OLTP típico.

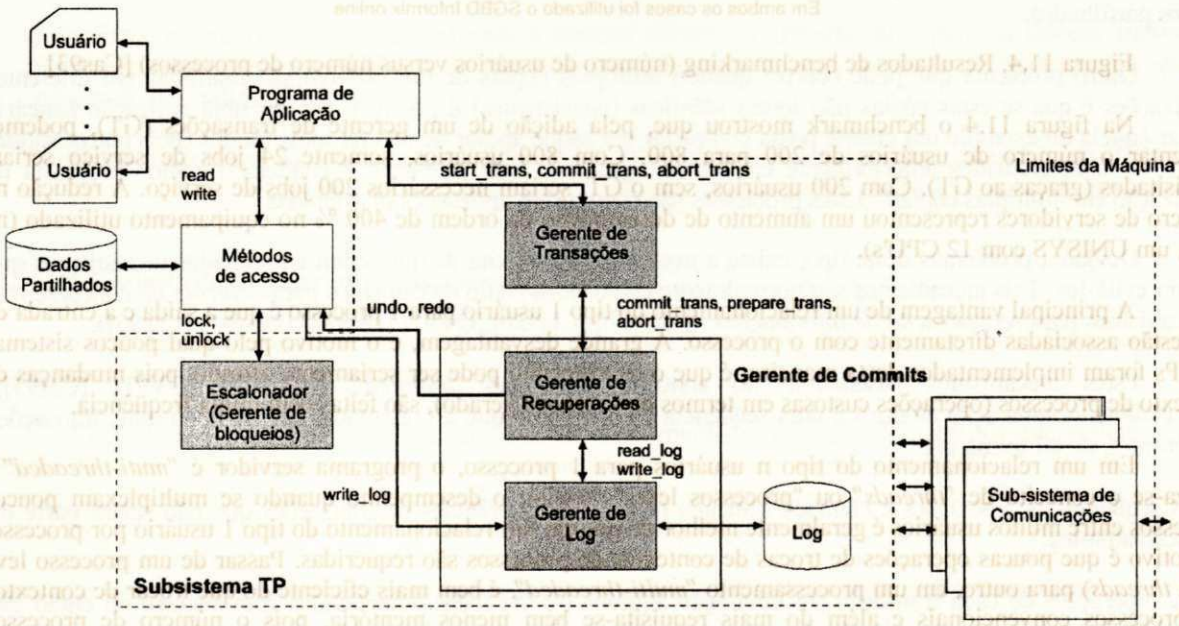


Figura 11.3. Um modelo para sistemas OLTP [Cla92b]

Três considerações merecem destaque nos sistemas OLTPs: o nível de relacionamento com o sistema operacional; o nível de relacionamento entre processos e usuários; o nível de segurança.

11.2.1. O Relacionamento entre o Sistema Operacional e o Sistema OLTP

A distribuição da funcionalidade de um sistema OLTP pode ser avaliada do ponto de vista de quais de seus módulos devem fazer parte do sistema operacional. Vários serviços como segurança, monitoramento de desempenho, controle de bloqueios, dentre outros, já existem disponíveis na maioria dos sistemas operacionais. Portanto, ao invés de serem replicados tais serviços em sistemas OLTP, é melhor que se utilizem aqueles já disponíveis nos sistemas operacionais.

Um argumento contrário à inclusão de funcionalidades OLTP em sistemas operacionais diz respeito ao fato de se poder tratar o nível de granularidade dos dados com maior precisão se o escalonador de transações for desenvolvido separadamente do escalonador do próprio sistema operacional.

11.2.2. O Relacionamento entre Processos e Usuários

Podemos considerar dois casos: no primeiro caso temos um relacionamento do tipo 1 para 1, ou seja, um processo de sistema operacional por usuário. No segundo caso alguns processos do sistema operacional são multiplexados entre vários usuários.

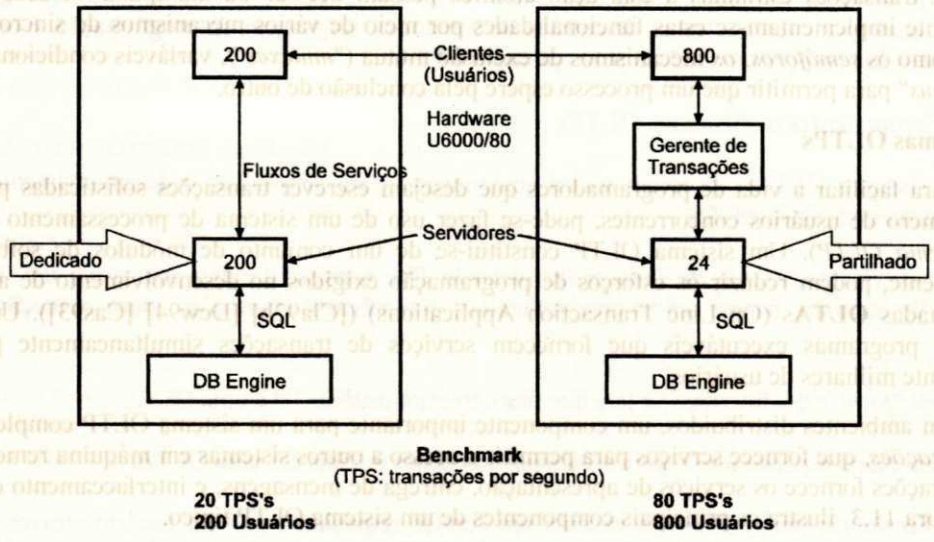


Figura 11.4. Resultados de benchmarking (número de usuários versus número de processos) [Cas93]

Na figura 11.4 o benchmark mostrou que, pela adição de um gerente de transações (GT), podemos aumentar o número de usuários de 200 para 800. Com 800 usuários, somente 24 jobs de serviço seriam requisitados (graças ao GT). Com 200 usuários, sem o GT, seriam necessários 200 jobs de serviço. A redução no número de servidores representou um aumento de desempenho da ordem de 400 % no equipamento utilizado (no caso, um UNISYS com 12 CPU's).

A principal vantagem de um relacionamento do tipo 1 usuário para 1 processo é que a saída e a entrada da tela estão associadas diretamente com o processo. A grande desvantagem, e o motivo pelo qual poucos sistemas OLTPs foram implementados desta maneira, é que o desempenho pode ser seriamente afetado, pois mudanças de contexto de processos (operações custosas em termos do overhead gerado), são feitas com muita freqüência.

Em um relacionamento do tipo n usuários para 1 processo, o programa servidor é *"muti-threaded"* - utiliza-se o conceito de *"threads"* ou "processos leves". Assim, o desempenho quando se multiplexam poucos processos entre muitos usuários é geralmente melhor do que em um relacionamento do tipo 1 usuário por processo. O motivo é que poucas operações de trocas de contextos de processos são requeridas. Passar de um processo leve (com *threads*) para outro, em um processamento *"multi-threaded"*, é bem mais eficiente do que trocar de contextos em processos convencionais e além do mais requisita-se bem menos memória, pois o número de processos envolvidos é bem menor.

11.2.3. Segurança em Sistemas OLTP

As aplicações desenvolvidas em sistemas OLTP (ou seja, as OLTAs - On Line Transaction Applications) podem ser projetadas para validar a identificação dos usuários da seguinte maneira: (a) os usuários não são autenticados; (b) os usuários são autenticados pelo sistema operacional; (c) os usuários ou são autenticados pelo sistema operacional e/ou pela OLTA.

No caso (a) onde os usuários não são autenticados as transações podem ser inicializadas assim que o menu de seleção de transações aparecer na tela. É uma situação atípica, onde não existe a mínima preocupação com a segurança no uso dos recursos do sistema.

No caso (b), uma OLTA assume o papel de dona do processo, e os usuários são autenticados individualmente pelo próprio sistema operacional. Assim, um usuário pode se conectar normalmente ao terminal, e em seguida requisitar uma aplicação OLTA. O processo de usuário fica separado do processo da OLTA.

Em (c), a OLTA autentica os usuários, como acontece em terminais de automação bancária do tipo ATMs (Automatic Teller Machines) em aplicações do banco 24 horas. Neste caso, o usuário deve informar a senha diretamente na tela. Isto evita a necessidade de se ter usuários registrados individualmente no sistema operacional.

Observe que de maneira mais geral, quando um usuário se "conecta" em uma OLTA, ele entra em um subsistema de serviços limitados, onde não existe a possibilidade de acesso aos serviços gerais do sistema operacional, como por exemplo linhas de entrada de comandos, linguagens de programação, e outros. Os usuários ficam limitados às operações fornecidas pela aplicação OLTA.

11.2.4. Os Componentes de Sistemas OLTPs

Os sistemas OLTPs fornecem vários componentes para simplificar o desenvolvimento de OLTAs. Os principais destes componentes são:

- *Usuário de Processamento de Transações (usuário TP)*: consiste de uma pessoa em um terminal, em uma estação de trabalho, ou em um microcomputador, que submete transações para execução através de uma OLTA.
- *Mapa (ou "form")*: é a interface na tela que fica entre um usuário TP e uma OLTA.
- *Menu*: um mapa especial usado para selecionar as opções iniciais em uma OLTA.
- *Rotina Servidora*: uma rotina que executa um serviço requisitado por um usuário através de um programa cliente. Por exemplo, a leitura de um registro ou a atualização de uma base de dados.
- *Programa Cliente*: um programa que recebe dos usuários pedidos por serviços e os envia para as rotinas servidoras. Um programa cliente é por vezes chamado de *TCP (Transaction Control Program)*.
- *Transação*: uma unidade atômica de execução, representada por programas clientes.
- *Configuração da aplicação*: uma especificação, escrita em uma linguagem de alto nível, ou especificada interativamente utilizando um utilitário, que define o ambiente no qual as transações serão executadas. Exemplos de informações contidas nesta especificação incluem: o número máximo de usuários, o número máximo de transações simultâneas, esquemas de bancos de dados, níveis de segurança, etc.
- *Utilitário para especificação da configuração*: uma linguagem ou uma interface interativa para se especificar o ambiente OLTA.
- *Sistema de gerenciamento de mapas (ou "forms")*: uma facilidade que pode ser usada estaticamente ou interativamente para desenhar e compilar telas de uma interface OLTA.
- *Monitor de processamento de transações*: um programa que monitora a execução das transações em uma OLTA.

Depois de montada toda a estrutura necessária, incluindo-se a configuração de inicialização, uma OLTA pode então ser disponibilizada para os usuários após a emissão de um comando de operador (na console do sistema) do tipo "iniciar OLTA". Este comando lê o arquivo de configuração e inicializa os processos clientes e servidores, executando as conexões necessárias com o monitor de processamento de transações. Os usuários podem então inicializar suas transações a partir de uma tela de menu. Um programa cliente executa sempre a seguinte seqüência de funções, independentemente de onde ele esteja:

1. Recebe uma mensagem de entrada;
2. Invoca uma rotina servidora e a direciona para executar ações sobre um determinado objeto (por exemplo, sobre um banco de dados);
3. Envia uma mensagem de saída para a tela do usuário como resposta da rotina servidora.

Em resumo, a seqüência será sempre do tipo: recebe <msg> de entrada; processa o pedido; envia <msg> de saída.

No caso (b), uma OLTA assume o papel de dona do processo, e os usuários são autenticados individualmente pelo próprio sistema operacional. Assim, um usuário pode se conectar normalmente ao terminal, e em seguida requisitar uma aplicação OLTA. O processo de usuário fica separado do processo da OLTA.

Em (c), a OLTA autentica os usuários, como acontece em terminais de automação bancária do tipo ATMs (Automatic Teller Machines) em aplicações do banco 24 horas. Neste caso, o usuário deve informar a senha diretamente na tela. Isto evita a necessidade de se ter usuários registrados individualmente no sistema operacional.

Observe que de maneira mais geral, quando um usuário se "conecta" em uma OLTA, ele entra em um subsistema de serviços limitados, onde não existe a possibilidade de acesso aos serviços gerais do sistema operacional, como por exemplo linhas de entrada de comandos, linguagens de programação, e outros. Os usuários ficam limitados às operações fornecidas pela aplicação OLTA.

11.2.4. Os Componentes de Sistemas OLTPs

Os sistemas OLTPs fornecem vários componentes para simplificar o desenvolvimento de OLTAs. Os principais destes componentes são:

- *Usuário de Processamento de Transações (usuário TP)*: consiste de uma pessoa em um terminal, em uma estação de trabalho, ou em um microcomputador, que submete transações para execução através de uma OLTA.
- *Mapa (ou "form")*: é a interface na tela que fica entre um usuário TP e uma OLTA.
- *Menu*: um mapa especial usado para selecionar as opções iniciais em uma OLTA.
- *Rotina Servidora*: uma rotina que executa um serviço requisitado por um usuário através de um programa cliente. Por exemplo, a leitura de um registro ou a atualização de uma base de dados.
- *Programa Cliente*: um programa que recebe dos usuários pedidos por serviços e os envia para as rotinas servidoras. Um programa cliente é por vezes chamado de *TCP (Transaction Control Program).*
- *Transação*: uma unidade atômica de execução, representada por programas clientes.
- *Configuração da aplicação*: uma especificação, escrita em uma linguagem de alto nível, ou especificada interativamente utilizando um utilitário, que define o ambiente no qual as transações serão executadas. Exemplos de informações contidas nesta especificação incluem: o número máximo de usuários, o número máximo de transações simultâneas, esquemas de bancos de dados, níveis de segurança, etc.
- *Utilitário para especificação da configuração*: uma linguagem ou uma interface interativa para se especificar o ambiente OLTA.
- *Sistema de gerenciamento de mapas (ou "forms")*: uma facilidade que pode ser usada estaticamente ou interativamente para desenhar e compilar telas de uma interface OLTA.
- *Monitor de processamento de transações*: um programa que monitora a execução das transações em uma OLTA.

Depois de montada toda a estrutura necessária, incluindo-se a configuração de inicialização, uma OLTA pode então ser disponibilizada para os usuários após a emissão de um comando de operador (na console do sistema) do tipo "iniciar OLTA". Este comando lê o arquivo de configuração e inicializa os processos clientes e servidores, executando as conexões necessárias com o monitor de processamento de transações. Os usuários podem então inicializar suas transações a partir de uma tela de menu. Um programa cliente executa sempre a seguinte seqüência de funções, independentemente de onde ele esteja:

1. Recebe uma mensagem de entrada;
2. Invoca uma rotina servidora e a direciona para executar ações sobre um determinado objeto (por exemplo, sobre um banco de dados);
3. Envia uma mensagem de saída para a tela do usuário como resposta da rotina servidora.

Em resumo, a seqüência será sempre do tipo: recebe <msg> de entrada; processa o pedido; envia <msg> de saída.

11.2.5. Transações Conversacionais e Não Conversacionais

Uma transação conversacional [Cla92b] envolve possívelmente uma série de mensagens sendo trocadas entre um usuário e uma aplicação OLTA dentro do escopo de uma única transação.

Uma transação não conversacional (ou pseudo conversacional) envolve uma mensagem entre um usuário e um programa cliente que contenha dados a serem usados no processamento do tipo de transação selecionada. A mensagem neste caso conterá de maneira objetiva a operação desejada: por exemplo, a alteração ou a remoção de um registro em uma base de dados. No entanto, uma transação não conversacional pode envolver a passagem de várias mensagens entre os processos clientes e servidores antes que aconteça o commit da transação.

Em ambos os casos, é importante que se diga que, quanto menor o número de mensagens trocadas entre os processos clientes e servidores, menor o *"overhead"* gerado na execução de uma transação.

11.2.6. Sistemas de Gerenciamento de Mapas

A leitura e a gravação de mapas (formulários - *"forms"* - para apresentação em telas de terminais) são tratadas por **sistemas de gerenciamento de mapas (SGMs)** [Cla92b]. Um SGM fornece um conjunto de ferramentas tanto na forma de telas interativas quanto na forma de editores em lote para ajudar no desenvolvimento de aplicações OLTAs. Dentre as principais funcionalidades de um SGM podemos incluir: permitir a criação de mapas e menus; tratar o recebimento e o envio de mapas para telas de terminais; passar adiante os dados recebidos para um processo cliente. Para que os mapas possam ser usados é necessário em primeiro lugar que os mesmos sejam criados. O processo de criação destes mapas envolve a definição de número de linha, número de coluna, nome e tamanho de cada campo que deva constar dos mesmos. Adicionalmente, podem-se estabelecer para estes campos valores "default" e mecanismos especiais como atributos sublinhados, piscantes ou escondidos.

Um SGM constitui a interface entre um programa cliente e um usuário. Os dados são passados do SGM para um processo cliente e deste de volta para o SGM, como mostrado na figura 11.5. Idealmente, um SGM deveria executar também validações de dados, porém o mais comum é que estas validações fiquem a cargo do programa cliente.

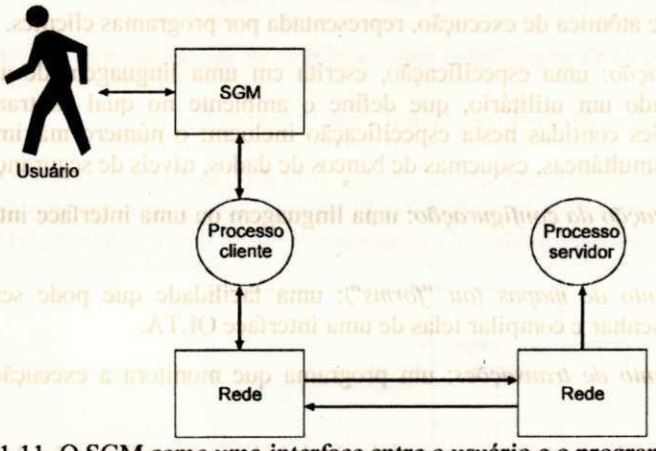


Figura 11.11. O SGM como uma interface entre o usuário e o programa cliente

11.2.7. O Monitor de Processamento de Transações

Como vimos há pouco, o ***Monitor de Processamento de Transações*** **(MPT)** [Cla92b] cuida de monitorar a execução das transações em uma OLTA. A ligação ("linkedição") do MPT com programas clientes e servidores pode ser implementada de duas maneiras. Na primeira, empregada em sistemas OLTPs convencionais de ambientes centralizados, o MPT, os programas clientes, e os programas servidores são ligados em um único módulo executável. Na segunda, o MPT, os programas clientes, e os programas servidores são ligados em programas executáveis separados que podem comunicar-se entre si por meio de troca de mensagens. Este último caso é perfeitamente adequado para os ambientes distribuídos, onde os processos para o MPT, e para os programas clientes e servidores podem ser arranjados em uma árvore de processamento tendo o MPT como sua raiz. A figura 11.5 ilustra o relacionamento entre usuários TP e estes processos.

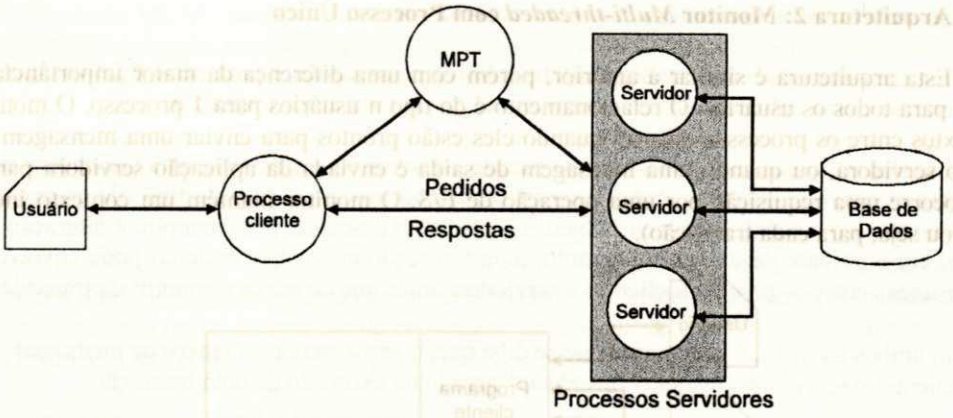


Figura 11.5. Relacionamento entre os componentes OLTP [Cla92b]

Um processo cliente envia menus e mapas ("forms") para a telas e recebe dados destas telas. Normalmente, os processos servidores são *"multi-threaded"* para permitir o seu uso simultâneo por muitos processos clientes.

O monitor de processamento de transações monitora e controla todos os processos e recursos em um sistema OLTP. Na figura 11.5 anterior, um fluxo de pedidos e respostas por serviços é mostrado diretamente entre os processos clientes e servidores. Este fluxo também fica sob o controle do MPT. Um processo cliente não pode enviar um pedido a um processo servidor nem um processo servidor pode enviar uma resposta a um processo cliente sem que o monitor deixe de tomar conhecimento deste fato. Os processos remotos são tratados da mesma maneira.

O MPT examina os pedidos vindos de processos clientes e determina que rotinas servidoras serão executadas. Se todos os processos servidores em um grupo estiverem ocupados, o monitor cuida de enfileirar estes pedidos. Quando um processo servidor torna-se disponível o pedido no topo da fila é selecionado.

11.2.7.1. Arquiteturas para Monitores de Processamento de Transações

Apresentamos a seguir três arquiteturas para monitores de processamento de transações.

Arquitetura 1: Monitor Mono-Processador com Processo Único

Nesta arquitetura é implementado um relacionamento do tipo 1 processo por cada usuário. Os programas clientes, os programas servidores, e o próprio monitor juntamente com a especificação da configuração são ligados em um único módulo executável. Este módulo é executado através de um processo individual para cada usuário.

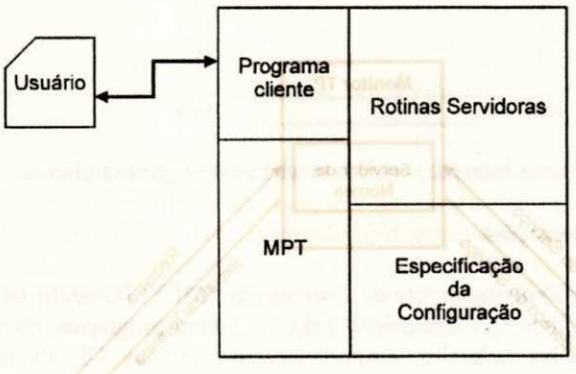


Figura 11.6. Monitor mono-processador com processo único [Cla92b]

Poucos sistemas OLTP foram desenvolvidos para esta arquitetura apesar de sua simplicidade. As desvantagens são enormes devido a fatores como um desempenho pobre, consumo excessivo de memória, e "overhead" adicional para tratar cada processo.

Arquitetura 2: Monitor *Multi-threaded* com Processo Único

Esta arquitetura é similar à anterior, porém com uma diferença da maior importância: existe um único processo para todos os usuários. O relacionamento é do tipo n usuários para 1 processo. O monitor cuida da troca de contextos entre os processos clientes quando eles estão prontos para enviar uma mensagem de entrada para a aplicação servidora, ou quando uma mensagem de saída é enviada da aplicação servidora para a tela, ou ainda quando ocorre uma requisição por uma operação de E/S. O monitor mantém um contexto individual para cada usuário (ou seja, para cada transação).

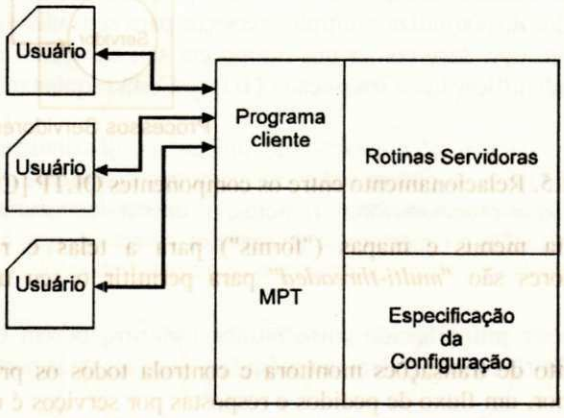


Figura 11.7. Monitor *"multi-threaded"* com processo único [Cla92b]

Este modelo é o mais comumente empregado pelos sistemas OLTPs comerciais disponíveis para as plataformas de mainframes. A troca de contextos nesta situação é menos complexa do que a troca de contextos de processos a nível do sistema operacional. O monitor mantém uma tabela com todas as rotinas servidoras e invoca a rotina servidora apropriada dependendo do tipo de pedido recebido na mensagem de entrada enviada por um usuário através da aplicação cliente.

A vantagem óbvia desta arquitetura é que existem poucos processos a nível do sistema operacional, o que implica em poucas trocas de contextos, com a conseqüente diminuição do consumo de memória. A principal desvantagem fica por conta de um código fonte muito grande e complexo para o monitor, o que implica em maiores possibilidade de surgirem "bugs".

Arquitetura 3: Monitor OLTP em Arquitetura Cliente/Servidor

Nesta arquitetura o próprio monitor cuida de implementar a comunicação entre processos clientes e servidores. Esta opção é sem sombra de dúvidas a mais adequada para a implementação de sistemas OLTPs em ambientes da computação distribuída. O relacionamento é do tipo n usuários para n processos, como mostrado na figura 11.8.

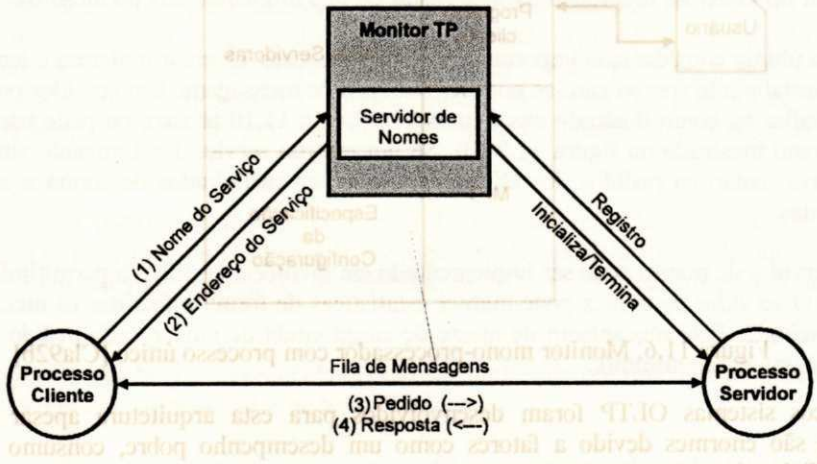


Figura 11.8. Monitor OLTP em arquitetura cliente/servidor [Cla92b]

Nesta arquitetura o servidor de nomes é um componente básico que cuida de atender aos pedidos por serviços de nomes enviados pelos processos clientes. O servidor de nomes determina a localização dos servidores mais adequados para atender a cada um destes pedidos. Os clientes utilizam a informação sobre esta localização para direcionarem seus pedidos para estes servidores. Como mostrado na figura 11.8 para cada serviço requisitado por um cliente, o fluxo de controle ocorre na seguinte ordem: (1) o cliente requisita o servidor de nomes; (2) o endereço do servidor é fornecido; (3) o cliente requisita o serviço ao servidor indicado; (4) o servidor responde ao pedido do cliente.

Um servidor pode suportar muitos serviços, e um serviço pode ser fornecido para vários clientes simultaneamente. Quando um servidor disponibiliza múltiplos serviços para seus clientes, uma mensagem enviada de um cliente para um servidor deste tipo deve conter um campo em seu cabeçalho que indique qual o tipo de serviço desejado, juntamente com a identificação da transação (TID) que está requisitando o serviço.

Obviamente, como em qualquer implementação de aplicações distribuídas, os processos clientes e servidores podem rodar na mesma máquina ou podem rodar em máquinas totalmente distintas (diz-se neste último caso que os processos são remotos). Se os processos forem remotos, os clientes conectam-se às máquinas servidoras por meio de um protocolo de comunicações e conectam-se ao servidor correto graças à informação fornecida pelo servidor de nomes.

Um processo cliente pode ser multiplexado entre muitos usuários, porém esta situação só é possível quando estes processos clientes *"multi-threaded"* rodam em máquinas capazes de suportar centenas ou milhares de terminais "mudos".

Com relação aos processos servidores em sistemas OLTP podemos considerar duas situações: na primeira, o servidor fica inabilitado para outros processamentos enquanto não terminar de atender ao pedido corrente, diz-se neste caso que o servidor é de *"thread-única"*. Na segunda situação, enquando uma thread estiver aguardando que uma operação de E/S seja completada para satisfazer ao pedido que está sendo tratado correntemente, outra thread pode estar processando outro pedido. Diz-se neste caso que o servidor édo tipo *"multi-threaded"*. Esta situação é mais comum, e bem mais eficiente, que a primeira.

Em servidores *"multi-threaded"*, se todas as threads estiverem ocupadas em um detrminado momento, então um processo servidor deste tipo pode ficar habilitado (em algumas implementações) para disponibilizar uma nova thread sempre que surgir um novo pedido, e pode em seguida destruir esta thread quando o pedido tiver sido atendido. Esta característica é particularmente interessante em muitas situações encontradas em aplicações OLTPs, e é chamada de "ajuste dinâmico de carga" ("Dynamic Load Balancing"). Por exemplo, o uso de terminais de automação bancária aumenta de maneira previsível em certas horas do dia. Pode-se portanto inicializar novos servidores nestes momentos de pico para que se evite uma saturação no sistema e o tempo de resposta não se degrade. Em outros casos, onde não se pode prever o uso do sistema, pode-se permitir que novos servidores sejam inicializados automaticamente quando certos indicadores chegarem a níveis predeterminados (isto é, atingirem um limite intolerável, ou seja, atingirem um "threshold" predeterminado). Estes servidores podem ser destruídos quando a carga for novamente restabelecida para um nível tolerável. A criação e destruição de processos servidores adicionais de maneira dinâmica é função inerente ao próprio monitor de processamento de transações (e certamente irá depender de esta funcionalidade estar ou não implementada no mesmo).

Uma última consideração importante para os monitores TP em arquitetura cliente/servidor é que clientes e servidores normalmente comunicam-se por meio de filas de mensagens. Um servidor pode ter a sua própria fila de mensagens dedicada, como ilustrado nas figuras 11.9 (a) e 11.10 abaixo, ou pode partilhar uma fila com outros servidores, como mostrado na figura 11.9 (b). Se um mesmo serviço for fornecido simultaneamente por dois ou mais servidores, então os pedidos dos clientes poderão ser enfileirados de forma a serem atendidos pelas filas menos saturadas.

O servidor de nomes pode ser implementado em memória partilhada permitindo acesso a todos os clientes e servidores. O servidor de nomes pode manter estatísticas de forma a facilitar os mecanismos de ajuste de carga ("Load Balancing"). Este mecanismo de ajuste de carga cuida de enviar cada pedido para o servidor que puder atendê-lo com um atraso mínimo.

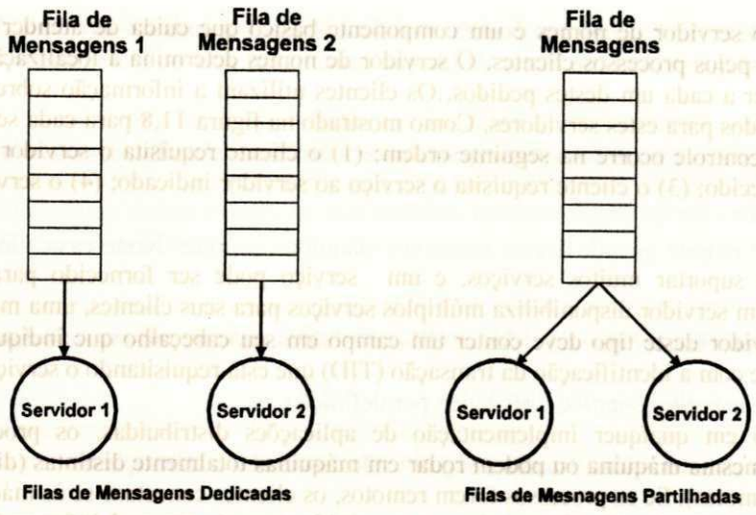


Figura 11.9. Filas de Mensagens e configurações de servidores [Cla92b]

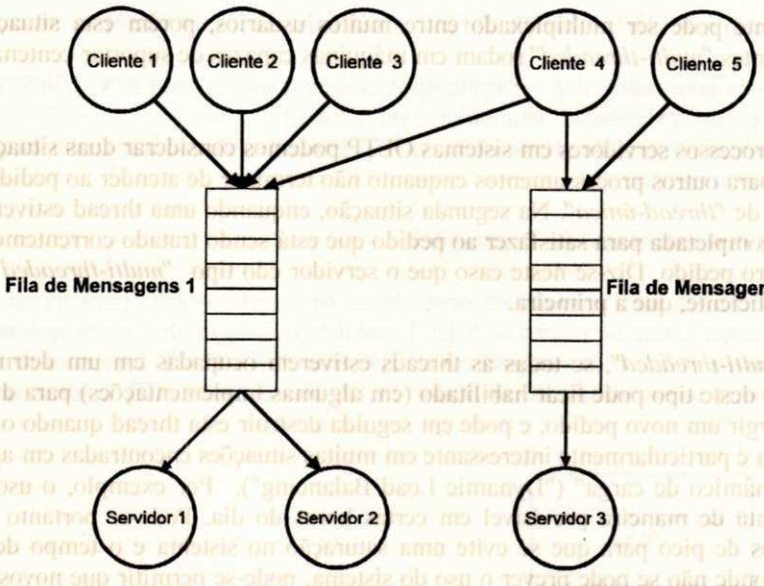


Figura 11.10 . Clientes, servidores e filas de mensagens [Cla92b]

11.3. Os Modelos de Processamento de Transações

Vários modelos foram desenhados para descrever a arquitetura dos sistemas TP [Cla92b]. Examinamos aqui três dos mais recentes destes modelos, os quais chamaremos de modelo A, modelo B e modelo C. Cada um deles é caracterizado pela natureza das interfaces entre seus componentes. Estes modelos podem tratar tanto de transações com pontos de commit únicos (transações simples) quanto de transações com múltiplos pontos de commits (transações multi-commit). Antes de continuarmos, no entanto, algumas definições adicionais sobre transações são necessárias.

Um *Gerente de Recursos* **(GR)** é o módulo do sistema TP responsável pela gerência dos recursos que estiverem sob seu controle. Os GRs cuidam de garantir os aspectos de consistência e de integridade destes recursos. Exemplos de GRs são os sistemas de arquivos distribuídos, SGBDs, sistemas de automação de escritórios, etc.

Uma *transação simples* é aquela que é executada dentro dos limites de um único gerente de recursos (por exemplo, dentro dos limites de um único gerenciador de bancos de dados).

Uma *transação com múltiplos commits (ou transação multi-commit)* é aquela onde dois ou mais GRs são chamados para atender aos serviços requisitados pela transação.

Transações distribuídas são aquelas que podem modificar dados que existem em mais de um nó de um banco de dados distribuído, ou que podem modificar dados que existam em dois ou mais bancos de dados em um mesmo nó da rede.

As transações simples e as transações *multi-commit* podem ainda ser classificadas como *transações locais ou remotas* [She93a]:

- Uma transação simples é remota quando acessa dados em máquinas remotas. Neste caso, ela é também uma transação distribuída.
- Uma transação *multi-commit* é local quando todos os GRs envolvidos estão na mesma máquina.
- Observe que uma transação *multi-commit* é distribuída por definição.

De maneira geral, as características listadas a seguir estão associadas a transações:

1. o identificador de transações (TID) não é visível para os programas de aplicações;
2. um processo (ou uma thread) pode ser associado com no máximo uma transação (TID) em determinado momento;
3. um TID pode ser associado a múltiplas threads durante a sua existência (por exemplo, no caso de transações distribuídas);
4. se uma thread se comunica com outra, por exemplo, através de um mecanismo RPC [Blo92], então a thread "chamada" irá tornar-se parte da transação "chamadora" por default;
5. qualquer thread pode chamar a rotina de aborto-de-tx, porém somente a thread que tenha iniciado a transação pode executar o commit da transação.

11.3.1. O Modelo "A" para Processamento de Transações

Neste modelo foi deenhada uma *unidade de controle de transações (UCT)* para fornecer o controle de transações para uma determinada classe de recursos. A UCT consiste de três módulos: um *gerente de transações (GT)*, de um *gerente de commits (GC)*, e de um *gerente de recursos (GR)*. O GT comunica-se tanto com o GR quanto com o GC diretamente, como mostrado na figura 11.12. Neste caso o GR assume uma função mais especializada.

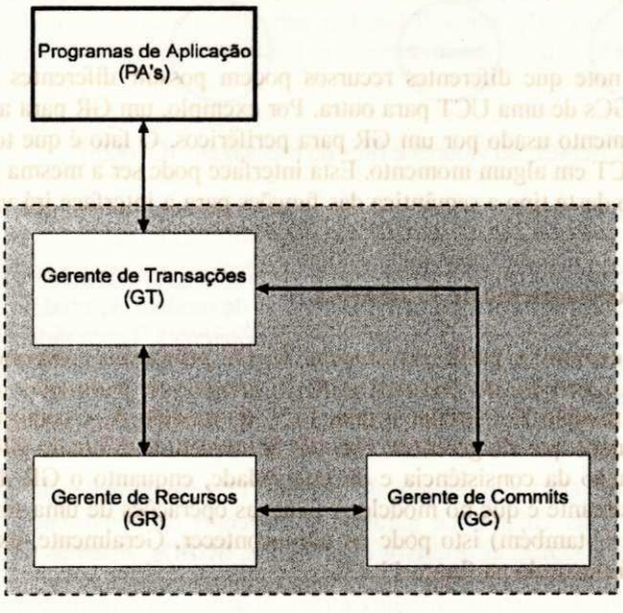


Figura 11.12. Arquitetura proposta para uma UCT [Cla92b]

Cada uma das classes de recursos neste modelo é gerenciada por um GR em particular. Os GRs cuidam da *segurança* e da *consistência* destes recursos, porém não cuidam das questões relativas à *integridade* dos mesmos. Basicamente, as UCT cuidam de aspectos como:

- A *segurança,* que consiste do fornecimento de dois tipos de serviços: permitir acessos a recursos ou proibir estes acessos.

- A *consistência, que* é mantida pelo módulo escalonador, o qual veremos com mais detalhes adiante. A consistência dos recursos deve ser garantida sempre que duas ou mais transações tentam acessar os mesmos recursos simultaneamente. Um escalonador assegura que o efeito das atualizações concorrentes forneçam os mesmos resultados que seriam obtidos se as transações fossem executadas em ordem serial (ou seja, ele cuida da serialização das transações).

- A *integridade*, por sua vez, irá assegurar que os dados armazenados em um arquivo ou banco de dados estejam sempre corretos, mesmo que ocorram falhas no sistema. O GC é o responsável por assegurar a integridade dos dados.

Um GR típico em um sistema TP é o gerenciador de arquivos. A funcionalidade específica de um GR irá depender das características da classe de recursos que ele gerencia.

O GC é o responsável pela manutenção da integridade dos recursos gerenciados pelo GR. Um GC controla todas as funções de commit, "rollback" para recuperação, ou abortos. Adicionamente, o GC implementa uma operação de "preparação para o commit", que é emitida pelo GT, e que faz parte do *protocolo de commit de duas fases ("two phase commit" ou protocolo 2PC)* [OV91] usado no processamento de transações distribuídas.

O protocolo 2PC é um protocolo de votação em duas fases que assegura um commit atômico quando existe mais de um participante envolvido no commit de uma transação. A primeira fase é uma fase de votação. A segunda fase é a fase de ação: ou commit ou aborta. Existe sempre uma transação coordenadora, e uma ou mais transações subordinadas envolvidas. A transação coordenadora pede a todas as transações subordinadas os seus votos em termos de elas estarem ou não prontas para efetivarem o commit. Se todas as transações subordinadas votarem "SIM", então a transação coordenadora irá instruir todas as transações subordinadas a executarem o commit. Se uma ou mais das transações subordinadas votarem "NÃO", então a transação coordenadora irá instruir todas elas a abortarem.

Um GT tipicamente mantém o estado de cada transação, como por exemplo, se ela foi commited, abortada, ou se continua ativa. Quando o GT recebe uma ação do tipo inicio-de-tx ele cria um *identificador de transação* único para esta transação (TID) e aloca um *descritor de transação* para esta transação. Um GT cuida adicionalmente de várias outras funções, incluindo a detecção e resolução de conflitos ("deadlocks") entre transações concorrentes.

É interessante que se note que diferentes recursos podem possuir diferentes características. Este fato implica na variação dos GRs e GCs de uma UCT para outra. Por exemplo, um GR para arquivos pode não utilizar o mesmo algoritmo de escalonamento usado por um GR para periféricos. O fato é que todas as transações devem utilizar uma das interfaces da UCT em algum momento. Esta interface pode ser a mesma para todos os recursos no sistema, porém em uma situação deste tipo a semântica das funções para a interface irá variar de uma função para outra.

11.3.2. O Modelo "B" para Processamento de Transações

O modelo B evoluiu a partir do modelo A. Os principais componentes do modelo são os *programas de aplicação (PA)*, o *gerente de recursos (GR)*, o *gerente de transações (GT)*, e o *subsistema de comunicações (SSC)*. O GR do modelo B é similar a uma UCT do modelo A, e como tal cuida de assegurar as propriedades ACID para os recursos que ele gerencia. Um GR do modelo B difere do GR do modelo A por ser de sua responsabilidade a manutenção da consistência e da integridade, enquanto o GR do modelo A só cuida da consistência. Outra diferença marcante é que no modelo A todas as operações de uma transação passam pelo GT, enquanto que nos modelos B (e C também) isto pode ou não acontecer. Geralmente, existem múltiplos GRs em uma instalação. O modelo B está ilustrado na figura 11.13 .

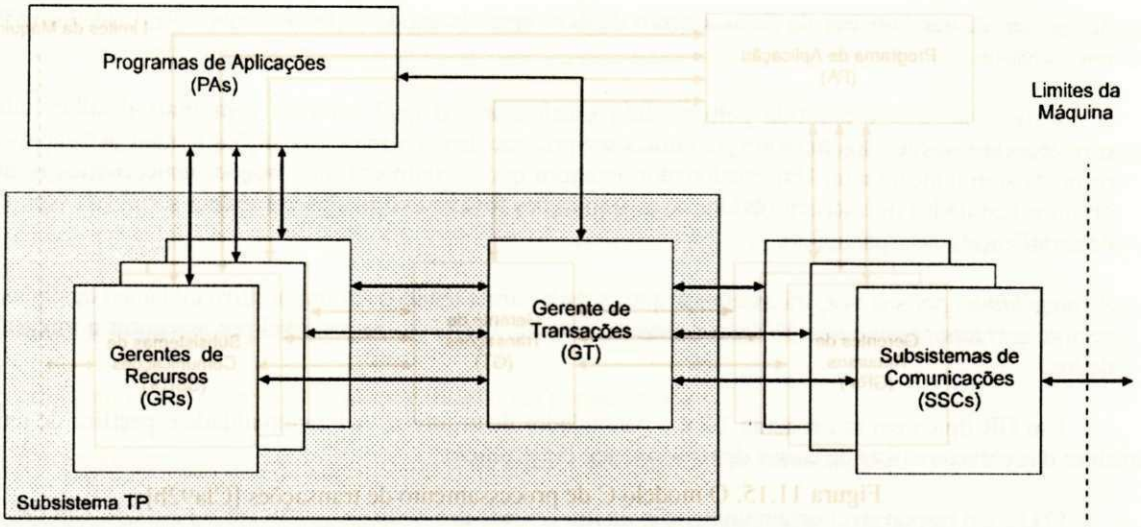


Figura 11.13. O Modelo "B" para Processamento de Transações [Cla92b]

O SSC implementa um conjunto de serviços de comunicações. Os PAs utilizam este conjunto de serviços de comunicações para comunicarem-se com outros PAs. A figura 11.14 mostra a interação de uma aplicação que consiste de clientes e servidores, com outros componentes de um sistema TP distribuído.

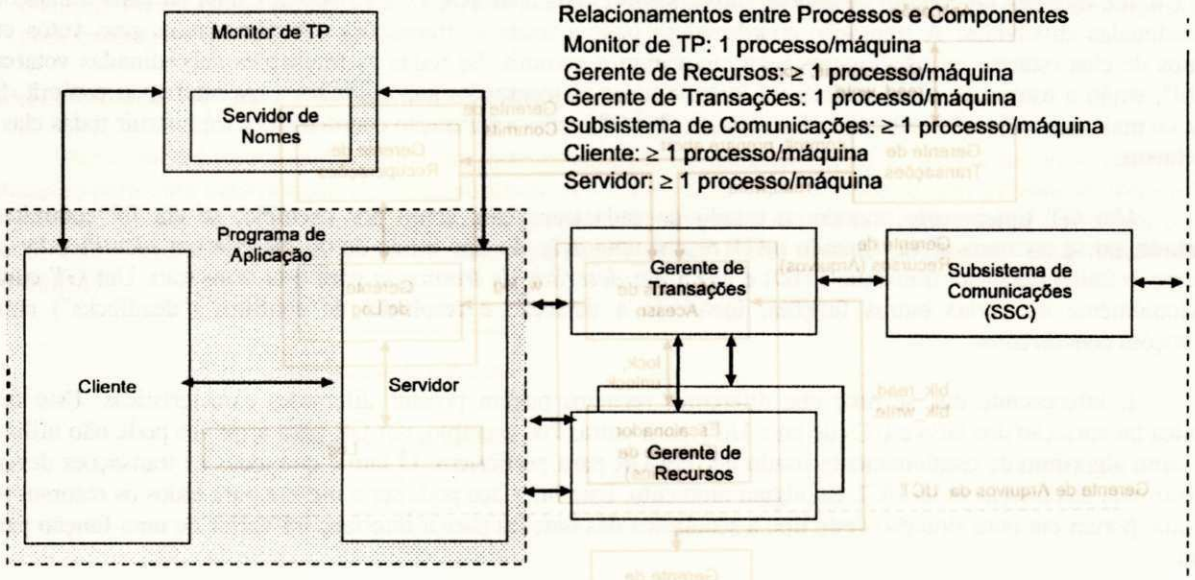


Figura 11.14. Um modelo de sistema TP distribuído [Cla92b]

Neste modelo, o GT trata os commits de duas fases para transações *multi-commit* e utiliza o SSC para se comunicar com outros nós na rede. Os serviços cliente/servidor permitem que um processo cliente transmita um pedido para um processo servidor, e que o servidor retorne uma resposta para este cliente. O relacionamento entre pedidos e respostas é sempre do tipo um-para-um. Este relacionamento pode ser modelado e implementado pelo uso de mecanismos de intercomunicação entre processos como RPCs.

11.3.3. O Modelo "C" de Processamento de Transações

Nosso último modelo ilustra um caso onde o SSC atua de maneira similar a um GR. Este modelo é motivado pelo interesse em se ter múltiplos SSCs em um sistema sem que haja a necessidade de ocorrerem mudanças no GT, e também para permitir a migração facilitada de programas de aplicações. Isto é, os PAs escritos baseados em serviços de comunicações específicos podem ser migrados facilmente para este modelo, sem que mudanças radicais nas interfaces de comunicações sejam necessárias (desde, é claro, que estas interfaces sejam suportadas pelo SSC utilizado).

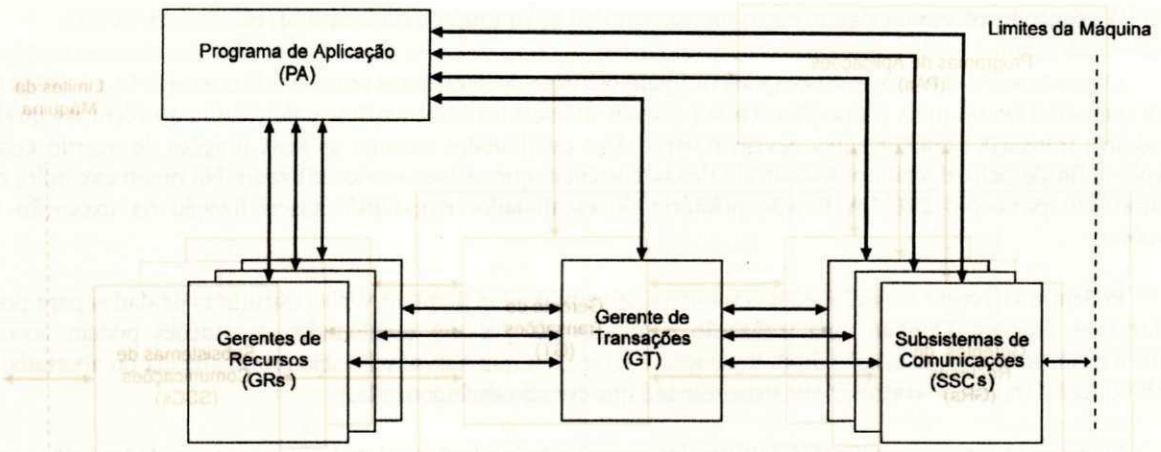


Figura 11.15. O modelo C de processamento de transações [Cla92b]

11.3.4. Características das Unidades de Controle de Transações (UCTs) e Gerentes de Recursos (GRs)

Apresentamos nesta seção o exemplo de um sistema de arquivos para ilustrar as funcionalidades e características exigidas de uma **UCT/GR.**

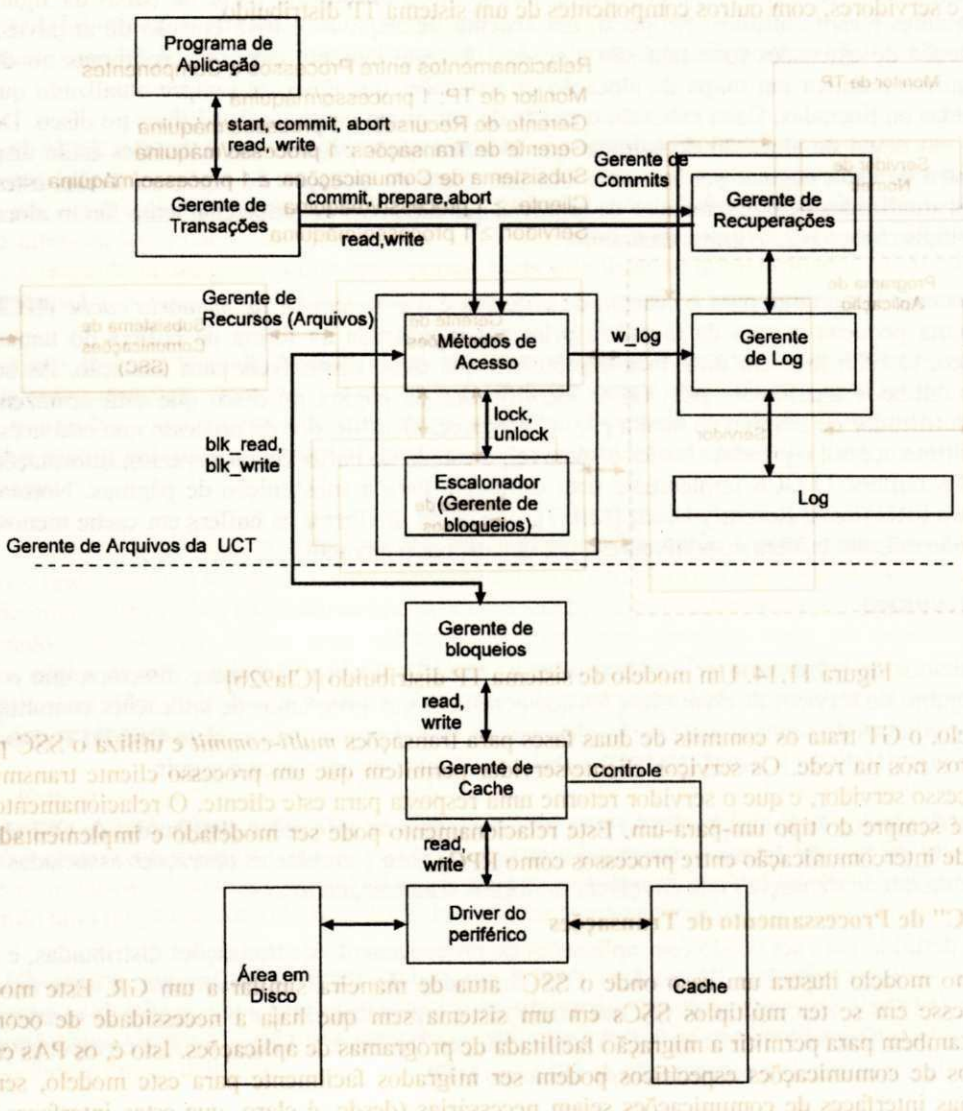


Figura 11.16. Uma UCT para gerenciamento de arquivos [Cla92b]

A seguir descrevemos alguns dos componentes da UCT mostrada na figura 11.16:

O *escalonador* é o programa responsável pelo controle de acesso aos recursos. Ele resolve os conflitos que surgem quando duas ou mais transações tentam acessar as mesmas informações, e determina a ordem na qual as ações destas transações concorrentes devem ocorrer. Um escalonador executa as suas funções de acordo com o protocolo definido pelo algoritmo de controle de concorrência que estiver sendo utilizado. No nosso exemplo, este algoritmo é o protocolo 2PC. A função primária do escalonador é garantir a serialização na execução das transações.

Podemos enxergar o escalonador como uma caixa preta que mantém várias estruturas de dados para poder executar suas funções. Quando uma transação é selecionada por um escalonador, três ações podem ocorrer: permitir a ação; recusar a execução desta transação e fazer com que a transação seja reinicializada ou abortada; ou tentar bloquear a transação e atrasar sua execução até que ela seja desbloqueada.

O *gerenciador de commits* (**GC**) utiliza informações redundantes coletadas enquanto os dados estão sendo atualizados, de tal forma que se consiga efetuar a recuperação caso ocorram falhas. Por exemplo, arquivos de *log* ou jornais ("journals") podem ser usados para coletar as informações redundantes necessárias, de tal forma que o componente "gerenciador de logs" dentro do GC possa recuperar o valor de um registro que estiver sendo atualizado em determinado arquivo, e possa então gravá-lo em um arquivo de *log* com imagens já atualizadas ("after images").

Para o gerenciador de arquivos propriamente dito, como mostrado na parte de baixo da figura 11.16 anterior, muitos detalhes foram omitidos. No geral, um sistema de arquivos é uma coleção de arquivos, e cada arquivo é uma coleção de extensões (que não são alocadas obrigatoriamente em áreas contínuas no disco). O gerenciador de arquivos mantém um mapa de alocação de extensões que deve ser sempre atualizado quando as extensões são alocadas ou liberadas. Cada extensão consiste de um grupo contínuo de páginas no disco. Dentro de cada extensão fica um mapa de alocação de páginas que indicam quais páginas nas extensões estão disponíveis para alocação. Dentro de cada página, por sua vez, fica um mapa de alocação de registros. Todos estes mapas devem ser mantidos atualizados pelo gerenciador de arquivos quando áreas de armazenamento forem alocadas, ou quando for exigido mais espaço para o armazenamento.

Um componente importante para o sistema de arquivos é o *gerenciador de memória cache* (**GCh**). Uma memória cache é uma pequena porção da memória principal organizada na forma de buffers do tamanho das páginas de um disco. O GCh mantém uma lista dos buffers que estão disponíveis para alocação. As seguintes informações são mantidas e atualizadas pelo GCh: identificador da página no disco que está armazenada em memória cache; identificador do arquivo ao qual a página pertence; identificador do processo que está acessando a página; a hora do último acesso; o próximo buffer disponível; o estado do buffer (em uso/vazio); informações sobre indisponibilidade dos buffers. O GCh implementa uma estratégia para a substituição de páginas. Normalmente, utiliza-se o algoritmo LRU (Least Recently Used) [Mil87], que cuida de liberar os buffers em cache menos usados recentemente caso não existam buffers disponíveis em um determinado momento.

11.3.5. O Modelo DTP/OSI

Quando falamos do subsistema de comunicações no modelo B visto há pouco, dissemos que o mesmo implementa um conjunto de serviços de comunicações que permitem que programas de aplicações comuniquem-se entre si. Um destes conjuntos de serviços pode ser implementado, por exemplo, pelo modelo **OSI/DTP** *(Distributed Transaction Processing)* ([Hal93b] [Cas93] [HM91]), sobre o qual falamos um pouco a seguir.

O elemento de serviço de aplicação (ESA) para processamento de aplicações distribuídas da OSI chama-se DTP, o qual está ainda em fase de desenvolvimento, e será voltado para controlar as operações associadas com os sistemas de processamento de transações que envolvem múltiplos sistemas abertos.

O DTP foi definido para ser usado com aplicações de processamento de transações distribuídas, e faz uso de um conjunto de serviços especiais do modelo OSI chamados de **CCR** (*Commitment, Concurrency and Recovery*) [Cla92b]. O CCR é uma espécie de GC que implementa o protocolo 2PC e o esquema de recuperação de imagens anteriores às atualizações ("before images") conhecido por "rollback". Assim, o CCR é responsável pelas questões de controle de concorrência e recuperação de falhas no DTP.

O DTP fornece serviços para controlar a troca de informações entre os programas de aplicações (PAs) que compõem as transações distribuídas, de acordo com as propriedades ACID já referenciadas no início deste capítulo. Na figura 11.17 descrevemos um ambiente típico para uma aplicação DTP.

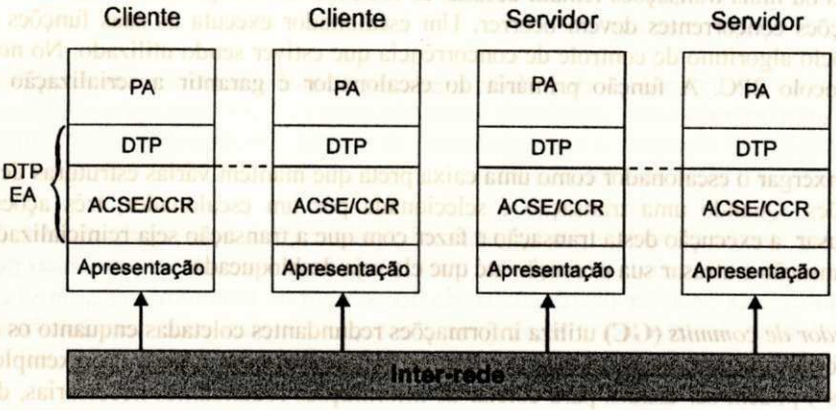


Figura 11.17. Esquema de uma aplicação DTP [Hal93b]

O DTP utiliza os serviços dos *elementos de serviços de aplicação* (ESAs) conhecidos por **CCR** e **ACSE** (*Association Control Service Element*) do modelo OSI . Como já explicamos o CCR, falamos agora sobre o ACSE.

O ACSE é quem inicializa e libera as associações entre duas aplicações. Uma *associação* é uma conexão lógica entre duas aplicações. Assim, a comunicação entre dois programas de aplicação de usuários é fornecida por meio de um canal lógico de comunicações que é estabelecido entre as duas aplicações antes que elas comecem a troca de informações. Pode-se também utilizar um mecanismo mais simples para esta comunicação, como por exemplo a troca de mensagens, o que seria porém mais difícil de se implementar. É o ACSE que cuida destes procedimentos.

Cada PA cliente pode desejar inicializar uma transação que envolva múltiplos servidores, e portanto, múltiplas sub-transações. O objetivo do serviço DTP é fornecer a cada cliente serviços que permitam que ele execute transações concorrentemente com outros clientes. Para cada transação distribuída, os PAs de usuários envolvidos são organizados em uma estrutura em árvore, com o cliente que inicializa a transação inicial na raiz desta árvore.

O PA raiz, conhecido como mestre (ou superior), irá então iniciar uma transação com múltiplos servidores, onde cada um destes servidores são conhecidos como escravos (ou subordinados). Adicionalmente, em aplicações mais sofisticadas, enquanto as operações associadas com uma transação estiverem sendo executadas, um subordinado pode desejar iniciar uma sub-transação. Um bom exemplo pode ser mostrado quando existem múltiplas cópias de um arquivo controlado por determinado servidor. Neste caso, antes de responder a um pedido de commit, o servidor pode querer iniciar a atualização de cada cópia deste arquivo, criando assim o que se chama de uma *árvore de transações*.

Existem vários níveis de complexidade associados com os sistemas de processamento de transações. Em alguns casos o acesso a um único recurso pode precisar ser controlado. Em outros caso onde existam múltiplas transações concorrentes, vários recursos precisam ser controlados por meio de árvores de transações. Portanto, os serviços de usuários associados com o DTP são divididos em várias unidades funcionais (UFs), cada uma das quais fornecendo uma função de controle bem definida. O serviços associados com as várias UFs são:

- A *UF núcleo (kernel)*: fornece os serviços que permitem ao usuário estabelecer uma associação com um servidor, inicializar uma transação (início do diálogo), executar operações associadas a esta transação (transferência de dados), sinalizar um erro de volta para a transação inicializadora, e, fechar a transação e a associação (fim do diálogo);

- A *UF Commit*: fornece os serviços adicionais que são necessários para permitir que um cliente leve adiante uma transação de acordo com as propriedades ACID, e portanto, fazendo uso dos serviços do CCR;

- A *UF de controle polarizado*: fornece os serviços que permitem que um servidor seja acessado por somente um usuário de cada vez;

- A *UF de transações sem ligações*: permite que um superior bloqueie a criação de uma sub-árvore de transações e a desbloqueie posteriormente;

- A *UF de aperto de mãos (handshake)*: permite que dois clientes sincronizem as funções de processamento associadas com uma transação.

11.4. O Processamento de Transações Distribuídas

Falamos há pouco sobre dois tipos de transações: *transações de commit simples* (ou simplesmente, *transações simples*) e *transações multi-commit*. As transações simples envolvem um único gerente de recursos (GR). Uma transação simples pode ser distribuída se ela acessar um nó remoto na rede, ou se ela acessar dois ou mais bancos de dados no mesmo nó. Assim, uma transação simples pode ser distribuída localmente ou remotamente, porém, irá envolver sempre um único GR.

Uma transação *multi-commit* sempre envolve dois ou mais GRs e deve utilizar o GT conforme mostramos na figura 11.13 anterior). Os GRs podem ser locais ou remotos. Transações *multi-commit* são distribuídas por default. A definição de uma transação distribuída difere da definição tradicional porque uma transação distribuída (na nova definição) não precisa fazer acessos remotos, isto é, uma transação distribuída pode existir em uma única máquina na ausência de uma rede.

Quando uma operação é remota, o nó da transação inicial passa adiante a operação para o nó remoto apropriado onde os dados desejados se encontram, e a operação é então executada como se fosse local a este nó remoto. A execução desta operação local pode ser vista como uma sub-transação da transação inicializadora. É criado um descritor de transação no nó remoto para esta sub-transação. O TID da transação inicializadora é armazenado no descritor da sub-transação. Este TID contém o endereço do nó da transação inicializadora. Caso exista somente uma cópia na rede dos dados que se desejem acessar, então a operação remota é passada adiante para o nó onde esta cópia única reside. Por outro lado, se os dados na rede estiverem replicados, então o nó da transação inicial pode escolher dentre vários nós para passar adiante a operação.

É necessário que se faça uma distinção entre transações simples e *multi-commit*s, e transações locais ou remotas, dentro do contexto da rede. Para as transações simples, e também para as transações distribuídas, o GT, dono do GR envolvido, mantém todos os identificadores de nós (incluindo o nó da transação inicial) envolvidos na execução da transação, e armazena esta informação no descritor da transação. Assim, se uma transação distribuída executa quatro operações remotas em quatro nós distintos, por exemplo, então o descritor de transação para esta transação distribuída irá manter os endereços deste quatro nós. Quando a transação distribuída efetuar o commit, o nó da transação inicializadora irá utilizar os endereços de nós armazenados em seu descritor de transação para poder executar o protocolo 2PC. Os gerenciadores de commit (GCs) em cada nó envolvido na transação são os responsáveis pela execução dos commits das sub-transações em cada nó remoto envolvido nesta transação distribuída. Os endereços dos nós são usados de maneira similar se a transação distribuída abortar. Observe que o GT não irá participar das transações distribuídas simples.

Em transações *multi-commit*, o GT controla os GRs envolvidos em uma transação. Se o envolvimento de determinado GR em uma transação *multi-commit* resultar em uma transação distribuída simples do próprio GR, então o GT deste GR irá executar seu próprio protocolo 2PC.

Deve ficar claro que executar um commit ou aborto de uma transação distribuída é muito mais complexo do que executar um commit ou aborto de uma transação que não seja distribuída. Existem diferenças significativas entre transações distribuídas locais e transações distribuídas remotas no que diz respeito às falhas.

As falhas importantes no contexto das transações locais são aquelas que incluem falhas em processos, queda do sistema, ou queda de algum dispositivo. Todos estes tipos de falhas existem no contexto das transações distribuídas remotas, e neste caso podem ocorrer também falhas em nós na rede e no subsistema de comunicações. Nos casos remotos, alguns dos nós podem se encontrar operacionais, enquanto outros podem ter falhado, gerando assim as chamadas "falhas parciais". As falhas parciais introduzem sérios problemas que podem afetar a confiabilidade dos sistemas distribuídos.

Apesar de todos estes problemas, muitos argumentos técnicos endossam a migração para os sistemas que implementam transações distribuídas, incluindo:

- *Disponibilidade/Confiabilidade*: se um nó falha, os nós restantes podem estar aptos a substituir este nó em falha.
- *Tempo de resposta*: quando os dados são armazenados próximos dos clientes reduz-se o tempo necessário para o acesso a estes dados; esta característica é particularmente importante para redes de longo alcance.
- *Custo*: a descentralização pode reduzir o tráfego de comunicações, e a economia que se faz pode ultrapassar os custos para a aquisição das facilidades distribuídas.
- *Capacidade*: as necessidades de algumas aplicações ultrapassam a capacidade de um único computador, sendo necessária a sua execução em vários nós da rede.
- *Modularidade*: o crescimento modular permite que nós sejam adicionados por demanda, aumentando assim a capacidade e a funcionalidade da rede.

11.4.1. Falhas em Sistemas Distribuídos

As falhas nos sistemas distribuídos são normalmente falhas em nós, falhas de comunicações, ou outras falhas que inviabilizam a troca de mensagens entre dois ou mais processos.

Falhas em Nós

Existem dois modelos para o tratamento de falhas em nós da rede [Byr94]. O primeiro modelo chama-se *modelo falhou-parou* ("fail-stop"), e baseia-se no conceito de que um nó ou está funcionando corretamente ou simplesmente não está funcionando. Quando um nó está no ar, ele jamais será considerado como funcionando de maneira incorreta (neste modelo em particular, pois na prática, isto é perfeitamente possível de ocorrer). O segundo modelo é chamado de *modelo Bizantino*, e neste caso, um nó pode ser considerado, apesar de estar no ar, funcionando de maneira incorreta (este modelo está mais próximo do mundo real). Assim, ou um nó vai estar em falha, ou simplesmente não vai estar funcionando, ou ainda, ele pode estar em um modo de *semi-falha* e funcionando incorretamente. É muito difícil de se detectar problemas no modelo Bizantino.

Do ponto de vista da rede, uma falha em um nó resulta em uma falha parcial da rede. Este é um tipo de falha muito difícil de se resolver, pois os nós ativos podem não conhecer o estado dos nós que tenham falhado. Os nós ativos podem ser bloqueados ou desabilitados para executar o commit ou o aborto de transações, até que se resolva o problema.

Falhas de Comunicações

As falhas de comunicações inviabilizam a troca de mensagens entre os nós da rede. Elas incluem falhas em ligações que causam a perda de mensagens, ruído que pode corromper uma mensagem, software incorreto, linhas de comunicações partidas, dentre muitas outras. A perda de mensagens pode ser tratada por meio de sua retransmissão. As falhas de ligações, linhas partidas, falhas em nós, são chamadas de falhas de partições. As falhas de partições fragmentam a rede em duas ou mais sub-redes que passam a ser chamadas de de partições.

Se uma rede suporta a replicação de dados, então os problemas associados com falhas de comunicações, particularmente as falhas de partições, são muito difíceis de serem resolvidos. A exatidão dos dados replicados deve ser sempre preservada. A replicação de dados é correta quando ela é *mutuamente consistente*, ou seja, quando todas as réplicas possuem o mesmo valor. Uma maneira de se garantir a exatidão para os dados replicados é por meio da garantia de que "a execução concorrente de transações em dados replicados é equivalente à execução serial destas transações em dados não replicados". Esta propriedade é chamada de *serialização de cópia-única*. Quando falhas de comunicações ocorrem com dados replicados na rede a viabilização da propriedade de *consistência mútua* entre todos os nós que contenham réplicas de dados torna-se bastante complexa.

Outras Falhas

Outras falhas típicas em ambientes distribuídos consistem normalmente de um dos seguintes casos:

- *Queda do sistema (falha suave)*: uma queda do sistema pode ser causada por um *"bug"* no código do sistema TP, ou pode ser uma falha do próprio sistema operacional, ou ainda, uma falha de hardware. O conteúdo da memória interna é perdido. Ou seja, a cache e todas as tabelas que suportam o sistema TP são perdidas;

- *Falha de Transações*: esta falha resulta de um aborto de uma transação. O aborto pode ser inicializado pela própria aplicação dona da transação, ou então por um sistema TP ao tentar resolver um deadlock;

- *Falha de Processo*: as transações são executadas por processos do sistema operacional. Se um processo cai inadvertidamente, ou então é morto (*"killed"*) por um operador do sistema, então a transação associada a este processo irá falhar e será abortada;
- *Falhas no meio (falha dura)*: arquivos e bancos de dados são armazenados em memória não volátil, geralmente em discos. Uma falha em um disco pode ocorrer quando o cabeçote de leitura/gravação tocar a superfície de gravação (aterrissar). Em alguns casos, somente partes do disco ficam prejudicadas.

11.4.2. O Processamento do Aborto de Transações

Transações podem ser abortadas por várias razões. A mais comum destas razões acontece quando uma transação está envolvida em um conflito ("deadlock"). Uma aplicação pode ainda iniciar um aborto baseada em certas informações que ela tenha acumulado em conseqüência de um código de estado ("return code") que tenha sido recebido de um sistema TP. Se o processo que estiver executando uma transação cair, então esta transação será ser imediatamente abortada.

O percentual de todas as transações que não completam com sucesso, isto é, não executam o commit, é bem pequena (< 5 %). Para uma aplicação em particular, a porcentagem de transações que abortam a si mesmas é geralmente constante e depende da freqüência dos erros nos dados de entrada e dos testes de consistência executados pela própria transação. A porcentagem de transações abortadas por um sistema TP, principalmente devido aos deadlocks, é determinada pelo grau de concorrência e pela *granularidade de bloqueio*. Quanto maiores o grau de concorrência e a granularidade de bloqueios, maior a probabilidade de ocorrerem conflitos ("deadlocks").

O processamento de abortos e da recuperação de falhas devem ser projetados de forma que um "overhead" mínimo seja gerado durante o processamento normal do sistema TP, pois 95 % de todas as transações processadas por um sistema TP normalmente completam com sucesso, e portanto, o impacto causado pelas transações que abortam deve ser minimizado sempre que possível.

Em caso de aborto de uma transação o sistema TP executará um "rollback" de todas as atualizações efetuadas por esta transação. Se forem usados mecanismos de bloqueio para o controle de concorrência, então serão tratadas também as liberações destes bloqueios. A transação coordenadora retorna um código de estado para a aplicação indicando que a transação foi abortada. A aplicação decide em seguida o que será feito, incluindo possívelmente a retomada desta transação.

O processamento de abortos pode ser muito simples e rápido, ou então pode ser complicado e consumir muito tempo de processamento. Muitas aplicações não conideram que possam ocorram abortos durante a sua execução, porém elas são sempre programadas para suportarem eventos desta natureza. A simplicidade e a rapidez com que os abortos podem ser tratados irá depender do tipo de informações de recuperação guardadas no arquivo de *log* e também no fato de permitir-se ou não a propagação de atualizações *"commited"* para arquivos permanentes.

Os abortos locais são muito simples se comparados com os abortos em transações distribuída. Quando um aborto ocorre em uma transação distribuída, a lista de nós envolvidos na transação é determinada e o coordenador envia uma mensagem para cada nó subordinado instruindo-os para abortarem. O envio de mensagens para outros nós na rede aumenta o tempo necessário para o processamento do aborto.

11.4.3. A Granularidade de Bloqueios

Uma das mais importantes considerações de projeto para a implementação dos mecanismos de bloqueio para o controle de concorrência é a que diz respeito à *granularidade de bloqueios* [OV91]. O nível de granularidade pode ser definido para arquivos, blocos ou registros (ou bases de dados, relações ou tuplas). O bloqueio a nível de campo (ou item de dados) é em geral muito caro com relação ao tempo de CPU e à quantidade de memória consumidos para seu controle. Existem duas regras gerais que resumem os estudos sobre desempenho nos projetos de mecanismos de bloqueio:

- a *granularidade fina* é mais apropriada para transações simples que acessam somente alguns "grânulos" de dados;
- a *granularidade grosseira* é mais apropriada para transações que acessam muitos "grânulos" de dados.

A escolha do nível de granularidade de bloqueios é uma negociação entre o grau de concorrência desejado e o overhead que será causado pelos bloqueios. O *overhead* de bloqueios está obviamente relacionado com o tamanho do grânulo, pois um transação que precise estabelecer e liberar um grande número de bloqueios irá gerar overhead ao invocar o gerente de bloqueios ("Lock Manager") muitas vezes. Observe que o tamanho do grânulo não é importante para a exatidão dos dados, de modo que o princípio de serialização de transações é independente da granularidade de bloqueios.

Existe uma estreita relação entre a granularidade de bloqueios e a granularidade utilizada para o arquivo de *log*. Se o componente de recuperação do gerente de commits (GC) for projetado para efetuar o *log* físico a nível de páginas, então utilizar granularidade de bloqueios menor que uma página é questionável (e não deve ser usada), pois certamente haverá perda de desempenho (*overhead*).

Geralmente, a granularidade de bloqueios é implementada a nível de registro ou de página. Gravar registros no arquivo de *log* é mais eficiente do que usar páginas, devido à menor quantidade de informações que precisa ser gravada, o que evita muitas operações de E/S para o disco (isto é, vários registros são primeiramente atualizados no buffer para serem em seguida gravados no disco quando a página onde o buffer estiver alocado for liberada). Assim, é mais comum que a granularidade a nível lógico seja de registro, com uma granularidade a nível físico sendo de página.

No passado, o mais popular nível de granularidade de bloqueios empregado era a página (ou bloco). No entanto, o bloqueio a nível de páginas limita o grau de concorrência que pode ser obtido, especialmente em alguns SGBDs relacionais. Por exemplo, como vimos na transação inicial de débito/crédito apresentada na figura 11.1 anterior, o bloqueio a nível de página só permitiria um grau satisfatório de concorrência se o número da conta corrente, e o número da agência forem aleatórios e se a tabela de agências contiver uma linha (tupla) por página. Estas premissas asseguram uma baixa probabilidade de que a execução concorrente de transações de débito/crédito irão requisitar bloqueios no mesmo conjunto de páginas.

11.4.4. O Processamento da Recuperação de Falhas

A recuperação é o processo de se restabelecer uma base de dados (ou um arquivo) de volta ao seu estado consistente após a ocorrência de algum tipo de falha. É necessário que se restaure o estado de um banco de dados, em um nó que tenha falhado, para um estado consistente quando este nó for reinicializado. A complexidade desta recuperação irá depender da existência ou não de dados replicados. Existem três inconsistências possíveis de ocorrerem neste caso:

- as atualizações originadas de nós ativos podem não ter sido passadas adiante para o nó em falha. Isto inclui as atualizações em andamento no momento da falha e também todas aquelas atualizações que tenham ocorrido subseqüentemente à falha;
- as atualizações, originadas de um nó em falha, que podem ter sido passadas adiante por este mesmo nó em falha. Esta possibilidade surge quando existem dados replicados e uma atualização nestes dados é propagada antes que a cópia local destes dados seja atualizada;
- outra inconsistência que pode ocorrer quando os dados estão replicados é quando um nó em falhas tiver alterado um banco de dados sem que os outros nós tenham recebido a propagação desta alteração. Uma solução para este problema pode ser a manutenção de uma lista das alterações executadas localmente e que ainda não tenham sido propagadas para outros nós.

Particularmente neste último caso, pode-se prevenir o problema proibindo-se que um nó atualize o banco de dados local até que seja recebida a confirmação da atualização nos outros nós. Este atraso, no entanto, pode ser considerável, pois envolve um tempo muito extenso para que sejam recebidas todas as mensagens de confirmação. A recuperação de falhas baseia-se na idéia de que um gerente de bloqueios do GC mantém informações redundantes das atualizações de bases de dados, de tal forma que estas informações redundantes ficam disponíveis em um meio de armazenamento seguro. O gerente de recuperação utiliza estas informações redundantes para a recuperação de falhas. Transações de somente-leitura não adicionam quaisquer informações para estes repositórios redundantes (arquivos de *log* ou jornais), pois elas simplesmente não alteram as bases de dados.

Qual informação redundante deve ser coletada, e como ela está organizada e é utilizada, irá depender da complexidade dos esquemas de commit e dos mecanismos de recuperação de falhas. Para decidir-se sobre a implementação de um gerente de recuperação, é importante que o projetista do sistema TP determine se os algoritmos para recuperação rápida irão causar ou não um peso adicional no processamento normal das transações. Se isto ocorrer, então esta certamente não será uma boa decisão, pois quedas no sistema são considerados eventos de relativa raridade.

Gostaríamos de enfatizar que os mecanismos para recuperação de falhas devem ser projetados de tal forma que impactem mínimamente o processamento normal. Se este critério for aceito para o projeto de um gerente de recuperação, então ele será usado para garantir os esquemas de recuperação baseados na propagação de páginas modificadas como uma unidade da base de dados permanente, forçando imediatamente todas as páginas modificadas durante o processamento de commits para o disco. Outro esquema que pode ser usado é o de "checkpoints", ou seja, pontos de checagem, orientados para transações. Estes esquemas de recuperação são atraentes para a recuperação de quedas do sistema, porém o overhead gerado pela sua implementação em um esquema de processamento normal de transações é normalmente muito pesado para que se justifique o seu uso.

11.5. O Protocolo de Commit de Duas Fases (ou Protocolo 2PC)

Em seções anteriores falamos sobre o protocolo 2PC e sua importância. Retornamos ao assunto para tratá-lo agora com maiores detalhes. Assim, a principal função do gerente de commits (GC) é assegurar que informações redundantes sejam armazenadas em arquivos de *log* em um meio de armazenamento estável. Ele também é responsável pela atualização de arquivos de dados em memória cache (e é quem descarrega estes dados para disco, por meio de uma operação "flush"), e notifica o usuário da ocorrência dos commits, liberando os bloqueios mantidos pelas transações.

O processamento de commits em sistemas centralizados é relativamente simples quando comparado com o processamento de commits em sistemas distribuídos. Em sistemas distribuídos cada nó tem um sistema TP local que trata o processamento de commits na atualização de todos os recursos neste nó. Quando uma transação distribuída acessa um arquivo em um nó remoto, o TID desta transação é passado juntamente com o pedido de acesso para este arquivo remoto.

O nó onde a transação distribuída origina-se é referenciado como *nó casa* (*"home node"*) ou como *nó coordenador*. Os nós remotos envolvidos na transação distribuída são chamados de nós subordinados ou participantes. Uma lista dos nós subordinados é mantida em uma tabela no descritor da transação inicial no nó coordenador. A idéia geral por detrás do protocolo 2PC é que antes de poder ocorrer um commit, o coordenador e seus subordinados devem todos concordar com a execução do commit, senão a transação será abortada.

O protocolo 2PC é um protocolo do tipo "aperto de mãos" (*"handshaking"*) com duas fases distintas. Na primeira fase, o coordenador pede a todos os subordinados para votarem SIM ou NÃO com relação à possibilidade de execução do commit. Na segunda fase, o commit irá ocorrer de fato, se todos os nós votarem SIM, ou será executado um aborto se um ou mais nós subordinados votarem NÃO.

A seguir descrevemos os passos para os nós coordenador e subordinados nas fases 1 e 2:

Coordenador - Fase 1:

1. o coordenador obtém a lista dos nós envolvidos na transação;
2. o coordenador grava um registro de *log* "inicia 2PC" no seu arquivo de log. Este registro contém uma lista com todos os nós envolvidos na transação;
3. o coordenador envia uma mensagem "prepare para commit" para cada nó subordinado;
4. o coordenador aguarda a resposta dos nós com relação à mensagem de preparação. A resposta de cada nó vem na forma de um voto SIM ou NÃO para o commit;
5. caso ocorra um estouro de tempo ("timeout") antes que todos os subordinados tenham respondido, então um registro de *log* "aborto" é gravado no *log* do coordenador, e uma mensagem de aborto é enviada para todos os outros nós envolvidos na transação.

Subordinados - Fase 1:

1. cada subordinado aguarda por uma mensagem "prepare para commit" do coordenador;
2. se ocorrer um estouro de tempo antes que chegue a mensagem de preparação para commit, ou se a transação já tiver falhado, então o nó grava um registro de "aborto" no seu arquivo de log;
3. em resposta à mensagem de preparação, cada nó grava seus registros de *log* em disco, ficando assim prontos para executar o commit ou o aborto;
4. se o nó vota SIM, então ele grava um registro "commit pendente" no seu arquivo de *log* e envia uma mensagem SIM para o coordenador. É importante que se grave o registro de *log* antes de enviar a mensagem SIM para o coordenador;
5. se o nó vota NÃO, então ele grava um registro de "aborto" no seu arquivo de *log* e envia um voto NÃO para o coordenador. O registro de *log* pode então ser gravado após o envio do voto NÃO.

Coordenador - Fase 2:

1. se um ou mais nós subordinados votarem NÃO, então o coordenador grava um registro de "aborto" no seu arquivo de *log* e envia uma mensagem de aborto para todos os nós envolvidos na transação;
2. se todos os votos forem SIM, incluindo o voto do próprio coordenador, então o coordenador grava um registro de "commit" no seu arquivo de *log* e envia uma mensagem instruindo todos os outros nós envolvidos a executarem o commit;
3. o coordenador aguarda um "acknowledgement" (aviso de recebimento) de todos os outros nós envolvidos na transação;
4. quando todos os nós tiverem respondido, um registro de "transação efetivada" é gravado no arquivo de *log* do coordenador. O registro "transação efetivada" do *log* irá guardar o fato de que todos os nós estão conscientes do ocorrido.

Subordinados - Fase 2:

1. cada nó aguarda no estado "commit pendente" (SIM) pela mensagem de decisão que virá do coordenador. Neste ponto, o nó fica aguardando e está preparado para o commit ou para o aborto, dependendo do resultado da decisão baseada nos votos recebidos pelo coordenador;
2. se ocorrer um estouro de tempo antes que a mensagem de decisão seja recebida do coordenador, então inicializa-se um protocolo de terminação;
3. se a mensagem de decisão for no sentido de executar o commit, então grava-se um registro de "commit" no arquivo de log, fazendo-se múltiplas atualizações nos arquivos de dados, se elas ainda não tiverem sido feitas. Grava-se então um registro de *log* de "transação completada", liberam-se todos os bloqueios e recursos, e envia-se uma mensagem de "acknowledgement" para o coordenador;
4. se a mensagem de decisão for um aborto, grava-se então um registro de "aborto" no arquivo de log, e restauram-se os arquivos de dados para seus estados anteriores às atualizações de quaisquer transações, liberando-se todos os bloqueios e outros recursos, e enviando-se uma mensagem de "acknowledgement" para o coordenador.

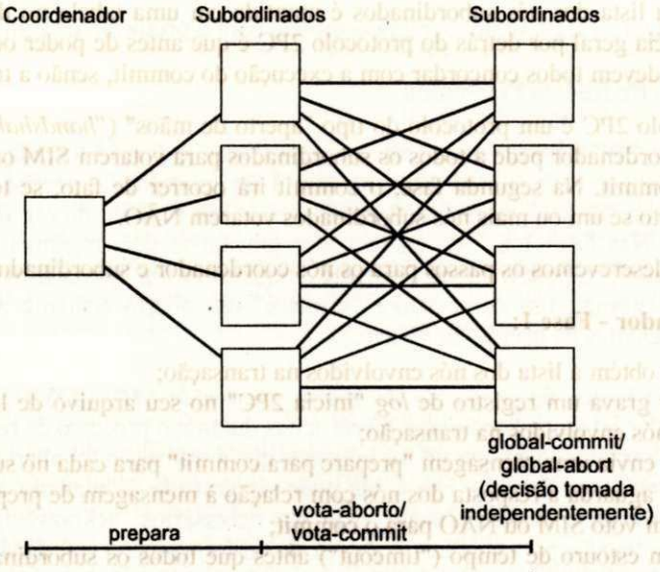


Figura 11.18. Resumo do protocolo de Commit de duas fases [OV91]

Observe que qualquer falha no coordenador antes de se completar o passo 1 ou o passo 2 de sua fase 2 irá resultar no aborto da transação. Isto porque na reinicialização, o *log* de transação do coordenador irá conter um estado de indecisão para a transação. Portanto, na reinicialização, o coordenador irá enviar uma mensagem de aborto para todos os outros servidores. Se o coordenador falhar em qualquer momento durante a sua fase 2, antes de completar o passo 4, então o *log* de transação do coordenador irá conter um registro de *log* do tipo "commit", porém não conterá um registro de *log* de "transação completada". Na reinicialização, o coordenador irá enviar mensagens de commit para todos os outros nós envolvidos na transação.

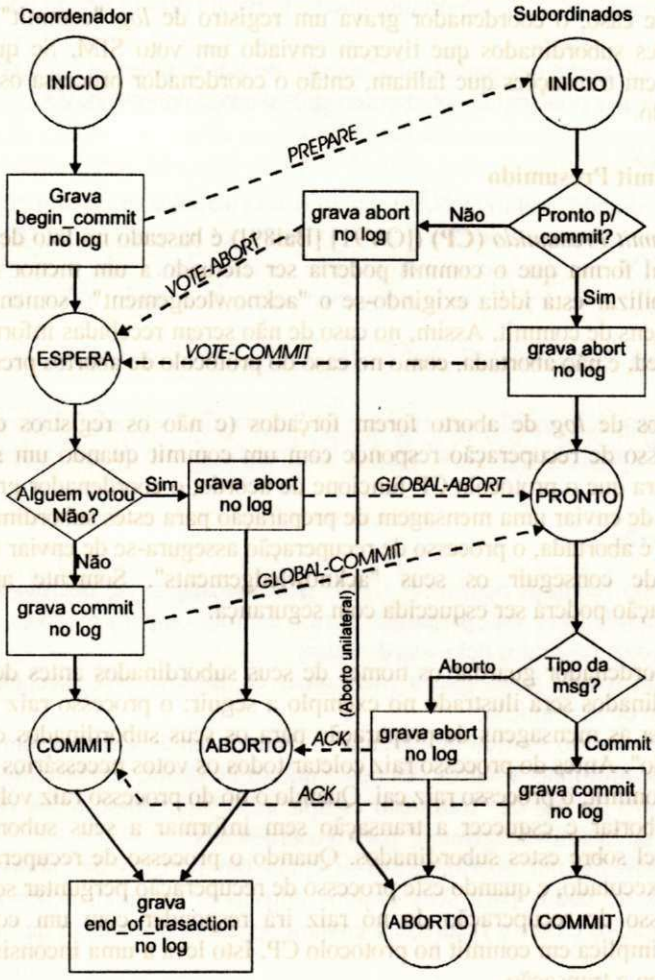


Figura 11.19. O fluxograma completo para o protocolo de commit de duas fases [OV91]

11.5.1. O Protocolo de Aborto Presumido

O protocolo 2PC descrito anteriormente requer uma gravação forçada, isto é, uma gravação sincrona de dois registros de *log* por cada subordinado: na fase 1 do subordinado precisa-se de um registro de *log* do tipo "commit-pendente" ou "aborto", e na fase 2 precisa-se de um registro de *log* do tipo "commit" ou "aborto" para cada subordinado. Nas fases 1 e 2 do coordenador gravam-se dois registros de log: "commit" (ou "aborto") e "transação completada", respectivamente. Somente o registro de *log* "commit" (ou "aborto") é forçado para o disco. No protocolo 2PC normal, um coordenador "esquece" um aborto somente após ele estar certo que todos os subordinados estão cientes da decisão de aborto.

O princípio básico do *protocolo de aborto presumido (AP)* ([OV91] [Bal89]) é que quando consultas vindas do coordenador ou dos subordinados forem feitas durante o processo de recuperação, nenhuma informação a respeito do estado da transação em um nó irá informar que a transação foi abortada. Isto significa que o registro de *log* "aborto" não precisa ser forçado, isto é, ele pode ser gravado de maneira assíncrona pelo coordenador e por cada um dos subordinados. Adicionalmente, não são necessários "acknowledgements" (avisos de recebimento) enviados pelos subordinados para sinalizar o aborto, de tal forma que o coordenador não precisará mais gravar um registro de *log* do tipo "transação completada" após o registro de *log* de "aborto".

Um subordinado de *somente-leitura* envia um voto de somente-leitura ("read-only") em resposta à mensagem "prepare para commit" vinda do coordenador, libera seus bloqueios e esquece a transação. Um subordinado de somente-leitura não grava registros de log, e nem mensagens de commit/aborto precisam ser enviadas pelo coordenador para um subordinado deste tipo.

Uma transação é parcialmente de somente-leitura se pelo menos um dos subordinados na árvore de processos não executam atualizações no banco de dados. Para que uma transação parcialmente de somente-leitura tenha sucesso, é necessário um voto SIM de cada subordinado executando a atualização, e os outros votos podem ser de somente-leitura. Neste caso, o coordenador grava um registro de *log* "commit" e envia a mensagem de commit somente para aqueles subordinados que tiverem enviado um voto SIM. Se qualquer dos subordinados votarem NÃO, como ocorre em transações que falham, então o coordenador processa os abortos de acordo com o protocolo de aborto presumido.

11.5.2. O Protocolo de Commit Presumido

O protocolo de *Commit Presumido* (**CP**) ([OV91] [Bal89]) é baseado no fato de a maioria das transações executarem o commit, de tal forma que o commit poderia ser efetuado a um menor custo de complexidade e processamento. Pode-se viabilizar esta idéia exigindo-se o "acknowledgement" somente para as mensagens de aborto, e não para as mensagens de commit. Assim, no caso de não serem recebidas informações, presume-se que a transação tenha sido commited, e não abortada, como no caso do protocolo de abortos presumido anterior.

Se somente registros de *log* de aborto forem forçados (e não os registros de *log* de commit) pelos subordinados, então o processo de recuperação responde com um commit quando um subordinado executa uma consulta após uma queda. Para que o protocolo CP funcione de acordo, o coordenador grava os nomes de todos os subordinados em disco antes de enviar uma mensagem de preparação para estes subordinados. Assim, quando o nó coordenado cai e a transação é abortada, o processo de recuperação assegura-se de enviar mensagens de aborto para todos os subordinados e de conseguir os seus "acknowledgements". Somente após conseguir todos os "acknowledgements" a transação poderá ser esquecida com segurança.

O motivo para o coordenador guardar os nomes de seus subordinados antes de emitir as mensagens de preparação para estes subordinados será ilustrado no exemplo a seguir: o processo raiz na árvore de commits do protocolo de duas fases envia as mensagens de preparação para os seus subordinados e um destes subordinados entra em estado de "preparado". Antes do processo raiz coletar todos os votos necessários para a tomada de decisão sobre abortar ou executar o commit, o processo raiz cai. Quando o nó do processo raiz volta a funcionar, o processo de recuperação no nó irá abortar e esquecer a transação sem informar a seus subordinados, pois não existe informação alguma disponível sobre estes subordinados. Quando o processo de recuperação, no nó subordinado, que já estava preparado for executado, e quando este processo de recuperação perguntar sobre o estado da transação que foi preparada, o processo de recuperação do nó raiz irá responder com um commit, pois o estado de "informações indisponíveis" implica em commit no protocolo CP. Isto leva a uma inconsistência pois o processo de recuperação no nó raiz abortou a transação.

No protocolo CP todos os registros de *log* de aborto são forçados para o disco, e todas as mensagens de aborto são "acknowledged" pelos subordinados. Os registros de *log* de commit não precisam ser forçados pelos subordinados. O coordenador força a gravação tanto dos registros de *log* de commit quanto de aborto. O coordenador grava um registro de *log* de "transação completada" somente após um registro de *log* de "aborto". Para transações tipo somente-leitura no protocolo de aborto presumido anterior, não são gravados registros de log, enquanto no protocolo CP um registro de *log* "commit" é gravado no final da primeira fase e em seguida o coordenador esquece a transação. Para aquelas transações totalmente de somente-leitura, o coordenador grava dois registros: um registro de *log* contendo os nomes de seus subordinados, e força este registro para disco e um registro de *log* tipo "commit" (não forçado). O coordenador envia uma mensagem de preparação para cada um dos seus subordinados. Os subordinados não gravam registros de log, porém cada um envia uma mensagem de somente leitura para seu coordenador.

Durante a execução de uma transação parcialmente de somente leitura, o coordenador envia duas mensagens: prepare e commit para os subordinados atualizadores, e somente uma mensagem *prepare* para os subordinados de somente-leitura. O coordenador grava dois registros de log: um registro contendo o nome de seus subordinados e um registro de commit. Ambos os registros de *log* são forçados. Um subordinado de somente-leitura envia somente um voto de somente-leitura. Um subordinado atualizado envia uma mensagem de PRONTO (para o commit ou aborto) e grava dois registros de log: um registro de *log* de "commit-pendente" que é forçado para disco e um registro de *log* de "commit" que não é forçado.

11.6. Monitores de Processamento de Transações

Os monitores de processamento de transações são considerados *"middleware"*, isto é, software que se situa entre as aplicações e o sistema operacional (ou seja, um subsistema). Os sistemas OLTPs implementados como monitores de transações voltados para sistemas abertos e distribuídos são todos baseados em ambientes UNIX. Apresentamos alguns destes sistemas monitores de transações para os ambientes da computação distribuída, incluindo o Tuxedo, o TOP END, e o Encina.

Os monitores de transações para ambientes distribuídos tipicamente implementam as funções já vistas em seções anteriores, como:

- suporte a transações que impactam múltiplos bancos de dados distribuídos;
- gerenciamento do fluxo de transações e distribuição de carga;
- envio e recebimento de mensagens entre os nós da rede;
- controle dos gerenciadores de recursos como SGBDs ou sistemas de arquivos por meio de interfaces padronizadas.

11.6.1. Tuxedo

O **Tuxedo** ([Dew93] [Cas93]) é um produto do Unix Systems Lab (USL) que foi adquirido da AT&T pela Novell em 1993. O Tuxedo é na realidade uma interface de gerenciamento de transações que fornece um conjunto de especificações para gerenciamento de aplicações e recursos. Esta interface é chamada de **ATMI (Application Transaction Manager Interface)** [Dew93], e suporta as seguintes características:

- transparência de localização;
- ajuste automático de carga;
- conversão transparente entre formatos de dados;
- roteamento sensível ao contexto;
- processamento de prioridades;
- independência da rede.

Adicionalmente, o Tuxedo utiliza a interface da camada de transporte do Unix System V chamada **TLI (Transport Layer Interface)** [Bil94] para fornecer acesso a tecnologias de redes como TCP/IP, OSI, e a uma interface proprietária para a ligação com mainframes IBM chamada APPC/LU6.2 (Advanced Program to Program Communication/Logical Unit 6.2) [Dew93]. O gerenciador de transações e o gerenciador de recursos do Tuxedo utilizam interfaces padronizadas para comunicações com os gerenciadores de recursos a nível do sistema operacional. A interface ATMI trata as transações que envolvem mais de um gerenciador de recursos e mais de uma localização física (ou seja, os ambientes de transações globais) como se fossem uma unidade lógica de trabalho. Para uma transação global ter sucesso, todas as transações que a compõem devem ter sucesso (implementa o protocolo 2PC).

Recentemente o USL anunciou o produto **TETP (Tuxedo Enterprise Transaction Processing)** [Cas93] que consiste dos produtos Tuxedo/Ws e Tuxedo/Host. Tuxedo/Ws fornece APIs para estações Unix, OS/2 e DOS poderem atuar como clientes. Tuxedo/Host fornece uma infra-estrutura para a construção de gateways entre as aplicações herdados dos ambientes de mainframe.

11.6.2. TOP END

O TOP END é um produto da NCR Corporation baseado no modelo Distributed Transaction Processing do consórcio X/Open, e utiliza as interfaces XA (X/Open resource manager) para o acesso a bancos de dados do mundo Unix. Na figura 11.20 , são apresentados os módulos que compõem o TOP END, o qual divide o processamento de transações em módulos que podem ser distribuídos entre os nós de uma rede para permitir maior eficiência e flexibilidade às aplicações distribuídas em-linha.

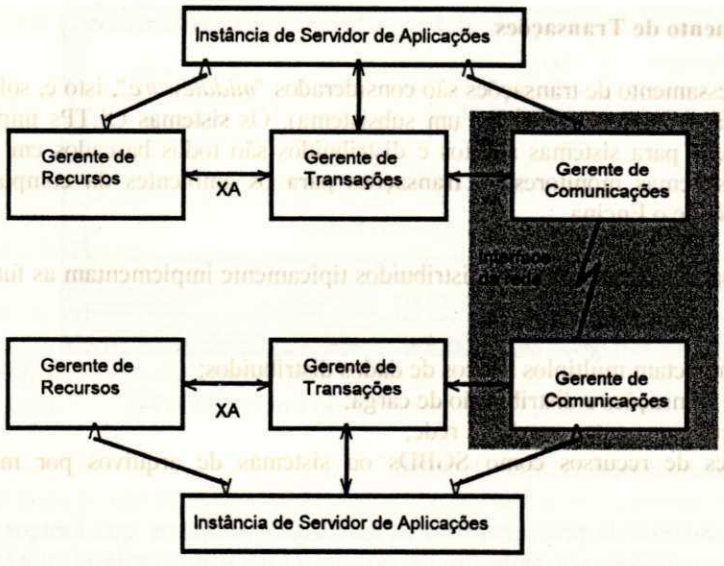


Figura 11.20. Os módulos do TOP END [Dew94]

O TOP END é similar em arquitetura e funcionalidade ao Tuxedo. As principais diferenças são:

- *Administração*: em TOP END, as definições de recursos podem ser criadas interativamente e armazenadas em um repositório. Tuxedo utiliza o editor do Unix para criar estas definições e o sistema de arquivos do próprio Unix para armazená-las.
- *Comunicações*: o TOP END apresenta suporte direto para a maioria dos protocolos de redes. Tuxedo só fornece suporte à TLI do Unix System V.
- *Segurança*: TOP END suporta o sistema de segurança Kerberos do MIT. O Tuxedo só fornece segurança a um baixo nível de autorização.
- *Ajuste de carga*: o ajuste de carga executado pelo TOP END é dinâmico e sensível à rede. Em Tuxedo, ele é pré-definido.

As empresas AT&T e NCR foram fundidas recentemente, de tal forma que é muito provável que um dos produtos, TOP END da NCR ou Tuxedo da AT&T (agora da Novell) sejam descontinuados. TOP END é considerado um produto mais robusto, porém o Tuxedo possui uma grande base instalada. No futuro veremos quem será o vencedor.

11.6.3. Encina

O Encina ([Dew93] [Cas93]) é baseado nas tecnologias do consórcio X/Open e do DCE da OSF. É na realidade uma linha de produtos da empresa americana Transarc Corporation, e fornece software de gerenciamento de transações altamente confiável e amigável para os ambientes da computação distribuída. A versão comercial do Encina foi liberada oficialmente no primeiro semestre de 1993, com funcionalidades do DCE como ferramentas de gerenciamento e de comunicações, transparência de localização de recursos, e segurança combinadas com ferramentas para o desenvolvimento de aplicações. Encina possui seu próprio ambiente *"multi-threaded"* e suporta transações aninhadas. Implementa o protocolo de commit de duas fases (2PC) para as transações distribuídas e um protocolo de commits de duas fases local para ser usado por aplicações não distribuídas.

Na figura 11.21 , apresentamos os componentes do Encina, baseados em uma estratégia de duas camadas projetadas para suportar gerenciadores de transações com todas as características necessárias, incluindo gerentes de recursos e outros sistemas distribuídos. Na primeira camada, expandem-se as fundações do DCE para incluir serviços que suportam o processamento de transações distribuídas (chamado Encina Toolkit Executive) e o gerenciamento de dados recuperáveis (chamado Encina Toolkit Server Core). Esta segunda camada consiste dos serviços de processamento de transações baseados em facilidades fornecidas pela primeira camada.

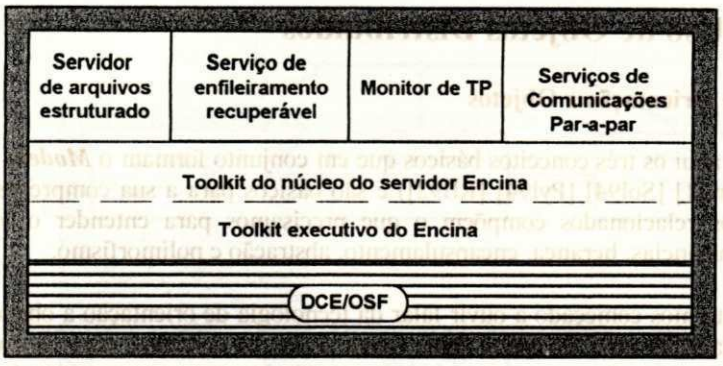


Figura 11.21. Os componentes do Encina [Dew94]

As ferramentas desenvolvidas pela Transarc incluem:

- O *Monitor Encina*: é um monitor de processamento de transações completo, que fornece um ambiente de desenvolvimento altamente amigável, um ambiente de execução altamente confiável, e adicionalmente, um ambiente administrativo;
- O *Servidor de Arquivos Distribuídos Encina*: é um sistema de arquivos orientado a registros que fornece alto desempenho e integridade de transações completa, e pode implementar o protocolo 2PC distribuído por vários servidores;
- O *Serviço de Comunicação Par-a-Par Encina*: suporta comunicações transacionais CPI-C (Common Programming Interfaces for Communications) par-a-par sobre TCP/IP e transporte LU 6.2 para integração com ambientes herdados do mundo IBM;
- O *Serviço de Enfileiramento de Recuperação Encina*: fornece enfileiramento de dados transacionais. Fornece múltiplos níveis de prioridades e pode suportar um grande número de usuários e grandes volumes de dados.

Capítulo 12: O Uso de Objetos Distribuídos

12.1. A Tecnologia de Orientação a Objetos

Introduzimos aqui os três conceitos básicos que em conjunto formam o ***Modelo de Orientação a Objetos*** ([Tay90] [KA90] [Rum91] [Sol94] [Pyl94] [RB92]) e são básicos para a sua compreensão: *objetos, mensagens* e *classes*. Outros termos relacionados compõem o que precisamos para entender o vocabulário sobre objetos: métodos, subclasses, instâncias, herança, encapsulamento, abstração e polimorfismo.

Apesar de só termos começado a ouvir falar da tecnologia de orientação a objetos em anos mais recentes, ela já existe na realidade há mais de vinte anos. Quase todos os conceitos básicos do enfoque de orientação a objetos foram introduzidos pela linguagem de programação **Simula** ([Tay90] [KA90]) desenvolvida na Noruega no final dos anos sessenta.

Simula é uma abreviação para "simulation language", e foi criada para suportar simulações de processos do mundo real por meio do uso de computadores. Seus autores desejavam construir modelos de trabalho precisos de sistemas físicos complexos que poderiam conter milhares de componentes. Já ficava aparente nos anos sessenta que a programação modular seria essencial para a construção de sistemas complexos, e esta modularização é essencial para a linguagem Simula. O que é especial nesta linguagem é a forma como se definem os módulos. Eles não são baseados em procedimentos, como acontece nas linguagens de programação convencionais. Em Simula, os módulos são baseados nos objetos físicos que estão sendo modelados na simulação.

Esta escolha faz muito sentido porque os objetos em uma simulação oferecem uma maneira natural de subdividir o problema a ser resolvido. Cada objeto possui uma abrangência de comportamento a ser modelado, e cada um dos objetos precisa manter algum tipo de informação sobre seu próprio estado [Wag92]. Assim, não precisamos buscar outras maneiras para se empacotar os procedimentos, se o próprio problema já o fizer por sí.

12.1.1. Conceitos Básicos sobre Objetos

O conceito de objetos de software surgiu da necessidade de se modelar objetos do mundo real em simulações de computadores ([Rum91] [Wag92]). Um **objeto** ([Bal89] [Bet94] [Cas93] [Dew93] [Dew94] [Kay94b] [Lu92] [Pyl94] [Ras92d] [RB92] [SG92b] [Sol94] [Tay90] [KA90] [Rum91]) é um "pacote" de software que contém uma coleção de procedimentos relacionados, e, dados. No enfoque de orientação a objetos, os procedimentos recebem uma denominação especial: são chamadas de **métodos** ([Kay94b] [RB92] [Tay90] [KA90] [Rum91]). Para aproveitar alguma terminologia da programação tradicional, os elementos de dados são referenciados como **variáveis** ([Kay94b] [RB92] [Tay90] [KA90] [Rum91]), porque seus valores podem variar com o passar do tempo. A figura 12.1 ilustra estes conceitos.

As figuras nos exemplos a seguir são adaptadas de [Tay90], e se o leitor já possuir conhecimento básico sobre a tecnologia de orientação a objetos pode optar por passar diretamente para a seção 12.6, onde começamos a falar do papel da tecnologia de orientação a objetos nos ambientes da computação distribuída.

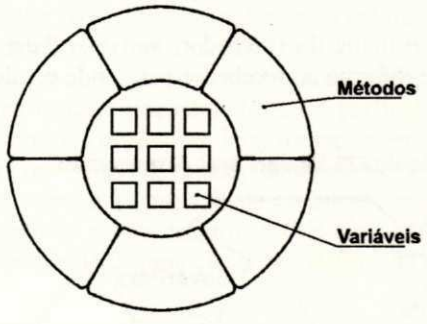


Figura 12.1. Métodos e variáveis no enfoque de orientação objetos

Por exemplo, considere a representação de um veículo muito comumente usado em grandes armazéns de fábricas, conhecidos como empilhadeiras. Este veículo pode exibir uma grande variedade de comportamentos, como por exemplo, movimentar-se de uma localização para outra, ou então carregar ou descarregar o seu conteúdo. Este veículo deve também manter informações sobre suas características inerentes, incluindo o tamanho do pallet, o peso suportado, velocidade máxima, dentre outras. O estado corrente (conteúdo, localização, direção, velocidade) também deve ser mantido.

Para representar este veículo como um objeto, deve-se descrever os seus comportamentos como métodos, e suas características como variáveis. Durante a simulação, o objeto deve executar os seus vários métodos, modificando as suas variáveis quando necessário, para refletir os efeitos de suas ações. A figura 12.2 ilustra os métodos e as variáveis para o modelo de objeto de uma empilhadeira.

Figura 12.2. Métodos e variáveis para o modelo de objeto de uma empilhadeira

O conceito de objeto é muito poderoso, apesar de extremamente simples. Os objetos constituem-se como excelentes módulos de software porque podem ser definidos e mantidos independentemente uns dos outros, com cada objeto formando um universo claro e auto-contido. Tudo que um objeto conhece está expressado por meio de suas variáveis. Tudo o que ele pode fazer está expresso pelos seus métodos.

12.1.2. Mensagens

Os objetos no mundo real podem exibir uma variedade infinita de efeitos uns nos outros: criando, destruindo, levantando, curvando, enviando e muitos outros. Esta enorme variedade faz surgir um problema muito interessante: como podemos representar todos estes tipos de interações por meio de módulos de software? A resposta para este problema veio dos autores da linguagem Simula, que apresentaram uma solução muito elegante, que é o uso de mensagens. Uma **mensagem** ([Kay94b] [RB92] [Tay90] [KA90] [Rum91]) é simplesmente o nome de um objeto seguido pelo nome de um método que este objeto saiba como executar. Se um método precisar de informações adicionais para poder determinar precisamente o que fazer, a mensagem irá incluir estas informações na forma de um conjunto de elementos de dados chamados de **parâmetros**. O objeto que inicia uma mensagem é chamado de **remetente** e o objeto que recebe uma mensagem é chamado de **recebedor**.

Para fazer um veículo automatizado mover-se para uma nova localização, por exemplo, algum outro objeto poderia enviar a seguinte mensagem:

Veículo223 mover-sePara armazém19

Neste exemplo, o *veículo223* é o nome do recebedor, *mover-sePara* é o método que se está querendo executar, e *armazém19* é o parâmetro que informa ao recebedor para onde ele deve ir. A figura 12.3 abaixo ilustra este exemplo.

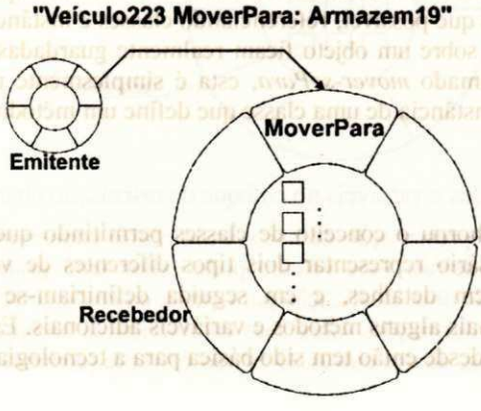


Figura 12.3 A mensagem para o veículo223

Uma simulação orientada a objetos consiste portanto de vários objetos interagindo uns com os outros por meio da emissão de mensagens. Como tudo que um objeto pode fazer é expresso pelos seus métodos, este mecanismo simples suporta todas as possíveis interações entre objetos.

12.1.3. Classes

Em algumas situações uma simulação envolve somente um único exemplo de um tipo particular de objeto. É muito mais comum, no entanto, que seja necessário mais de um exemplo de objeto de cada tipo. Uma fábrica automatizada, por exemplo, pode possuir qualquer quantidade de veículos guiados. Esta possibilidade nos leva a outra consideração: seria extremamente ineficiente se redefinir os mesmo métodos em cada ocorrência única de cada objeto. Para resolver este problema, novamente os autores da linguagem Simula apresentaram uma solução muito elegante: a **classe** ([Kay94b] [RB92] [Tay90] [KA90] [Rum91]). Uma classe é uma "template" (máscara) que define os métodos e variáveis a serem incluídos em um tipo particular de objeto. A descrição dos métodos e variáveis suportados são incluídos uma única vez na definição de uma classe. Os objetos que pertencem a uma classe, chamados de **instâncias** da classe, contêm somente os valores particulares para as variáveis. A figura 12.4 abaixo ilustra o conceito de classe com suas instâncias.

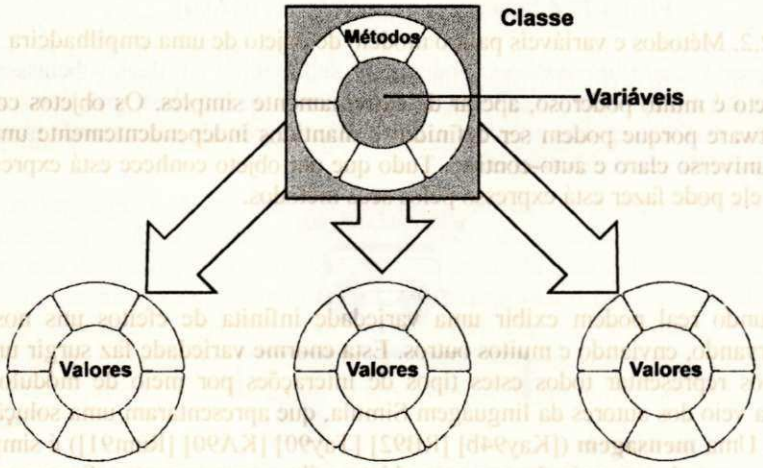


Figura 12.4 Uma classe e suas instâncias

Para continuarmos com o nosso exemplo, uma fábrica simulada deveria conter vários veículos automatizados, cada um dos quais executam as mesmas ações e mantêm os mesmo tipos de informações. Toda a coleção de veículos poderia ser representada por uma classe chamada *VeículosAutomatizados*, e esta classe irá conter as definições de seus métodos e variáveis. Os veículos reais seriam representados por instâncias desta classe, cada um deles com seu nome único (veículo201, veículo202, veículo203, ...). Cada instância irá conter valores de dados que irão representar seus próprios conteúdos e localização. Quando um veículo receber uma mensagem para executar um método, ele deve se voltar para a classe em busca da definição deste método, e em seguida o aplica aos seus próprios valores de dados locais.

Um objeto é, portanto, uma instância de uma classe em particular. Seus métodos e variáveis são definidos na classe, e seus valores são definidos nas instâncias. Para tornar as nossas explicações ainda mais claras, falamos de objetos de maneira geral sempre que possível, referenciando classes e instâncias somente quando for importante para destacar onde as informações sobre um objeto ficam realmente guardadas. Por exemplo, se dissermos que o veículo223 possui um método chamado *mover-sePara*, esta é simplesmente uma maneira mais conveniente de dizermos que o veículo223 é uma instância de uma classe que define um método chamado *mover-sePara*.

12.1.4. Herança

A linguagem Simula melhorou o conceito de classes permitindo que elas fossem definidas umas com relação às outras. Se fosse necessário representar dois tipos diferentes de veículos automatizados, poder-se-ia definir uma classe de veículos em detalhes, e em seguida definiriam-se as outras classes com todas as características da primeira classe mais alguns métodos e variáveis adicionais. Esta estratégia era uma antiga forma de herança, uma característica que desde então tem sido básica para a tecnologia de orientação a objetos.

Herança ([Hol93b] [Kay94b] [RB92] [Tay90] [KA90] [Rum91]) é um mecanismo através do qual uma classe de objetos pode ser definida como um caso especial de uma ou mais classes genéricas, incluindo automaticamente as definições de variáveis e métodos da classe geral. Casos especiais de classes são as **subclasses** de uma classe; a classe mais genérica por sua vez é chamada de **superclasse** dos seus casos especiais. Além dos métodos e variáveis que são herdados pelas subclasses elas podem adicionalmente definir seus próprios métodos e variáveis e podem sobrepor quaisquer das características herdadas. A figura 12.5 ilustra o conceito de subclasses e superclasses.

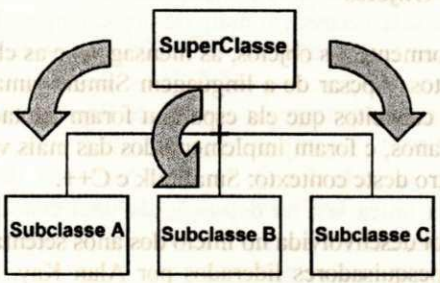


Figura 12.5 Subclasses e superclasses [KA90]

Por exemplo, a classe *veículoAutomatizado* poderia ser subdividida em duas subclasses, *Empilhadeiras* e *RoloCompressor*, cada uma das quais herdando as características gerais de sua classe pai, como mostrado na figura 12.6 . Cada subclasse poderia estabelecer suas próprias características especiais adicionando novas características à classe pai, ou pela sobreposição de seu comportamento.

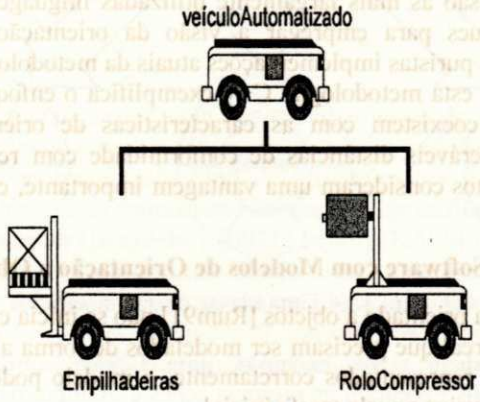


Figura 12.6 Duas subclasses para veículos automatizados

As classes podem ser aninhadas em qualquer nível, e a herança será automaticamente acumulada nos níveis abaixo. A estrutura em forma de árvore resultante é conhecida como uma **hierarquia de classes** ([Tay90] [[KA90] [Rum90] [Pyl94]). Uma classe chamada *peça*, por exemplo, poderia ser subdividida em tipos especiais de peças como: *motor, chassis, conector*, e outras. A peça motor por sua vez poderia ser subdividida em "*motorTracionador*" e "*motordeSuporte*", e cada uma destas poderia ainda ser subdivida o quanto fosse necessário. Uma instância de *motorTracionadordeVelocidadeVariável* iria herdar todas as características da classe *peça,* assim como as características das classes *motor* e *motortracionador*, como podemos ver na figura 12.7 abaixo.

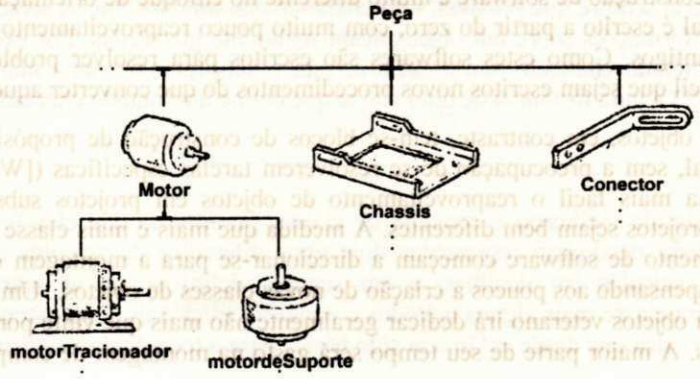


Figura 12.7 Uma hierarquia de classes para peças

É na idéia por detrás das hierarquias de classes que reside a verdadeira genialidade da tecnologia de orientação a objetos [Hol93]. O conhecimento humano é estruturado desta mesma maneira, baseando-se em conceitos genéricos e o seu refinamento em casos cada vez mais especializados. A tecnologia de orientação a objetos utiliza os mesmos mecanismos conceituais que empregamos em nossa vida cotidiana para construir sistemas de software complexos porém altamente compreensíveis.

12.1.5. A Programação com o Uso de Objetos

Como já mencionamos anteriormente, os objetos, as mensagens e as classes são os principais mecanismos para a tecnologia de orientação a objetos. Apesar de a linguagem Simula jamais ter sido largamente aceita como uma linguagem de propósito geral, os conceitos que ela espalhou foram adotados por várias outras linguagens de programação durante os últimos vinte anos, e foram implementados das mais variadas maneiras. Falamos sobre as duas linguagens mais importantes dentro deste contexto: Smalltalk e C++.

Smalltalk ([KA90] [Tay90]) foi desenvolvida no início dos anos setenta nos laboratórios da Xerox em Palo Alto, na California, por um time de pesquisadores liderados por Alan Kay. Smalltalk reflete a estratégia de se projetar uma linguagem totalmente nova que pudesse fazer uso do enfoque de orientação a objetos. C++ ([KA90] [Tay90]) por sua vez foi desenvolvida por Bjarne Stroustrup no início dos anos oitenta, nos laboratórios Bell da AT&T. C++ é uma extensão da popular linguagem C, também da AT&T, e representa a estratégia de se incluir os conceitos do enfoque de orientação a objetos em uma linguagem já existente. O nome estranho, é um jogo de palavras de um programador da linguagem C. O "++" é um operador que incrementa uma variável com mais uma unidade inteira, assim, C++ representa o próximo passo após a linguagem C.

Estas duas linguagens são as mais largamente utilizadas linguagens orientadas a objetos atualmente, e ilustram dois diferentes enfoques para empregar a visão da orientação a objetos. Smalltalk é geralmente reconhecida como uma das mais puristas implementações atuais da metodologia de orientação a objetos, e como tal requer uma estreita aderência a esta metodologia. C++ exemplifica o enfoque híbrido, no qual características de uma linguagem convencional coexistem com as características de orientação a objetos. Diferentemente de Smalltalk, C++ permite consideráveis distâncias de conformidade com relação à metodologia de orientação a objetos, uma qualidade que muitos consideram uma vantagem importante, enquanto outros consideram esta como uma falha lastimável.

12.1.6. O Desenvolvimento de Software com Modelos de Orientação a Objetos

O projeto de um sistema orientado a objetos [Rum91] não se inicia com as tarefas a serem executadas, mas sim com os aspectos do mundo real que precisam ser modelados de forma a permitirem a execução destas tarefas. Desde que estes aspectos sejam representados corretamente, o modelo poderá ser empregado para resolver uma grande variedade de tarefas, inclusive aquela tarefa inicial.

O enfoque de orientação a objetos para a construção de sistemas de software apresenta muitas outras vantagens além da flexibilidade fornecida [Bal89]. Como a estrutura do software irá refletir o mundo real, os programadores podem compreender e modificar com maior facilidade estes softwares, mesmo que eles não sejam as mesmas pessoas que o construíram. Mais importante ainda, as operações básicas de uma companhia tendem a se modificar muito mais devagar que as necessidades de informações requeridas por grupos individuais de pessoas. Isto significa que o software baseado em modelos corporativos terá uma vida mais longa do que aqueles programas escritos para apagar fogueiras (ou seja, para resolver problemas mais imediatos).

O processo de construção de software é muito diferente no enfoque de orientação a objetos. A maior parte do software convencional é escrito a partir do zero, com muito pouco reaproveitamento de procedimentos escritos para programas mais antigos. Como estes softwares são escritos para resolver problemas muito específicos, é geralmente bem mais fácil que sejam escritos novos procedimentos do que converter aqueles já existentes.

Com o uso de objetos, em contraste, tem-se blocos de construção de propósito geral que modelam as entidades do mundo real, sem a preocupação de se resolverem tarefas específicas ([Wag92] [Rum91] [Mez93]). Esta característica torna mais fácil o reaproveitamento de objetos em projetos subseqüentes, mesmo que os objetivos destes novos projetos sejam bem diferentes. À medida que mais e mais classe vão sendo acumuladas, os esforços de desenvolvimento de software começam a direcionar-se para a montagem em novas combinações de objetos já existentes, dispensando aos poucos a criação de novas classes de objetos. Um grupo de desenvolvedores de sistemas orientados a objetos veterano irá dedicar geralmente não mais que vinte porcento de seu tempo para a criação de novas classes. A maior parte de seu tempo será gasto na montagem de componentes já existentes para criar novos sistemas.

É na idéia por detrás das hierarquias de classes que reside a verdadeira genialidade da tecnologia de orientação a objetos [Hol93]. O conhecimento humano é estruturado desta mesma maneira, baseando-se em conceitos genéricos e o seu refinamento em casos cada vez mais especializados. A tecnologia de orientação a objetos utiliza os mesmos mecanismos conceituais que empregamos em nossa vida cotidiana para construir sistemas de software complexos porém altamente compreensíveis.

12.1.5. A Programação com o Uso de Objetos

Como já mencionamos anteriormente, os objetos, as mensagens e as classes são os principais mecanismos para a tecnologia de orientação a objetos. Apesar de a linguagem Simula jamais ter sido largamente aceita como uma linguagem de propósito geral, os conceitos que ela espalhou foram adotados por várias outras linguagens de programação durante os últimos vinte anos, e foram implementados das mais variadas maneiras. Falamos sobre as duas linguagens mais importantes dentro deste contexto: Smalltalk e C++.

Smalltalk ([KA90] [Tay90]) foi desenvolvida no início dos anos setenta nos laboratórios da Xerox em Palo Alto, na California, por um time de pesquisadores liderados por Alan Kay. Smalltalk reflete a estratégia de se projetar uma linguagem totalmente nova que pudesse fazer uso do enfoque de orientação a objetos. C++ ([KA90] [Tay90]) por sua vez foi desenvolvida por Bjarne Stroustrup no início dos anos oitenta, nos laboratórios Bell da AT&T. C++ é uma extensão da popular linguagem C, também da AT&T, e representa a estratégia de se incluir os conceitos do enfoque de orientação a objetos em uma linguagem já existente. O nome estranho, é um jogo de palavras de um programador da linguagem C. O "++" é um operador que incrementa uma variável com mais uma unidade inteira, assim, C++ representa o próximo passo após a linguagem C.

Estas duas linguagens são as mais largamente utilizadas linguagens orientadas a objetos atualmente, e ilustram dois diferentes enfoques para empregar a visão da orientação a objetos. Smalltalk é geralmente reconhecida como uma das mais puristas implementações atuais da metodologia de orientação a objetos, e como tal requer uma estreita aderência a esta metodologia. C++ exemplifica o enfoque híbrido, no qual características de uma linguagem convencional coexistem com as características de orientação a objetos. Diferentemente de Smalltalk, C++ permite consideráveis distâncias de conformidade com relação à metodologia de orientação a objetos, uma qualidade que muitos consideram uma vantagem importante, enquanto outros consideram esta como uma falha lastimável.

12.1.6. O Desenvolvimento de Software com Modelos de Orientação a Objetos

O projeto de um sistema orientado a objetos [Rum91] não se inicia com as tarefas a serem executadas, mas sim com os aspectos do mundo real que precisam ser modelados de forma a permitirem a execução destas tarefas. Desde que estes aspectos sejam representados corretamente, o modelo poderá ser empregado para resolver uma grande variedade de tarefas, inclusive aquela tarefa inicial.

O enfoque de orientação a objetos para a construção de sistemas de software apresenta muitas outras vantagens além da flexibilidade fornecida [Bal89]. Como a estrutura do software irá refletir o mundo real, os programadores podem compreender e modificar com maior facilidade estes softwares, mesmo que eles não sejam as mesmas pessoas que o construíram. Mais importante ainda, as operações básicas de uma companhia tendem a se modificar muito mais devagar que as necessidades de informações requeridas por grupos individuais de pessoas. Isto significa que o software baseado em modelos corporativos terá uma vida mais longa do que aqueles programas escritos para apagar fogueiras (ou seja, para resolver problemas mais imediatos).

O processo de construção de software é muito diferente no enfoque de orientação a objetos. A maior parte do software convencional é escrito a partir do zero, com muito pouco reaproveitamento de procedimentos escritos para programas mais antigos. Como estes softwares são escritos para resolver problemas muito específicos, é geralmente bem mais fácil que sejam escritos novos procedimentos do que converter aqueles já existentes.

Com o uso de objetos, em contraste, tem-se blocos de construção de propósito geral que modelam as entidades do mundo real, sem a preocupação de se resolverem tarefas específicas ([Wag92] [Rum91] [Mez93]). Esta característica torna mais fácil o reaproveitamento de objetos em projetos subseqüentes, mesmo que os objetivos destes novos projetos sejam bem diferentes. À medida que mais e mais classe vão sendo acumuladas, os esforços de desenvolvimento de software começam a direcionar-se para a montagem em novas combinações de objetos já existentes, dispensando aos poucos a criação de novas classes de objetos. Um grupo de desenvolvedores de sistemas orientados a objetos veterano irá dedicar geralmente não mais que vinte porcento de seu tempo para a criação de novas classes. A maior parte de seu tempo será gasto na montagem de componentes já existentes para criar novos sistemas.

Os objetos contidos em objetos compostos podem ser eles mesmos objetos compostos, e este aninhamento pode ser levado adiante por muitos níveis. Os principais componentes de uma aeronave, por exemplo, são todos objetos muito complexos por sí mesmos, como ilustrado na figura 12.9 abaixo.

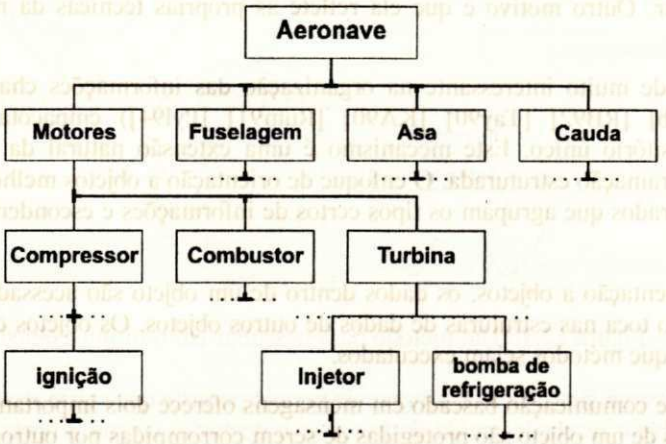


Figura 12.9. Um objeto aeronave com múltiplos níveis

Como os objetos podem ser compostos por outros objetos, as linguagens orientadas a objetos podem representar informações da mesma maneira que pensamos naturalmente sobre elas. Mesmo estruturas complexas, com muitos níveis, podem ser tratadas como um objeto único. Como os objetos complexos possuem seu próprio comportamento, outros objetos podem utilizá-lo sem precisar conhecer os detalhes de sua complexidade interna. Este enfoque torna as coisas mais simples, e pode também tornar as coisas complexas em coisas simples.

12.4. Mensagens e a Ativação de Objetos

Um objeto por mais bem definido que seja é inútil se ficar isolado; o seu valor surge das interações com outros objetos. O veículo para estas interações é a mensagem. Uma mensagem consiste de três partes: o nome do objeto recebedor, o nome de um método que o recebedor sabe como executar, e quaisquer parâmetros que este método requisite para executar as suas funções. Esta terceira parte é opcional, e, se o método não exigir informações adicionais, não são necessários parâmetros para esta mensagem.

As mensagens são normalmente comunicações em dois sentidos. A primeira comunicação é um pedido do emitente de uma mensagem, para um recebedor. O emitente pode ainda requerer uma resposta do recebedor, para se certificar que o mesmo estava apto a executar a ação desejada. Esta resposta é chamada de *valor de retorno*.

A Reutilização de Nomes

Dentro de um programa uma tarefa pode ser efetuada de várias maneiras possíveis. Esta situação nos leva a uma questão que diz respeito ao nome de uma tarefa: pode-se usar o mesmo nome para todas as variações ou deve-se usar um nome diferente para cada uma. Esta é uma questão importante porque um mesmo programa pode conter tantos nomes que fica praticamente impossível controlar todos eles. Mantendo-se os nomes em um programa a um mínimo simplifica o processo de programação e torna os programas mais compreensíveis.

Por exemplo, em um programa de desenho por computador pode-se desenhar várias formas para serem apresentadas na tela. Uma linguagem de programação convencional exigiria que os comandos necessários para desenhar estas formas tivessem nomes únicos, do tipo: *desenhePonto, desenheLinha, desenheRetângulo, desenheCírculo*, etc. Dependendo do número de formas, o programa de desenho por computador poderia requerer muitos nomes diferentes para os nomes de comandos. Em sistemas orientados a objetos cada forma de um desenho seria representada por uma diferente classe. Através de uma técnica chamada **sobrecarga** ([Rum91] [Tay90]) pode-se utilizar o mesmo nome para o método de desenho usado para cada classe. Ao invés de usar *desenhePonto* ou *desenheLinha*, todos os métodos seriam chamados simplesmente *desenhe*. Como cada classe irá conhecer somente sua própria versão para o método desenhe, não existe a possibilidade de se criar confusão dentro do programa, como podemos observar na figura 12.10 .

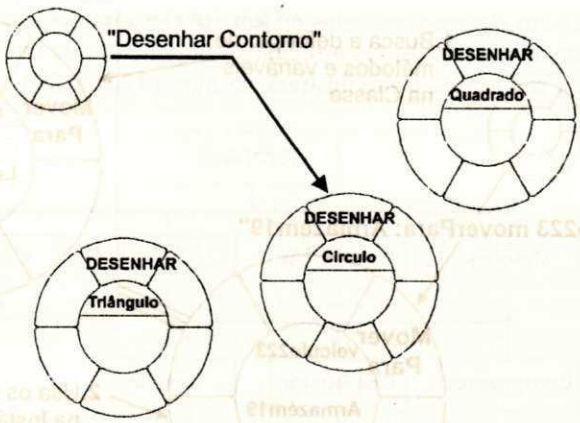


Figura 12.10 Um método para desenhar em múltiplos objetos

A técnica de sobrecarga simplifica os programas porque permite que sejam usados os mesmos nomes para as mesmas operações. O objeto recebedor cuida de executar a operação de sua própria maneira. A sobrecarga de nomes também simplifica a maneira com que são requisitados os serviços por objetos, porque a mesma mensagem pode ser usada para toda uma família de objetos.

Esconder diferentes procedimentos por detrás de um nome comum é uma propriedade dos objetos conhecida como **Polimorfismo** ([Tay90] [Rum91]), um termo grego que significa "muitas formas". Polimorfismo é tão importante que é considerado uma das características que definem o enfoque de orientação a objetos. Os principais benefícios do polimorfismo são o de tornar os objetos mais independentes uns com relação aos outros, e permite que novos objetos sejam adicionados com modificações mínimas para aqueles já existentes. Assim, o polimorfismo é a habilidade de se usar o mesmo nome de um método em mais de uma classe.

12.5. Classes e a Ordenação dos Objetos

Objetos comunicando-se por meio de mensagens formam a essência da tecnologia de orientação a objetos. Estes dois mecanismos sozinhos podem fornecer as bases para uma poderosa linguagem de programação. Adicionalmente, o conceito de classes dá ordem ao enfoque de orientação a objetos tornando-o mais eficiente para representar informações complexas de maneira ordenada e racional.

A função básica das classes é definir um tipo particular de objeto. Desde que se tenha definido uma classe, pode-se criar qualquer número de instâncias únicas para ela. É a classe que define as características partilhadas por todas as instâncias, enquanto as instâncias propriamente ditas irão conter as informações que as tornam únicas. O único aspecto de uma classe que diferencia uma instância de outra são os valores de suas variáveis, de tal forma que uma instância consiste de nada mais que uma seqüência de posições para guardar valores. Quando uma instância recebe uma mensagem, ela procura em sua classe pelos métodos e definições dos tipos de variáveis e em seguida os aplica para seu conjunto único de valores.

Por exemplo, suponha um veículo automatizado que receba a seguinte mensagem:

veículo223 moverPara:armazém19

O objeto veículo223 irá procurar o método moverPara na classe VeículosAutomatizados, e executar este método. O método mover-sePara precisará por sua vez checar a localização corrente para o veículo223. O método irá identificar a variável apropriada dentro da classe, e irá pegar o valor para ela na instância veículo223. A figura 12.11 ilustra o nosso exemplo:

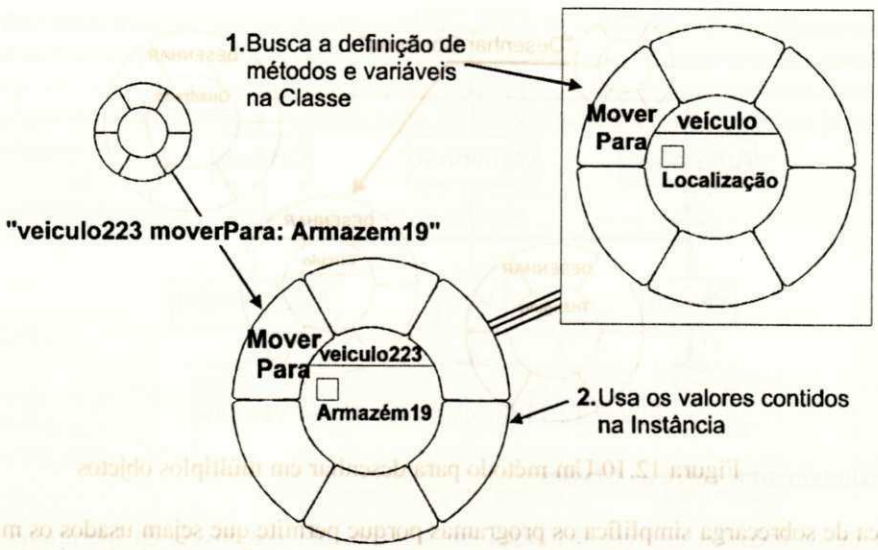


Figura 12.11 Instância e classe para veículo223

Quando um objeto recebe uma mensagem para executar um método que não está definido em sua classe, o objeto irá procurar por este método na classe de hierarquia. Primeiramente, o objeto procura na próxima superclasse de sua própria classe. Se o método não estiver ali definido, o objeto irá procurar na superclasse da superclasse, e assim por diante. Se a definição para o método for encontrada o objeto irá executá-lo. Se o objeto chega ao topo da hierarquia e não encontra o método procurado, ele irá responder com uma mensagem de erro. A figura 12.12 ilustra este conceito.

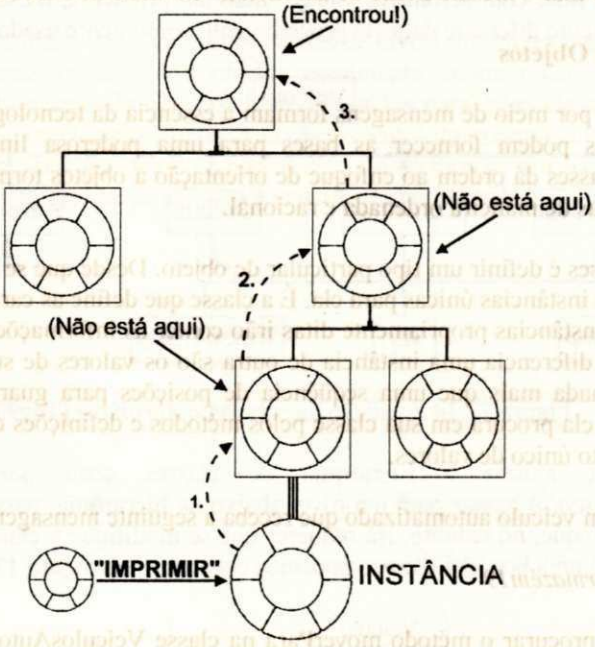


Figura 12.12 A busca através da hierarquia de classes

Os objetos usam estes mesmos conceitos para localizarem as variáveis também. Antes de um objeto poder operar em um de seus valores, ele deve encontrar a definição da variável correspondente. O objeto irá procurar pela variável por toda a hierarquia de classes, e responderá com uma mensagem de erro se a busca falhar.

A hierarquia de classes é um mecanismo muito eficiente porque pode-se usar as definições de métodos e variáveis em mais de uma subclasse sem precisar duplicar as suas definições [Hol93]. Por exemplo, considere um sistema que representa vários tipos de veículos. Este sistema irá conter uma classe genérica para veículos, com subclasses para os tipos especializados. A classe *Veículos* irá conter os métodos e variáveis que foram pertinentes a todos os veículos. As subclasses irão conter métodos e variáveis adicionais que sejam específicas para os casos individuais. A figura 12.13 ilustra o exemplo.

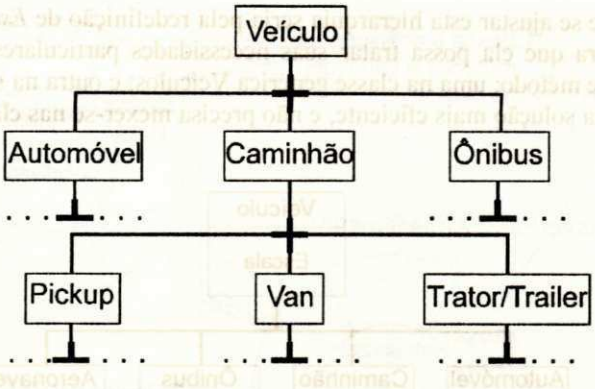


Figura 12.13 Subclasses da classe Veículos

12.5.1. Os Relacionamentos entre as Classes

Apesar de os métodos estarem definidos em somente um dos níveis da hierarquia de classes, e mais precisamente, naquele nível onde ele é mais aplicável, não há nada que proíba um método de ser definido em um nível mais genérico, e também em um nível mais específico. Neste último caso, os objetos deveriam sempre usar a definição mais específica, pois esta seria a primeira a ser encontrada na hierarquia de classes.

A habilidade de se poder definir métodos em mais de um nível ao mesmo tempo pode ser usada para criar exceções para as regras gerais [Rum91]. Suponha por exemplo que exista em uma determinada empresa um sistema com uma classe *Veículos* que inclui um método chamado *EscalonamentodeManutenção* (ou simplesmente, *escala*) usado para estabelecer serviços de manutenção baseados em tempo, quilômetros percorridos, e outros fatores. Esta empresa a posteriori adquire uma aeronave (veja figura 12.14), e uma nova subclasse da classe *Veículos* terá que ser criada para lidar com aeronaves. Surge então um problema; o escalonamento de manutenção para aviões é um processo totalmente diferente daquele procedimento comumente usado para veículos terrestres.

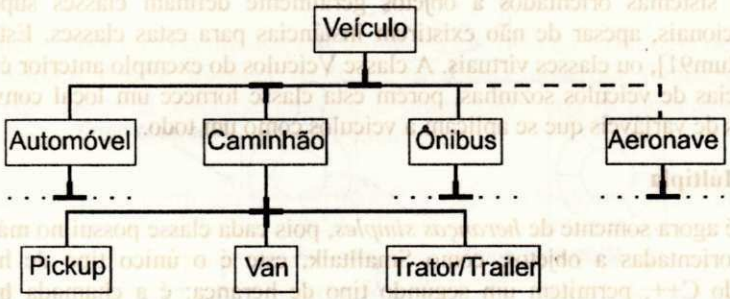


Figura 12.14 A adição da subclasse *Aeronave* [Tay90]

Uma maneira de se ajustar a hierarquia de classes seria por meio do método de *EscalonamentodeManutenção* (*escala*) passar para um nível abaixo na hierarquia, passando para as subclasses de vários tipos de veículos. Este enfoque, no entanto, irá requerer que se modifique a classe Veículos e todas as suas subclasses. Esta certamente não é uma boa idéia, como podemos constatar pela figura 12.15 .

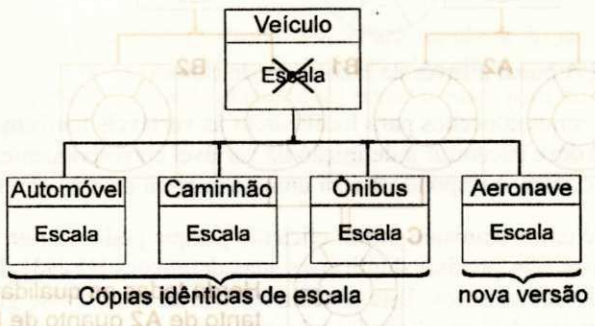


Figura 12.15 O deslocamento de um método para um nível abaixo na hierarquia [Tay90]

Uma melhor maneira de se ajustar esta hierarquia seria pela redefinição de *EscalonamentodeManutenção* (*escala*) na classe *Aeronave* para que ela possa tratar suas necessidades particulares de manutenção. teríamos assim somente duas versões deste método; uma na classe genérica Veículos; e outra na subclasses Avião para tratar a exceção. Esta certamente é uma solução mais eficiente, e não precisa mexer-se nas classes já existentes. A figura 12.16 ilustra o conceito.

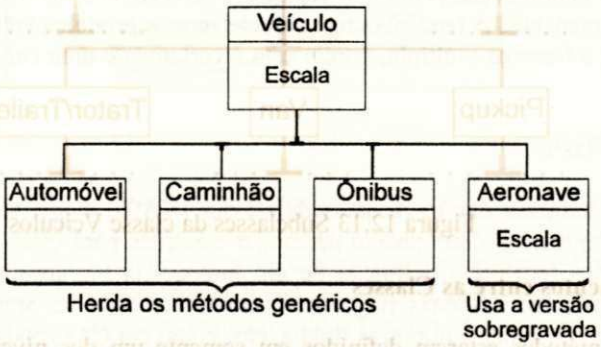


Figura 12.16 A redefinição de um método

Esta técnica para a redefinição de um método em outra subclasse é um outro exemplo de sobrecarga, onde o mesmo nome de um método é usado em duas classes diferentes. No caso particular do nosso exemplo, uma das classes envolvidas é subclasse da outra. neste caso nos deparamos com uma situação de **sobreposição** ([Kay94b] [RB92] [Tay90] [KA90] [Rum91]) porque o método na subclasse sobrepõe o método na classe mais geral. A sobreposição é uma técnica muito importante e permite que casos especiais sejam tratados eficientemente e com muita simplicidade, com impacto mínimo para os objetos envolvidos.

Classes Abstratas

A habilidade de se tratar casos gerais em classes de níveis superiores é muito útil, e é muito comum que os desenvolvedores de sistemas orientados a objetos geralmente definam classes superiores simplesmente com propósitos organizacionais, apesar de não existirem instâncias para estas classes. Esta classes são chamadas de **classes abstratas** [Rum91], ou classes virtuais. A classe Veículos do exemplo anterior é um caso de classe abstrata. Não existem instâncias de veículos sozinhas, porém esta classe fornece um local conveniente para se colocar os métodos e definições de variáveis que se aplicam a veículos como um todo.

Herança Múltipla

Falamos até agora somente de *heranças simples*, pois cada classe possui no máximo uma superclasse. Em muitas linguagens orientadas a objetos, como Smalltalk, este é o único tipo de herança que existe. Outras linguagens, incluindo C++, permitem um segundo tipo de herança: é a chamada **herança múltipla** ([Tay90] [Rum91]) (veja figura 12.17), que permite que um objeto tenha mais de uma superclasse. As heranças múltiplas surgem quando uma classe de objetos precisa desempenhar vários papeis, e cada um destes papeis fica caracterizado pelas outras classes.

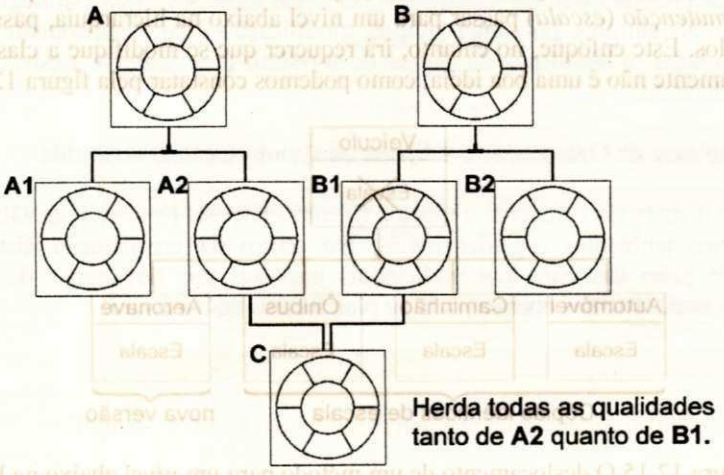


Figura 12.17 Heranças Múltiplas

Apesar do conceito de heranças múltiplas simplificar certas situações, ele também pode levar a complicações. Pode ocorrer por exemplo de uma subclasse herdar duas versões diferentes de um método (qual deles usar?). Outro problema é que ela é geralmente mal usada. Para usar a herança múltipla deve-se ter certeza de que um objeto é verdadeiramente um exemplo de duas ou mais classes.

Devido a estas complicações, muitos estudiosos questionam se os benefícios da herança múltipla compensam seus custos potenciais. A tendência tem sido de vermos produtos voltados para o enfoque de orientação a objetos implementarem a herança múltipla, porém esta é certamente uma característica que deve ser usada com muito critério.

12.5.2. Hierarquias de Classes

A grande funcionalidade das hierarquias de classes [KA90] é que elas aplicam regras gerais a vários grupos de objetos ao mesmo tempo que podem acomodar exceções para estas regras. Devido à forma como um sistema orientado a objetos localiza a definição de um método na hierarquia de classes, a mais especializada definição será sempre aplicada. Algumas linguagens, como é o caso de Smalltalk, são construídas com uma hierarquia geral para todo o sistema. Em outras linguagens, como em C++, podem-se definir várias hierarquias de classes, o que é muito mais conveniente para representar situações do mundo real. A figura 12.18 ilustra o conceito de múltiplas hierarquias de classes.

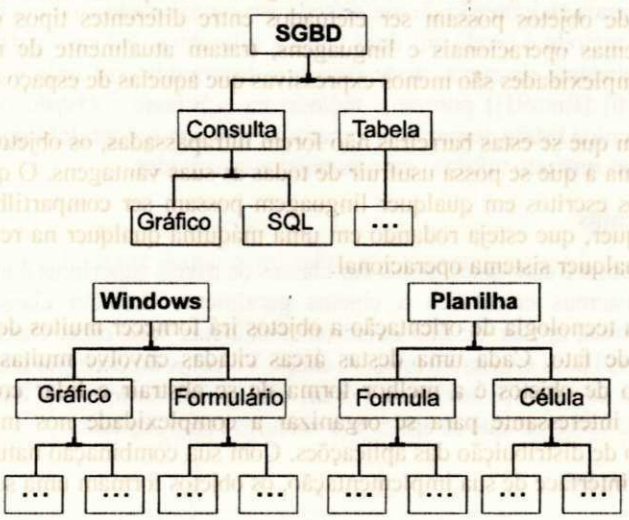


Figura 12.18 Múltiplas hierarquias de classes [KA90]

No projeto das hierarquias de classes algumas considerações são de extrema importância. Primeiramente, as classes devem sempre ser de propósito o mais geral possível, de tal forma que elas possam sempre ser reutilizadas com relativa facilidade. Em segundo lugar, os métodos devem ser definidos nos níveis o mais alto possíveis, mesmo que eles tenham que ser sobrepostos em casos especiais; esta estratégia elimina redundâncias, e torna as modificações mais simples, pois uma simples modificação é automaticamente propagada para todas as subclasses. Em terceiro lugar, a hierarquia de classes deve refletir precisamente a estrutura do sistema do mundo real que ela estiver modelando.

12.6. O Papel da Tecnologia de Orientação a Objetos na Computação Distribuída

A computação distribuída orientada a objetos ([Bal89] [Bet94] [Sol94] [Ude93b]) irá atingir a todos que trabalham nos modernos ambientes organizacionais. No centro da computação distribuída estão os objetos interoperáveis, que vão além das barreiras tradicionais impostas por linguagens de programação, espaço de endereçamento de processos, sistemas operacionais ou plataformas de hardware.

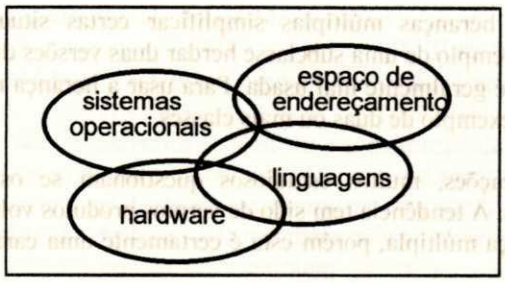


Figura 12.19. As barreiras a serem rompidas pelos objetos interoperáveis [Bet94]

Atualmente, existe um *acoplamento binário forte* entre uma aplicação e suas classes de objetos, isto é, em alguns ambientes de programação orientada a objetos tudo é implementado em uma única linguagem, que roda em um processo único localizado em um único tipo de máquina, a qual roda sob um único tipo de sistema operacional. São esta as barreiras que pretendemos ultrapassar com o uso dos objetos interoperáveis. A primeira barreira que deve ser sobrepassada é a dos espaços de endereçamento, de tal modo que se permita que um processo em determinado espaço de endereçamento possa requisitar os serviços oferecidos por um objeto em outro, ou então que dois outros processos possam compartilhar um objeto em um terceiro espaço de endereçamento. A outra barreira a ser ultrapassada é a do hardware, a qual envolve uma grande complexidade. O que se deseja é que se permita que pedidos por serviços de objetos possam ser efetuados entre diferentes tipos de processadores. As outras duas barreiras citadas, sistemas operacionais e linguagens, tratam atualmente de maneira adequada as questões de distribuição, e suas complexidades são menos expressivas que aquelas de espaço de endereçamento e do hardware.

Observe porém que se estas barreiras não forem ultrapassadas, os objetos interoperáveis dificilmente serão implementados de forma a que se possa usufruir de todas as suas vantagens. O que buscamos é um modelo que irá permitir que os objetos escritos em qualquer linguagem possam ser compartilhados entre aplicações escritas em outra linguagem qualquer, que esteja rodando em uma máquina qualquer na rede (sem importar sequer o tipo do processador), e com qualquer sistema operacional.

Observe que a tecnologia de orientação a objetos irá fornecer muitos dos meios que irão disponibilizar os sistemas distribuídos de fato. Cada uma destas áreas citadas envolve muitas considerações com alto grau de complexidade, e o uso de objetos é a melhor forma de se abstrair e lidar com esta complexidade. Os objetos fornecem uma forma interessante para se organizar a complexidade nos modernos sistemas operacionais, e simplificam o processo de distribuição das aplicações. Com sua combinação natural de dados e comportamento e a separação explícita da interface de sua implementação, os objetos formam uma solução ótima para a distribuição de dados e processos.

12.6.1. O Modelo de Objetos Distribuídos

Podemos vislumbrar um modelo de objetos distribuídos subdividido em três camadas [Bet94]: Modelo de Objetos, Modelo de Componentes e as Aplicações e Sistemas. A figura 12.20 descreve o modelo.

Figura 12.20. O modelo de referência para objetos distribuídos [Bet94]

No nível mais baixo do modelo temos os **Modelos de Objetos**, como por exemplo o ***SOM/DSOM*** *(Distributed System Object Model)* da IBM [Ude93b], *o* ***COM*** *(Component Object Model)* da Microsoft [Bet94] ou o ***CORBA*** *(Common Object Request Broker Architecture)* da OMG (Object Management Group) [Sol94].

Cuidam-se nesta camada de tecnologias a nível de sistemas distribuídos, de tal forma que se solucionem os problemas decorrentes do acoplamento binário forte que existe entre uma aplicação e os objetos nos quais ela se baseia. Por exemplo, vamos considerar uma aplicação escrita em C++ que utiliza várias classes e cujos métodos foram implementados em bibliotecas de ligação dinâmica (DLLs - Dynamic Linking Libraries). Uma DLL não faz parte do código de uma aplicação, e portanto pode ser alterada ou atualizada sem afetar a aplicação, caso a interface permaneça a mesma. Infelizmente, esta funcionalidade não irá se suceder caso as alterações feitas em uma classe baseada em uma DLL alterem o tamanho do objeto. Neste caso, a aplicação chamadora poderá precisar ser recompilada apesar de seu código fonte não ter sido alterado.

Assim, os modelos de objetos são definidos para servir como um meio de ligação entre as implementações de objetos distribuídos. Este modelo é geralmente definido em termos de uma Linguagem de Definição de Interface ou IDL (Interface Definition Language), a qual é processada independentemente da linguagem de implementação usada. O leitor deve estar lembrado do conceito de IDL visto no capítulo 2, quando falamos de RPCs.

A IDL é um meio de definir as interfaces para um serviço, e geralmente gera-se código que seja passível de ser chamado pela aplicação. A IDL fornece a transparência necessária a nível da linguagem de programação, de tal forma que "stubs" possam ser gerados para qualquer linguagem para a qual a IDL possua um mapeamento.

A principal diferença entre a definição de interfaces para procedimentos tradicionais e para aqueles de modelos de objetos é que no modelo de objetos a interface é parte da semântica de construção que representa um objeto. Dependendo do modelo específico, este modelo poderá fornecer algumas ou todas as características e vantagens esperadas dos objetos, incluindo encapsulamento, polimorfismo, e herança. Uma alternativa para o enfoque de IDLs estáticas que existem na maioria dos modelos, é fornecida pela possibilidade de se fazer dinamicamente um pedido de serviço de objetos. Nesta chamada dinâmica, a interface é determinada em tempo de execução, e os dados requisitados são construídos neste momento em uma estrutura especial que será passada para o sistema.

Na segunda camada, vêm os **Modelos de Componentes** de integração, como o OLE da Microsoft, o OpenDoc da Apple, e o OpenStep da Next, ou o CORBA CF do consórcio OMG ([Bet94] [Sol94] [Dew93] [Dew94]). Como mostrado na figura 12.20, estas facilidades baseiam-se nos modelos de objetos da camada inferior, de tal forma que possam implementar funções como a ligação e aninhamento, a ativação local, e outras relacionadas com objetos (*scripting, drag-and-drop*).

Nesta camada as considerações dizem respeito aos projetistas de aplicações e aos usuários. As facilidades fornecidas baseiam-se em um modelo de aplicações baseado em documentos. Neste modelo, uma aplicação especial irá servir como a infra-estrutura básica para apresentar aos usuários vários objetos ou "componentes", e cada um deles é auto contido em termos de seus dados e das ações que podem ser efetuadas sobre estes dados. Tal agrupamento de objetos é normalmente chamado de "documento composto". Estes tipos de documentos são comuns em ambientes multimídia, e envolvem objetos de som, imagem, texto, e gráficos.

Os documentos compostos são perfeitamente adequados para a utilização de objetos compartilhados e distribuídos. Esta tecnologia baseia-se totalmente na definição de objetos, de tal forma que eles ficam descritos em sí mesmos e podem exportar suas interfaces para serem usadas por aplicações distribuídas. Em essência, estes objetos são módulos de ligação dinâmicos. Por exemplo, o OLE 2.0 da Microsoft é um bom exemplo de um modelo de componente usado para integração de aplicações a nível de objetos, permitindo que se exportem interfaces que possam ser ligadas dinamicamente por outras aplicações.

Na camada três, **Aplicações e Sistemas**, ficam os módulos desenvolvidos pelos usuários finais, através das facilidades fornecidas pelas ferramentas disponíveis nas camadas inferiores.

12.7. O Modelo de Objetos CORBA

Um consórcio cujos fins não objetiva o lucro é o grande catalisador para o uso do enfoque de orientação a objetos na computação distribuída. Este consórcio, chamado de **OMG (Object Management Group, Inc.)** ([Sol94] [Dew93] [Dew94] [Bet94] [Pyl94]) está dedicado à busca de soluções para os problemas de interoperabilidade em ambientes distribuídos através do uso de novos enfoques baseados na criação consensual de padrões "de facto" e endossados pela tecnologia de orientação a objetos e seus produtos já comercialmente disponíveis. A sua base inicial consiste de uma suite de linguagens de programação orientadas a objetos já padronizadas, uma série de interfaces e protocolos que em conjunto formam uma especificação de comunicação chamada de **CORBA (Common Object Request Broker Architecture)** ([Sol94] [Dew93] [Dew94] [Bet94] [Pyl94]).

Esta arquitetura é voltada para fornecer interoperabilidade a nível da camada de aplicação entre os sistemas que estiverem executando em plataformas com diferentes sistemas operacionais, linguagens, protocolos de redes, e arquiteturas de hardware. Os planos do OMG incluem o desenvolvimento de serviços e facilidades que possam dar suporte ao programador de aplicações.

O processo aberto de adoção de tecnologias do OMG inclui requisitos por informações e propostas que exigem detalhes técnicos e informações a respeito da disponibilidade comercial de produtos que possam ser recomendados para preencher partes específicas do modelo de referência. As respostas a estes requisitos são enviadas pelos membros do OMG, e em seguida são avaliadas por comitês técnicos e de negócios. O corpo de diretores do OMG toma a decisão final a respeito da adoção de determinada tecnologia. Estas especificações aprovadas ficam à disposição, de graça, para os afiliados ou não do OMG.

As linhas gerais do processo de adoção, que geralmente consome de 16 a 18 meses, são mostradas a seguir:

Requisitos por Informações (RFI - Request for Information): este passo envolve uma pesquisa na indústria de computadores, e estabelece os limites para os softwares comercialmente disponíveis que atendam às necessidades de partes específicas da arquitetura proposta.

Requisitos de Propostas (RFP - Request for Proposals): depois de estabelecidos os limites de soluções comercialmente disponíveis, uma lista de requisitos específicos (não de objetivos) é emitida para estabelecer o software para o qual as especificações já estão prontas para adoção.

Processo de Seleção Competitivo: é baseado em todos os objetivos da arquitetura proposta e na oferta de software disponível no mercado. A seguir o OMG escolhe uma interface única para ser adotada como padrão.

Promulgação da Especificação da Interface: depois de feita a escolha, o OMG irá cuidar de tornar esta especificação o mais difundida quanto possível, e começa ao mesmo tempo os processos de revisão e melhoria para manter a especificação atualizada com a tecnologia e as tendências correntes.

Processos recentes de seleção resultaram em duas importantes tecnologias do OMG: um agente de comunicações (com mensagens) para a camada do ORB, chamada de **CORBA (Common Object Request Broker Architecture)** ([Sol94] [Dew93] [Dew94] [Bet94] [Pyl94]); e um modelo extensível de modelagem de objetos chamado Core Object Model.

12.7.1. O Grupo de Gerenciamento de Objetos OMG

O OMG (Object Management Group, Inc.) foi criado com o objetivo fornecer soluções para o problema de interoperabilidade em ambientes da computação distribuída. O objetivo básico do OMG é a criação de uma suite de linguagens, interfaces e protocolos padronizados para suportar a interoperabilidade em ambientes heterogêneos. Diferentemente de outros grupos que tentaram estabelecer ambientes homogêneos em vários níveis da tecnologia de informação (hardware, sistemas operacionais, interfaces de aplicações), o OMG parte da premissa que os ambientes heterogêneos no mundo da computação ainda existirão por muito tempo (devido à concorrência crescente e às próprias forças de mercado); a suite de interfaces propostas pelo OMG será construída acima de interfaces já existentes, sem querer substitui-las de imediato.

A chave para a interoperabilidade de aplicações baseia-se na tecnologia de orientação a objetos. Esta tecnologia oferece uma grande modularidade e interfaces claramente definidas, assim como permite um enfoque rigoroso na reutilização de software já existente (devido ao mecanismo de *herança*).

Como já destacamos, o enfoque de orientação a objetos é a escolha perfeita para a definição de serviços em ambientes da computação distribuída. Adicionalmente, o OMG baseia-se totalmente em tecnologias que estejam comercialmente disponíveis, gerando um esforço para a criação de um mercado largamente disponível, ofertando produtos de "prateleira". Estes dois conceitos, o de uso da tecnologia de orientação a objetos e o de um grande número de produtos comercialmente disponíveis formam a fundação do processo para o OMG.

12.7.2. A Arquitetura de Gerenciamento de Objetos

O OMG está empregando um enfoque do tipo "de-baixo-para-cima" ("bottom-up") para a criação de padrões *de facto*, começando pelo ponto de vista da rede e seguindo acima para as interfaces de aplicações. O primeiro passo para a adoção de padrões pelo OMG surgiu de um acordo em terminologias e arquiteturas. Este acordo sobre as direções, estrutura e objetivos foi efetivado em meados de 1990, e publicado a seguir na forma de um guia de referência chamado de **OMAG (Object Management Architecture Guide)** [Sol94]. Um modelo de referência, chamado de **OMA (Object Management Architecture)** ([Sol94] [Bet94]) foi criado para estabelcer as bases da seleção de tecnologia a ser adotada. A figura 12.21 abaixo apresenta este modelo..

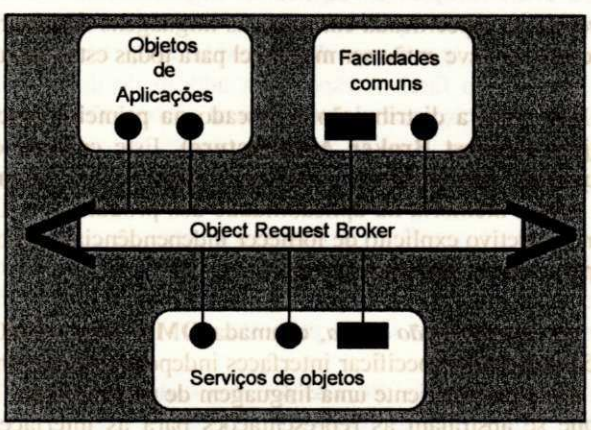


Figura 12.21. A arquitetura para o gerenciamento de objetos [Sol94]

Foram definidos juntamente com o modelo de referência OMA quatro áreas de padronização:

Um servidor de requisitos por objetos chamado **ORB** *(Object Request Broker)*: que constitui o elemento de comunicação chave, e cuida da distribuição de mensagens entre objetos de aplicações de uma forma altamente interoperável. Constitui-se como o modelo lógico e o meio para armazenamento físico dos objetos.

Um modelo de objetos: que é um modelo de abstração para projeto único de aplicações portáveis, e permite a comunicação com sistemas orientados a objetos que estejam de acordo com as especificações do OMG.

Os serviços de objetos: que fornecem as principais funções para possibilitar as funcionalidades básicas dos objetos, por meio do ORB.

As facilidades comuns: que agrupam as facilidades que são úteis para muitos domínios de aplicações e que serão disponibilizadas pelas interfaces que forem aderentes às classes da OMA.

É importante que se diga que os nomes de serviços de objetos, de facilidades comuns e de objetos de aplicações são simplesmente categorias de objetos. No modelo OMA, cada parte de software é representada como um objeto, que se comunica com outros objetos por meio do ORB.

Uma das novas áreas do CORBA são as facilidades da segunda camada do modelo de referência mostrado na figura 12.20 anterior, que incluem os novos esforços do OMG baseado no componente **CORBA-CF** (*Common Facilities*) [Bet94]. O CF apresenta um foco orientado para o nível de aplicação, e define objetos que fornecem funções chaves para a implementação de suporte aos grupos de trabalho: impressão, correio-eletrônico, consultas a bases de dados, *newsgroups* e objetos compostos (que incluem facilidades multimedia, com som e imagens de alta qualidade). Espera-se que esta venha a ser a camada mais usada por desenvolvedores interessados no desenvolvimento de aplicações para os ambientes distribuídos.

12.7.3. Uma Visão Geral do CORBA

O mais comum enfoque para a resolução de problemas de integração de comunicações em um ambiente de computação distribuída é o uso **RPCs (Remote Procedure Calls)** [OSF91e], que simplificam a tarefa de reintegração de um programa que tenha sido distribuído por vários processadores. O enfoque típico de uso de RPCs envolve a escolha de uma área de abrangência para as chamadas a procedimentos, através da qual o programa será distribuído (manualmente), especificando-se em seguida a interface entre o programa chamador e a rotina a ser chamada (que será distribuída). Uma seqüência de compilação semi-automática insere a seguir o código extra necessário para a execução remota da chamada.

Este procedimento abstrai claramente os detalhes do canal de comunicação estabelecido entre os dois processos, porém apresenta algumas desvantagens:

- as necessidades e detalhes para que se possa executar a RPC devem ser conhecidos na hora da compilação;

- os detalhes das estruturas de dados (representação, extensão, etc) para os parâmetros passados dos processos chamadores para os processos chamados devem também ser conhecidos em tempo de compilação;

- a interface deve ser ou especificada em todas as linguagens de programação a partir da qual o processo chamado deverá ser invocado, ou deve então ser mapeável para todas estas linguagens.

O enfoque do OMG para a distribuição é baseado na primeira especificação de tecnologia adotada, o **CORBA (Common Object Request Broker Architecture)**. Este enfoque emprega uma abstração similar às RPCs, com algumas mudanças que podem simplificar a vida de um programador (e assim gerar custos de manutenção mais baixos e uma melhoria na aplicabilidade dos produtos). O CORBA especifica os ítens abaixos, que foram projetados com o objetivo explícito de fornecer independência para os componentes como linguagens de programação, sistemas operacionais, hardware e redes.

Uma *linguagem de especificação única*, chamada **OMG IDL (OMG Interface Definition Language)** [Bil94]: esta linguagem é usada para especificar interfaces independentes de linguagens de programação que serão usadas na execução. IDL não é propriamente uma linguagem de programação. Ela fornece um enfoque orientado a objetos, o que permite que se abstraiam as representações para as interfaces (encapsulamento), fornece ainda polimorfismo de mensagens, e herança de interfaces.

Uma *representação dinâmica total* para as interfaces disponíveis, por meio de um **repositório de interfaces (RI)** [Bil94] que irá representar as interfaces (ou classes) de todos os objetos disponíveis no ambiente da computação distribuída.

Uma *extensão de chamadas dinâmicas total* para permitir a descoberta de objetos em tempo de execução do programa de aplicação a partir de um RI, permitindo a construção de mensagens e o envio destas mensagens.

Uma *extensão de contextos* para permitir a passagem de parâmetros de nomes opcionais. Usado para o controle explícito das funções fornecidas.

Um *mecanismo de abstração* (chamado de **adaptador de objetos**) para esconder detalhes da implementação de objetos.

12.7.3.1. Detalhes do CORBA

O serviço básico fornecido pelo CORBA é a entrega de mensagens de um processo para outro, e a entrega de uma resposta para o processo chamador. Estes dois objetos nas transações que se comunicam são o cliente e o servidor, apesar de na estrutura do CORBA qualquer objeto poder ser tanto um cliente quanto um servidor ou ambos. De fato, espera-se que a maior parte dos objetos (isto é, as aplicações) venham a ser tanto clientes quanto servidores enquanto existirem.

Os sistemas aderentes ao CORBA podem estabelecer o transporte para as comunicações entre os objetos de dois esquemas totalmente diferentes, apesar de compatíveis:

O primeiro esquema chama-se *esquema dinâmico*: neste caso o cliente pode descobrir servidores disponíveis em tempo de execução (baseado por exemplo na localização e facilidades fornecida pelo servidor), descobrir quais são as interfaces para estes servidores, construir um pedido para uso do serviço, enviar este pedido (múltiplas vezes, opcionalmente), e recuperar as respostas. Isto é tudo possível sem a necessidade de nenhum conhecimento prévio, em tempo de compilação, sobre os servidores ou interfaces disponíveis.

O segundo esquema chama-se *esquema estático*: neste caso o cliente deve conhecer a interface do servidor a ser acessado em tempo de compilação, apesar de se poder determinar a disponibilidade ou não de servidores em tempo de execução.

Afinal por que o CORBA estabelece estes dois enfoques? A resposta é simples. Existem razões técnicas para o uso do enfoque estático algumas vezes, apesar do esquema dinâmico ser consideravelmente mais eficiente para algumas aplicações. Parece ser óbvio que, pelo menos em teoria, um esquema estático leve a melhores desempenhos, pois como todos os detalhes da linguagem de implementação, sistema operacional, e canais de comunicações são todos conhecidos no momento da compilação, várias técnicas de otimização podem diminuir o overhead da execução. No entanto, existem aplicações importantes para as quais os esquemas dinâmicos são imprescindíveis.

A solução adotada pelo CORBA para fornecer mecanismo de troca de mensagens dinâmicos ou estáticos só é possível graças à linguagem IDL. A mesma linguagem fonte usada para compilar os "stubs" e esqueletos (como nas RPCs) torna-se disponível para os clientes em tempo de execução para auxiliar estes clientes na descoberta de interfaces e na construção dos pedidos. A figura 12.22 ilustra estes conceitos.

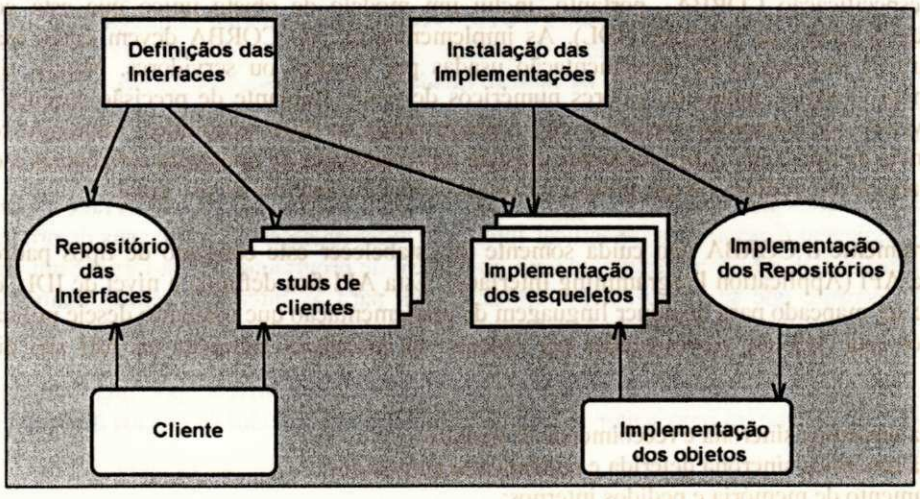


Figura 12.22 A linguagem IDL - Interface definition Language [OSF91e]

Esta possibilidade de escolha dentre duas interfaces, estáticas ou dinâmicas, permite que os programadores de software cliente ou servidor tenham total liberdade para escolherem um dos dois esquemas, baseados nos pedidos da aplicação a ser desenvolvida. Melhor ainda, permite que o software de aplicação utilize ambos os esquemas em um mesmo programa de aplicação. Sempre que o programador conhecer totalmente a interface em tempo de compilação, pode-se usar a interface estática; nos outros casos, pode-se usar a interface dinâmica.

Uma extensão importante para o núcleo do CORBA chama-se **adaptador de objetos**, o qual faz adaptações de código genérico de objetos do modelo ORB para as particularidades da implementação de objetos em vários esquemas diferentes. Por exemplo, os objetos podem ser implementados como programas de aplicações dentro dos próprios processos do sistema operacional; ou então podem ser representados como objetos em SGBDs orientados a objetos; ou podem ainda representar objetos a nível de linguagens de programação (por exemplo, em C++ ou Smalltalk). A figura 12.23 ilustra este exemplo.

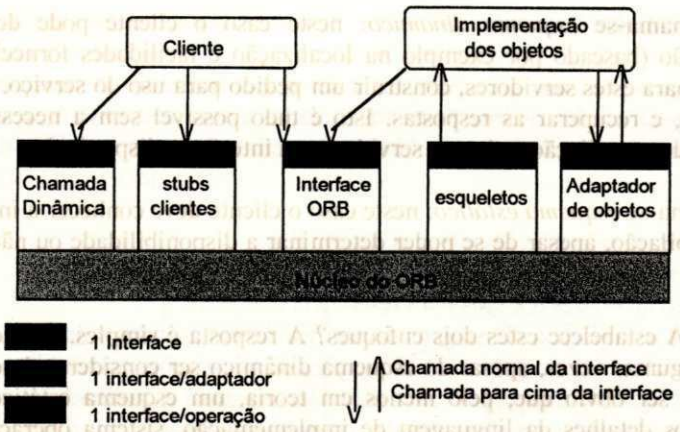


Figura 12.23 Chamadas de interfaces ORB [Sol94]

De fato, uma das maiores dificuldades para os padrões CORBA foram as muitas maneiras diferentes como os vendedores concorrentes enxergavam a implementação dos objetos. Todos que usam objetos possuem uma idéia diferente do que um objeto realmente representa, qual o tamanho de um objeto (em bytes), e com que frequência os objetos têm que interagir. Por exemplo, os objetos na linguagem C++ têm pouco em comum com objetos gerenciados no seu sentido comum, exceto por algumas similaridades de abstração.

A especificação CORBA portanto, inclui um modelo de objeto único que está contido na própria linguagem de definição da interface (IDL). As implementações do CORBA devem então mapear este modelo genérico para as linguagens de implementação usadas por clientes ou servidores. Muitos tipos de dados são fornecidos pelo CORBA, incluindo valores numéricos de ponto flutuante de precisão simples ou dupla; valores booleanos; vetores de tamanhos variáveis; etc. Existem ainda os tipos estendidos, incluindo registros; união de tipos; seqüência de tipos, etc. Adicionalmente existem representações de tipos para referências a objetos que podem ser usadas para enviar pedidos ou que podem ser usadas como parâmetros de uso geral.

Certamente o CORBA não cuida somente de estabelecer este conjunto de tipos padronizados, e inclui também uma API (Application Programming Interface). Esta API fica definida a nível de IDL, de tal forma que o seu uso pode ser mapeado para qualquer linguagem de implementação que o usuário deseje utilizar. As construções acessadas por esta API são representadas por objetos. As interfaces incluídas na API são divididas em cinco categorias:

- chamada dinâmica síncrona e recebimento de pedidos;
- chamada dinâmica síncrona deferida e recebimento de pedidos;
- gerenciamento de memória e pedidos internos;
- manutenção de lista de contextos;
- interface de pedidos de busca em repositórios.

12.7.4. Tendências do CORBA

Como em qualquer tecnologia, não se pode considerar a interface CORBA já terminada. Em particular, apesar dos documentos do CORBA definirem com bastante consistência detalhes da linguagem IDL e da API, o que permite a criação de códigos de aplicação aderentes ao ORB, a versão CORBA 1.1 não aborda as considerações de interoperabilidade que comumente surgem nas grandes redes de computadores do mundo real. Para retificar esta falta, o OMG liberou no final de 1994 uma revisão compatível chamada CORBA 2.0 que cuida primeiramente dos padrões de comunicações ORB-para-ORB, de tal forma que duas implementações aderentes ao ORB, mesmo que sejam desenvolvidas separadamente, tenham a garantia de interoperar e partilhar os mesmos espaços de nomes.

Atualmente, muitas empresas de grande relevância para a indústria de Informática, incluindo a IBM, a Hewlett-Packard, Olivetti, NEC, Sun Microsystems e muitas outras, e também vários consórcios da indústria, incluindo a OSF, International Multimedia Association, X/Open, etc., têm expressado seu endosso ao padrão CORBA, e estão incluindo implementações da API CORBA em seus produtos [Bet94].

12.7.5. Serviços de Objetos e Facilidades Comuns

Com o surgimento de muitos produtos que começam a ser disponibilizados comercialmente, surge uma necessidade imediata de que as especificações de interfaces em camadas acima do CORBA sejam definidas. Para definir estes serviços em outras camadas, o OMG está cuidando de definir serviços baseados na linguagem IDL para suportar:

- serviços de ciclo de vida de objetos: criação, remoção e coleta de lixo;
- serviços persistentes: armazenamento em banco de dados do estado dos objetos;
- serviços de nomes: mapeamento de nomes textuais para referência a objetos;
- serviços de notificação de eventos: suporte a mensagens de notificação em grandes volumes.
- serviços de relacionamento/associação: para gerenciar as ligações de associações entre objetos;
- serviços de controle de concorrência de transações: para controlar commits e "rollback" a partir de uma série de mensagens CORBA, a serem utilizadas em situações de concorrência nos sistemas distribuídos;
- serviços de tempo: para que sejam estabelecidas concordâncias em valores de tempo globais para a sincronização dos ambientes distribuídos;
- serviços de internacionalização: para gerenciar importações e exportações de estados de objetos.

O OMG busca ativamente parcerias com grupos externos que estão desenvolvendo interfaces padronizadas ou implementando alguns dos serviços necessários para a computação distribuída, incluindo serviços de baixo nível (como interfaces de autenticação/segurança) e de alto nível (como correio-eletrônico). Sempre que possível o OMG busca acatar trabalhos existentes, de tal forma que se ganhe em portabilidade, interoperabilidade e reutilização de objetos. Aos poucos, e à medida que os níveis das interfaces forem melhorando, os esforços do OMG ficarão vez mais evidentes.

12.8. O Modelo de Objetos da IBM

O modelo de objetos *System Object Model* (**SOM**) ([Ude93b] [Bet94]) da IBM é um padrão binário para objetos que são neutros com relação aos sistemas operacionais e às linguagens de programação, e cujas interfaces estão de acordo com as definições do CORBA expressas por sua IDL. A versão distribuída do SOM, chamada **DSOM** (*Distributed System Object Model*) ([Ude93b] [Bet94]) compõe uma infra-estrutura aderente ao ORB do CORBA. No entanto, o SOM trata também da implementação de objetos, o que o afasta um pouco da especificação CORBA, a qual define interfaces de objetos sem considerar detalhes de implementações.

O SOM não define uma tecnologia distribuída. O DSOM serve para este propósito. O SOM foi projetado principalmente para resolver problemas de acoplamento binário forte (ou seja, de difícil migração entre plataformas) entre as aplicações e as bibliotecas de classes fornecidas. Para permitir que isto se tornasse possível, o SOM baseia-se em interfaces definidas em uma versão estendida da IDL do CORBA, fornecendo também módulos *"run-time"* para suportar seus objetos.

A tecnologia SOM/DSOM é completa, e está disponível atualmente para três plataformas: AIX, OS/2 e Windows 3.1. O SOMobjects Toolkit, que é o nome comercial do produto, e inclui várias funcionalidades como as classes SOM, que fornecem facilidades de alto-nível para desenvolvedores de aplicações. Dentre estas facilidades podemos citar: uma interface de repositórios aderente ao CORBA; uma infra-estrutura persistente para arquivar objetos entre as sessões run-time; uma facilidade de replicação, que permite que objetos sejam espelhados por múltiplos espaços de endereçamento (com a garantia de sincronização, tolerância a falhas e consistência entre as cópias). O kit inclui também uma coleção de classes, metaclasses de utilitários, e classes para gerenciamento de eventos.

12.8.1. O Modelo de Objetos SOM

O modelo de objetos SOM é um modelo clássico, onde as classes definem as características dos objetos e os métodos identificam um único objeto onde ele deve ser executado. Todos os objetos derivam-se da classe básica *SOMobject*, que fornece métodos para suporte a módulos "run-time" comuns a todos os objetos no sistema.

Como em outras linguagens orientadas a objetos "puras", as classes SOM constituem-se elas mesmas em objetos, chamados de instâncias da *metaclasse* SOM. Uma metaclasse é um tipo de classe. Enquanto uma classe descreve um conjunto de instâncias de objetos, uma metaclasse descreve um conjunto de classes. As metaclasses definem funções que operam na classe como um todo, incluindo os métodos que são executados quando a instância de uma classe é criada.

Para substituir a implementação do mecanismo de herança, a Microsoft oferece um modelo diferente de reutilização de código chamado de "agregação", que permite que objetos sejam construídos a partir de sub-objetos, e de tal forma que o objeto composto (ou agregado) poderá referenciar diretamente os sub-objetos.

Mas afinal, será que a agregação substitui completamente a herança? Na verdade não. A herança em linguagens OO é um mecanismo intrínseco da linguagem, enquanto que a agregação é uma alternativa que permite várias formas de implementação. A herança geralmente requer pouco ou mesmo nenhum suporte de código extra, enquanto que a agregação deve ser completamente suportada pelo próprio programador, o que certamente é uma grande desvantagem.

Voltando para a figura 12.20 anterior, onde apresentamos o modelo de referência para plataformas de objetos interoperáveis, podemos ver que o modelo de objetos da Microsoft é o Componente Object Model (COM) e que o seu componente mais importante é o OLE, que representam as camadas 1 e 2 do modelo respectivamente. A seguir falamos um pouco sobre eles.

12.9.1. O Modelo de Objetos COM

A Microsoft baseia seu modelo de objetos no *Component Object Model* (**COM**) [Bet94], que executa algumas das mesmas tarefas de um ORB, só que em escala diferente, e com o uso de diferentes técnicas. O COM baseia-se em objetos de janelas, que definem entidades funcionais que obedecem aos princípios OO de encapsulamento. Os clientes no modelo não manipulam diretamente estes objetos, é o próprio objeto que expõe para os seus clientes vários apontadores de conjuntos de funções chamados de "*interfaces*". Uma interface é um apontador efetivo para uma tabela de apontadores de funções. A figura 12.25 seguir mostra o relacionamento entre uma tabela de interfaces e a implementação do objeto.

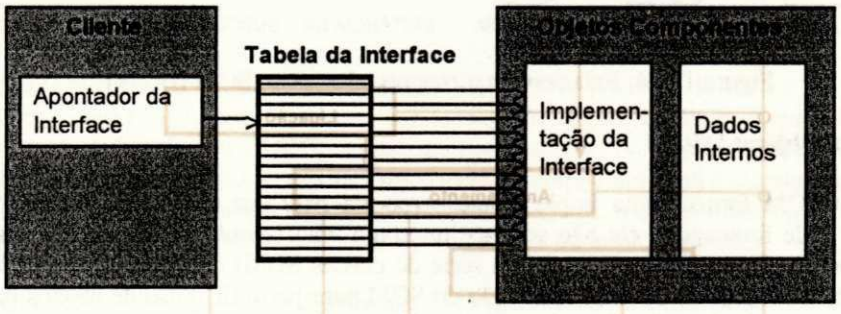


Figura 12.25. O relacionamento entre clientes, objetos componentes e interface no COM [Bet94]

Todos os objetos de janela devem suportar a interface mais básica, *IUnknown*, que suporta três métodos para disponibilizar a funcionalidade básica de todos os objetos de janelas. Estes métodos são: QueryInterface, que permite que um cliente pergunte que interfaces são suportadas por um determinado objeto, e AddRef e Release, que gerenciam contadores de referência para objetos.

Contadores de referência são um mecanismo familiar para os programadores de linguagens OO, com os quais o sistema pode acompanhar como os clientes estão usando um determinado ponteiro que aponta para uma ou mais das interfaces de objetos fornecidas. Quando um contador de referência chega a zero, o sistema pode remover o objeto, e recuperar seus recursos. A Microsoft especificou 60 ou mais destas interfaces na arquitetura OLE 2.0, incluindo interfaces para ativação local, ligação, e aninhamento, as quais compõem o núcleo da tecnologia OLE. Existem ainda interfaces tipo "drag-and-drop", transferência uniforme de dados, automação, arquivos compostos, e outras bastante úteis.

12.9.2. O Componente OLE

O OLE é um componente do nosso modelo de referência (vide figura 12.19 anterior) voltado para fornecer uma série de funcionalidades OO para que usuários finais possam utilizar objetos partilhados e distribuídos. A fundação para o OLE é o modelo COM (que é atualmente voltado para suportar somente este componente, diferentemente do CORBA).

Para entendermos um pouco mais a respeito do OLE, falamos um pouco sobre o antigo modelo de processamento interativo da Microsoft, conhecido por **DDE** (*Dynamic Data Exchange*) [Dew94]: que é um protocolo de difusão onde uma aplicação pode estabelecer um canal de comunicação entre um servidor DDE, localizado na máquina onde a aplicação estiver rodando. O DDE é um protocolo assíncrono, isto é, desde que a comunicação seja estabelecida, o cliente pode emitir um pedido, porém fica esperando em um loop pelo retorno dos resultados. Este mecanismo é mais complicado que uma função de chamada síncrona, devido à possibilidade de que ocorram falhas de comunicações, estouros de tempo, e outros erros que precisam ser detectados e recuperados. Muitos desenvolvedores consideram o DDE frustrante, e passível de falhas. Estes são os motivos de sua pouca popularidade.

A versão 1.0 do OLE foi projetada para fornecer um mecanismo de ligação e aninhamento para objetos onde anteriormente usava-se o DDE como o mecanismo principal de comunicação. Assim, a versão OLE 1.0 herdou muitos dos problemas associados com um protocolo de difusão assíncrono.

A versão OLE 2.0 melhora a versão 1.0 pela definição de muitos serviços de sistema adicionais às funcionalidades de ligação e aninhamento fornecidas. Dentre estes serviços estão a transferência uniforme de dados, um serviço de automação para o OLE cuja função é permitir que aplicações possam disponibilizar APIs para uso em outras linguagens, e fornecer armazenamento persistente para hierarquias de objetos aninhados, o que constitui um serviço de armazenamento estruturado. A mudança mais importante no OLE 1.0, no entanto, é o total abandono do protocolo DDE, substituindo-o pelo Component Object Model.

O relacionamento entre o OLE 2.0 e o COM é mostrado na figura 12.26 . É bom lembrarmos que o COM não é uma tecnologia distribuída, o que o coloca aquém das soluções oferecidas em modelos como o da IBM ou o CORBA vistos anteriormente.

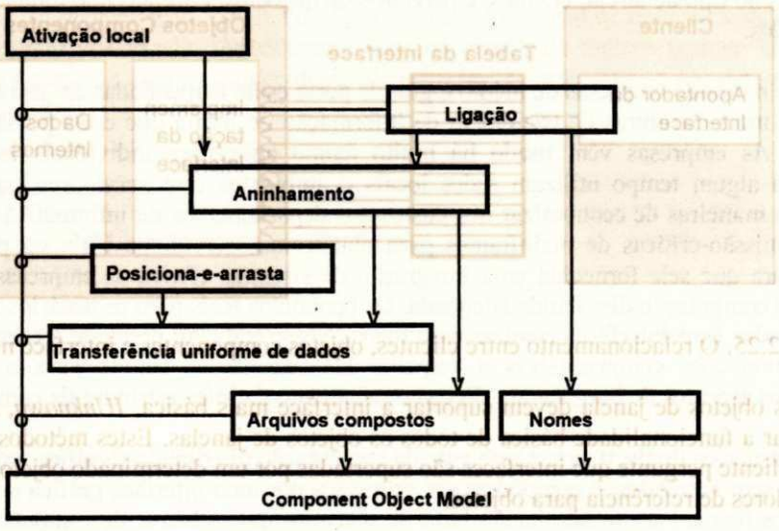


Figura 12.26. As interfaces OLE 2.0 e o COM [Bet94]

Para entendermos um pouco mais a respeito do OLE, falamos um pouco sobre o antigo modelo de processamento interativo da Microsoft, conhecido por **DDE** (*Dynamic Data Exchange*) [Dew94]: que é um protocolo de difusão onde uma aplicação pode estabelecer um canal de comunicação entre um servidor DDE, localizado na máquina onde a aplicação estiver rodando. O DDE é um protocolo assíncrono, isto é, desde que a comunicação seja estabelecida, o cliente pode emitir um pedido, porém fica esperando em um loop pelo retorno dos resultados. Este mecanismo é mais complicado que uma função de chamada síncrona, devido à possibilidade de que ocorram falhas de comunicações, estouros de tempo, e outros erros que precisam ser detectados e recuperados. Muitos desenvolvedores consideram o DDE frustrante, e passível de falhas. Estes são os motivos de sua pouca popularidade.

A versão 1.0 do OLE foi projetada para fornecer um mecanismo de ligação e aninhamento para objetos onde anteriormente usava-se o DDE como o mecanismo principal de comunicação. Assim, a versão OLE 1.0 herdou muitos dos problemas associados com um protocolo de difusão assíncrono.

A versão OLE 2.0 melhora a versão 1.0 pela definição de muitos serviços de sistema adicionais às funcionalidades de ligação e aninhamento fornecidas. Dentre estes serviços estão a transferência uniforme de dados, um serviço de automação para o OLE cuja função é permitir que aplicações possam disponibilizar APIs para uso em outras linguagens, e fornecer armazenamento persistente para hierarquias de objetos aninhados, o que constitui um serviço de armazenamento estruturado. A mudança mais importante no OLE 1.0, no entanto, é o total abandono do protocolo DDE, substituindo-o pelo Component Object Model.

O relacionamento entre o OLE 2.0 e o COM é mostrado na figura 12.26 . É bom lembrarmos que o COM não é uma tecnologia distribuída, o que o coloca aquém das soluções oferecidas em modelos como o da IBM ou o CORBA vistos anteriormente.

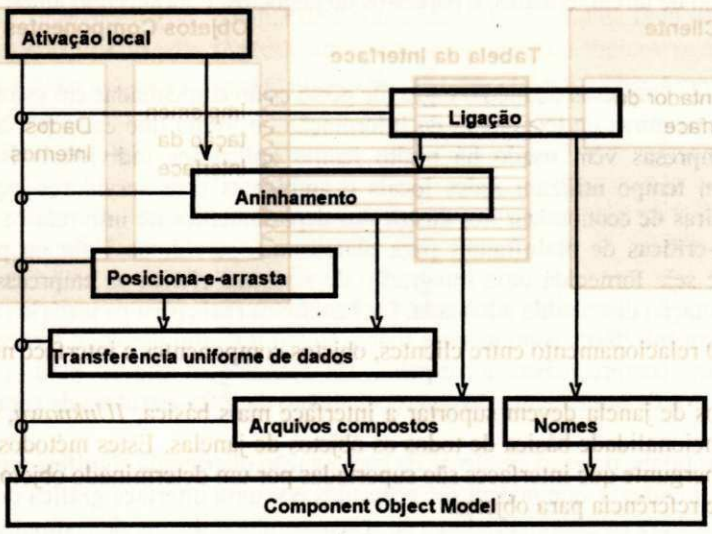


Figura 12.26. As interfaces OLE 2.0 e o COM [Bet94]

Esperamos que com nosso trabalho possamos enriquecer os leitores interessados com uma visão geral das tecnologias da computação distribuída. Nele, descrevemos desde as experiências com os primeiros sistemas operacionais distribuídos, passando por todos os conceitos relevantes e apresentando as estratégias de implementação e gerenciamento que irão beneficiar todos aqueles interessados em conhecer com mais detalhes os ambientes da computação distribuída.

Nosso trabalho pretende ser útil a todos os profissionais de Informática que estejam envolvidos com o projeto, implementação ou gerenciamento dos ambientes da computação distribuída. Alunos de cursos de Ciências da Computação em universidades de todo o Brasil serão largamente beneficiados com a leitura e o estudo de nosso trabalho, pois sairão para um mercado de trabalho cheio de novidades, as quais muitas vezes não são apresentadas de maneira integrada durante o curso. Os alunos de cadeiras como Redes de Computadores, ou de Sistemas Distribuídos podem utilizar nosso livro para compreenderem melhor os assuntos relacionados com a implementação e o gerenciamento de todo um ambiente de computação distribuída.

O grande diferencial para o nosso trabalho baseia-se na apresentação de forma coesa, didática e integrada de uma série de conceitos sobre tecnologias tão diversas quanto o Gerenciamento Integrado da Rede ou o Uso de Objetos Distribuídos. Nossa perspectiva é de sermos úteis para leitores que tenham que suportar um grande número de usuários em um ambiente de computação distribuída autêntico, e não para leitores com uma visão menor do todo, como aqueles envolvidos com pequenas redes locais departamentais.

É bom que enfatizemos o aspecto da originalidade do nosso trabalho. Escrevemos este livro com o mais criterioso trabalho de pesquisa baseado em uma grande disponibilidade de fontes de referências bibliográficas, todas recentes, publicadas em sua grande maioria a partir de 1993. Nosso trabalho estendeu-se por mais de dois anos, e incluímos como parte de nosso esforço a oportunidade de termos aproveitado a aplicação de nossos conhecimento de maneira prática, tendo atendido a consultorias especializadas para grandes empresas envolvidas com as questões de projeto, implementação, escolha de ferramentas e migração de uma série de aplicações para o novo ambiente da computação distribuída. Consideramos que os principais tópicos tenham sido escolhidos de maneira a suprirem a maior parte de todo o vasto conteúdo de conceitos relacionados, e são portanto pontos chaves para o sucesso de nosso trabalho.

Bibliografia

[ACM92a] ACM. Debating Encryption Standards. *Communications of the ACM.* 35(7):32-35, july 1992.

[ACM92b] ACM. The Digital Signature Standards Proposed by NIST. *Communications of the ACM.* 35(7):36-40, july 1992.

[ACV92] F.Antonelli,M.Ciardi,and M.Volpe. A Public Telecomm Operator's Point of View for the Integrated Management of Local and Wide Area Networks. In *Proceedings Toward a New World in Computer Communications,* pages 357-362, Rome, Italy, 1992. SIP Research and Development. [106].

[And92] Ron Anderson. SQL Databases: High Powered, High Priced. *PC Magazine,*11(15):369-397, September 1992. [381b].

[And93] Dave Andrews. RAID Down to the Desktop. *BYTE,*18(8):28, July 1993. [293a].

[AP94a] S.Aidarous and T.Plevyak. *Principles of Network Management,* chapter 1, pages 1-18.Volume 1 of Aidarous and Plevyak [AP94b], 1994. [152].

[AP94b] Salah Aidarous and Thomas Plevyak,editors. *Telecommunications Network Management Into the 2lst Century: Techniques, Standards, Technologies and Applications, V*olume 1. IEEE Press, New York, NY, 1994. [151].

[Bak92] Steven Baker. Networth: Defense to the Rescue. *Unix Review,*10(3):13-22, March 1992. [094].

[Bal89] H.E.Bal. *The Shared Data-Object Model as a Paradigm for Programming Distributed Systems.* PhD thesis, Vrieje University,Amsterdan,The Netherlands, 1989. [162].

[Bal92] Howard Baldwin.The Wizard of Auspex. *UnixWorld,*9(7):G1-64,July 1992. [250a].

[Bar85] R Baron et al. Mach-l: An Operating Environment for Large Scale Multiprocessor Applications. *IEEE Software,* 2(7):65-67, July 1985. [204+].

[Bar93a] Nicholas Baron. Windows NT Supports POSIX, But Does it Matter? *BYTE,* 18(12) :142, November 1993. [297d].

[Bar93b] Andrew Bart. Locating and Retrieving Internet Files. *UnixWorld,*10(6):75-82, July 1993. [254a].

[Bau94a] David Baum. Categorizing Client/Server Tools.*Data Based Advisor,*12(4):70, April 1994.[358b].

[Bau94b] David Baum. Chemical Company Makes Middleware Work. *Data Based Advisor,* 12(1):108-110, January 1994. [355c].

[Bau94c] David Baum. Client/Server Development Tools: Which One's for You? *Data Based Advisor,* 12(4):68-69, April 1994. [358a].

[BC91] Joseph Boykin and David Cheirton. Operating Systems: A Vision of the Year 2000. *IEEE Computer,* pages 108-109, September 1991. [095].

[Ben92] Lee Anne Bentsari. Flying to Open Systems. *Unix Review,*10(3):25-29, March 1992. [096].

[Ber91] Thomas Berlage. *OSF/Motif: Concepts and Programming.* Addison-Wesley, Workingham, England, 1991. [124].

[Ber94] Marcelo Bernstein. As ferramentas Necessarias em Projetos de Reengenharia. *BYTE Brasil,* 3(6):58-60, June 1994. [330b].

[Bet94] Mark Betz. Interoperable Objetcts. Dr.*Dobb's Journal,* (220):18-39, October 1994. [119b].

[BG93] Scott Bradner and David Greenfield. Routers: Building the Highway. PC Magazine, 12(6):221-258, March 1993.[385a].

[Bil94] Sally A.Biles. *ONC+ Distributed Computing,* chapter 6, pages 129-174. Volume 1 of Khanna [Kha94a], 1994. [132].

[Bir93] Kenneth Birman. The Process group approach to Reliable Distributed Computing. *Communications of the ACM*, 36(12):36-53, december 1993.

[Bla90] D.L.Black. Scheduling Support for Concurrency and Paralellism in the Mach. *IEEE Computer,* 23(5):35-43, May 1990. [205+].

[Bla91] Uyless Black. *The X Series Recommendations: Protocols for Data Communication Networks.* McGraw-Hill, New York, NY, 1991. [021].

[Blo92] John Bloomer. *Power Programming with RPC.* O'Reilly & Associates, Sebastopol, CA, 1992. [025].

[Blo95] John Bloomer. Distributed Computing and the DCE/OSF. Dr.*Dobb's Journal,* (227):18-30, February 1995.

[BM90] S.M.Bellovin and M.Merritt. Limitations of the Kerberos Authentication System. *Practice on Software Engineering,* 1990. [115].

[Bor93] Pauline Borsook. Seeking Security. BYTE, 18(6):118-128, May 1993. [291b].

[Bou90] K. Bounemra. *Normalisation des Reseaux - La Couche Application et la Gestion des Reseaux.* Eyrolles, Paris, France, 1990.

[Bri93a] Linda L. Briggs. Forrest & Trees Gives GUI Face to Mainframe Data. *Data Based Advisor,* 11(5):121-126, May 1993. [350d].

[BRI93b] BRISA. *Gerenciamento de Redes: Uma Abordagem de Sistemas Abertos.* Makron Books/BRISA Sociedade para a Interconexao de Sistemas Abertos, Sao Paulo, SP, 1993. [032].

[BRI94b] BRISA. *Arquiteturas de Redes de Computadores - OSI e TCP/IP.* Makron Books,1994.

[Bry93] Erik Brynjolfsson. The Productivity Paradox of Information Technology. *Communications of the ACM*, 36(12):66-77, december 1993.

[Bro93] Mark Brownstein. Data Traffic Control Comes to Unix. *UnixWorld*, 10(9):63-64, September 1993.[256a].

[Bry93) John Bryan. Pumping up Ethernet. *BYTE,* 18(9):121-129, August 1993. [294c].

[Bur92] James Burke. Technology and the New World Order. *BYTE,* 17(14):324, December 1992. [285h].

[Bur93a] Mike Burgard. 4GLs Promise Speed and Power. *UnixWorld*, 10(4):109-112, April 1993. [251c].

[Bur93b] Mike Burgard. RAID Users Talk About Solutions. *UnixWorld,* 10(5):115-120, May 1993.[252b].

[Bur93c] Mike Burgard. Enterprise Wide Databases for Everyone. *UnixWorld*, 10(11):131-136, November 1993. [258e].

[Bur93e] Mike Burgard. Unix and Mainframes: Working Toward a Peaceful Coexistence. *UnixWorld*, 10(9):111-114, September 1993. [256c].

[Byr94] Charles J. Byrne. *Fault Management,* chapter 9, pages 268-299. Volume 1 of Aidarous and Plevyak [AP94b], 1994. [159].

[Cas93] Jerry Cashin. *Client/Server Technology: The New Direction in Computer Networking.* Computer Technologies Research, Charleston, SC, 1993. [010].

[CB83] A.L. Cervo and P.A. Bervian. *Metodologia Cientifica.* McGraw-Hïll do Brasil, São Paulo, SP, 3a.edição, 1983. [120].

[CB94] W.Cheswick and Steven Bellovin. *Firewalls and Internet Security.* Addison Wesley, Reading, Massachussetts, 1994. [166].

[CFSD9O] J.Case, M.Fedor, M.Schoffstall, and J.Davin. A Simple Network Management Protocol SNMP. *Request For Comments 1157,* may, 1990.

[Cha90] G.A. Champine et al. Project Athena as a Distributed Computer System. *IEEE Computer,* 23(9):40-51, September 1990. [207+].

[Cha92] David Chappell. Tutorial on OSF Distributed Computing Environment DCE: Concepts and Protocols. Conference Presentation, Chappell and Associates; Presented in conjunction with the 1992 IFIP WG6.5 International Conference on Upper Layer Protocols, Architectures and Applications, Vancouver, Canada, May 1992. [070].

[Cha94] David Chappell. *The OSF Distributed Computing Environment DCE,* chapter 7, pages 175-199. Volume 1 of Khanna [Kha94a], 1994. [133].

[Chi94] Mark Childres. Client/Server Connectivity. *Data Based Advisor,* 12(1):49-50, January 1994. [355g].

[CHo93] Michael Horwith. Programming Standards Easy Development. *Data Based Advisor,* 11(11):111-114, November 1993. [353d].

[Cla92a] Mark Clarkson. Whats in an Object? *BYTE,* 17(14):148, December 1992. [285b].

[Cla92b] Bill Claybrook. *OLTP: On-line Transaction Processing Systems.* John Wiley and Sons, New York, NY,1992. [144].

[Cla93a] Mark Clarkson. All Terrain Networking. *BYTE,* 18(9):111-120, August 1993. [294b].

[Cla93b] Mark Clarkson. The Many Flavors of SQL. *BYTE,* 18(7):109-112, June 1993. [292b].

[Cla93c] Mark A. Clarkson. Hitting Warp Speed for LANs. *BYTE,* 18(3):123-128, March 1993. [288d].

[CL93] Larry Constantine and Lucy Lockwood. Project Organization and Management. *Communications of the ACM.* 36(10):30-33, october 1993.

[CMRW93] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Introduction to Version 2 of the Internet-Standard Network Management Framework. *Request For Comments 144l,* apr, 1993.

[Coh93] David L. Cohen. AFS:NFS on Steroids. *Lan Technology,* pages 51-62, March 1993. [092].

[Col93a] Barbara Cole. Finding the Trainers: Client/Server Training. *Data Based Advisor,* 11(10):74-78, October 1993. [352d].

[Col93b] Barbara Cole. Getting ODBC Drivers. *Data Based Advisor,* 11(11):88-93, November 1993. [353b].

[Col93c] Barbara Cole. What About IDAPI? *Data Based Advisor,* 11(11):94-96, November 1993. [353c].

[Col93d] Regan Coleman. Inside Sybase SQL Server. *Data Based Advisor,* 11(12):101- 103, December 1993. [354b].

[Der92c] Daniel P Dern. Plugging Into the Internet. *BYTE,* 17(10):149-156, October 1992. [283a].

[Dew93] Dawna Travis Dewire. *Client/Server Computing.* McGraw-Hill,New York, NY,1993. [125].

[Dew94] Dawna Travis Dewire. *Application Development for Distributed Environments.* McGraw-Hill, New York, NY, 1994. [126].

[DF91] Frank J. Derfler and Les Freed. *Guide to Using NetWare.* Ziff-Davis Press, Emeryville, Ca, 1991. [004].

[dH93a] John de Haven. Stealth Virus Attacks. *BYTE,* 18(6):137-143, May 1993. [291d].

[dH93b] John de Haven. Virus Protection for Networks. *BYTE,*18(6):144-145, May 1993. [291e].

[DJ92] Frank J.Derfler Jr. Server Monitoring Software: Getting Inside Your File Server. *PC Magazine,* 11(15):277-299, September 1992. [381a].

[DJ93a] Frank J.Derfler Jr. Low Cost Analyzers: An Eye Into the LAN. *PC Magazine,* 12(1):277-299, January 1993. [382a].

[DJ93b] Frank J.Derfler Jr. Making the WAN Connection: Linking LANs. *PC Magazine,* 12(5):183-206, March 1993. [384a].

[DJ93c] Frank J.Derfler Jr. Network Management: Behind the Scenes. *PC Magazine,* 12(21):335-365, December 1993. [387a].

[Dod94] Mark Dodge. Network and System Administration: Doing it All. *UnixWorld's Open Computing,* 11(5):33-34, May 1994. [262g].

[Don92] John Donovan. Overview: Operating Systems Trends.*BYTE,* 17(10):158-166, October 1992. [283b].

[Dru91] Peter F.Drucker. Managing for Business Efectiveness. *Harvard Business Review Classics: Fifteen Key Concepts for Managerial Success - Special Edition,* pages 58-65, 1991. [216a].

[Dum91] Ed P. Dumphy. *The UNIX Industry: Evolution, Concepts, Architecture, Applications and Standards.* QED Technical Publishing Group, Wellesley, MA, 1991. [030].

[Eng94a] Natalie Engler. Cross Platform E-Mail Promises the World. *UnixWorld's Open Computing,* 11(1):58-62, January 1994. [259b].

[Eng94b] Natalie Engler. The Selling Game. *UnixWorld's Open Computing,* 11(6):52-60, June 1994. [265a].

[Eng94c] Natalie Engler. Turning GUI Into Gold. *UnixWorld's Open Computing,* 11(3):68-76, March 1994. [261a].

[ES93] H.Eglowstein and B.Smith. Multiplatform E-Mail. *BYTE,* 18(3):123-128, March 1993. [288e].

[ES94] Howard Eglowstein and Ben Smith. E-mail From Afar. *BYTE,* 19(5):122-132, May 1994. [303c].

[FA94] D.Frey and R.Adams. *!@%#:A Directory of Electronic Mail Addressing and Networks.* O' Reilly and Associates, Sebastopol, CA, 4th edition, 1994. [164].

[Far93a] Rick Farrow. Keeping the World Out of Your Networks. *UnixWorld's Open Computing,* 10(12):56-57, December 1993. [263c].

[Far93b] Rick Farrow. Micro-Kernels: the Soul of a New OS. *UnixWorld,* 10(11):62-64, November 1993. [258b].

[Far93c] Rick Farrow. The Politics of System Administration. *UnixWorld,* 10(6):57-60, June 1993. [253b].

[Far93d] Rick Farrow. Two Worlds Collide: Unix Goes Intel. *UnixWorld,* 10(6):61, June 1993. [253c].

[Far94a] Rick Farrow. Whose Security? *UnixWorld's Open Computing,* 11(5):103, May 1994. [262f].

[Fil91] Rich Filkenstein. Four Rules for Downsizing Databases. *Data Based Advisor,* 9(4):110, April 1991.[359a].

[FS92] Richard Freund and Howard Jay Siegel. Heterogeneous Processing. *IEEE Computer,* June 1992. [112].

[FW93] H Fersko-Weiss. Contact Managers: Keeping in Touch. *BYTE Special Issue* - spring/93, 18(5):109-114, April 1993. [290a].

[Gar93] Ann Marie Garcia. Directory of Database Management Systems. *Data Based Advisor,* 11(5):58-83, May 1993. [350b].

[Gar94a] Ann Marie Garcia. The Changing Application Development Industry. *Data Based Advisor,* 12(3):53-70, March 1994. [357a].

[Gar94b] Rochelle Garner. How Client/Server Computing is Like Teenage Sex. *Unix World's Open Computing,* 11(2):20, February 1994. [260b].

[Gar94c] Rochelle Garner. Unsupportable Costs. *UnixWorlds Open Computing,* 11(2):34-40, February 1994. [260a].

[Gar95] Simon Garfinkel. PGP: Pretty Good Privacy. O'Reilly & Associates, Sebastopol, CA, 1995.

[GC93] Raymond Ga Cotê. Workgroup: a First Class Experience. *BYTE,* 18(10):149-152, September 1993.[295b].

[Gee94] Daniel E. Geer. *Lessons Learned from Project Athena,* chapter 9, pages 221- 247. Volume I of Khanna [Kha94a], 1994. [135].

[Ges93a] Chet Geschickter. Data Replication Moves Center Stage. *Client/Server Toolwatch,* 2(4):2-3, June 1993. [056c].

[Ges93c] Chet Geschickter. Making the Decision to Replace Legacy Applications. *Client/Server Toolwatch,* 2(6):2-3, August 1993. [054].

[Gil91] Kelly Gillespie. Overcoming MIS Old Thinking. *Unix Today,* July 1991. [212].

[Gil93] Kelly Gillespie. Inside Informix Online 6.0. *Data Based Advisor,* 11(9):115, January 1994. [351g].

[Gil94] Kelly Gillespie. Squash Those Client/Server Bugs. *Data Based Advisor,* 12(1):130, January 1994. [355e].

[GL92] Olivier Guedon and Herve Lecart. Administration des Environnements Distribues: Presentation de Simple Network Management Protocol. D.E.S.S. I.R.S., 1992.

[Gli92] Steven C. Glines. *Downsizing to Unix.* New Riders Publishing, Carmel, Indiana, 1992. [007].

[God93a] Doug Goddard. The Event-Message-Action Paradigm. *Data Based Advisor,* 11(12):118-120, December 1993. [354d].

[God93b] Doug Goddard. SQL Windows is Rooted in Client/Server and Continues to Bloom. *Data Based Advisor,* 11(10):102-106, October 1993. [352g].

[Goe93a] Felix Goehtgens. Configuring DNS. *UnixWorld,* 10(7):72-77, August 1993. [255c].

[Goe93b] Felix Goehtgens. Configuring DNS part II. UnixWorld,10(9):70-72, September 1993. [256b].

[Gou89] S.J. Gould. *Vida Maravilhosa: o Acaso na Evolução e a Natureza da História.* Companhia das Letras, Sao Paulo, SP, 1989. [163].

[Gou94] Michael Goulde. The Ten Worst Things That Could Happen to Interoperability in 1994. *UnixWorld's Open Computing*, 11(1):37, January 1994. [259c].

[Gow91] Kathleen Gow. Fast Tech Change Means Org Chart Must Adjust. *Client/Server Computing Supplement,* September 1991. [213].

[Gra91] Pamela A.Gray. *Open Systems: A Business Strategy for the 1990s.* McGraw-Hill, Berkshire, England, 1991. [006].

[Gre93] Rick Greham. Building SQL Front-Ends. *BYTE,* 18(12):240-248, November 1993. [297e].

[Gre94a] Joshua Greenbaum. OSI in Europe: Right Idea,Wrong Approach. *UnixWorld's Open Computing*, 11(3):45-46, March 1994. [261c].

[Gre94b] Joshua Greenbaum. Piracy: Europes Dirty Little Secret. *UnixWorld's Open Computing*, 11(5):37-38, May 1994. [262h].

[Gre94c] Joshua Greenbaum. Unified Unix in Europe: Too Little, Too Late. *UnixWorld's Open Computing,* 11(1):39, January 1994. [259d].

[Gun93] Gary Gunnerson. Network OSs: Playing the Odds. *PC Magazine*, 12(18):285-335, October 1993. [390b].

[Gun94] Angela Gunn. Front Ends Easy Internet Access. *BYTE,* 19(5):30, May 1994. [303a].

[Hal93a] Gene Hall. How to Make Reengineering Really Work. *Harvard Business Review,* 71(6):119-130, nov/dec 1993. [215a].

[Hal93b] Fred Halsall. *Data Communications, Computer Networks and Open Systems.* Addison-Wesley, Workingham, England, 3rd edition, 1993. [161].

[Ham93] James Hamilton. Power Mail Handling for Unix Systems. *Unix World*, 10(11):66-76, November 1993. [258c].

[Han94] Ted Hanss. *Institutional File System at the University of Michigan,* chapter 10, pages 248-273. Volume 1 of Khanna [Kha94a], 1994. [136].

[Hay92] Frank Hayes. Rivals Compete for E-MAIL Standards. *UnixWorld, 9*(7):89-90, July 1992. [250c].

[Hay93a] Frank Hayes. Disaster happens. *UnixWorld*, 10(7):65-68, August 1993. [255a].

[Hay93b] Frank Hayes. IBM Crosses the Line. *UnixWorld*, 10(6):61, June 1993. [253d].

[Hay93c] Frank Hayes. Texas Instruments CSA/CSM. *Client/Server Toolwatch,* 2(9):4-15, November 1993.
[052c].

[Hay93d] Frank Hayes. The Trouble With Client/Server Systems. *UniXWorld*, 10(4):63-66, April 1993. [251a].

[Hay94] Frank Hayes. Today's Compilers. *BYTE*, 19(2):76-80, February 1994. [300a].

[HC94] Michael Hammer and James Champy. *Reengenharia: Revolucionando a Empresa em Funcao dos Clientes, da Concorrencia e das Grandes Mudancas da Gerencia.* Editora Campus, Rio de Janeiro, RJ, 9th edition, 1994. [150].

[Her92] James Herman. Distributed Network Management. *Data Comminucations,* pages 74-82, June 1992. [104].

[HL93] Jeffrey Hsu and Tony Lockwood. Collaborátive Computing. *BYTE,* 18(3):112-123, March 1993. [288b].

[HM91] Michaela Howard and Russel Meredith. *Practical Open Systems for the 1990s: An Essential Guide*
for IS Managers. Eosys Ltd., United Kingdom,1991. [014].

[Ho93) H.Ho. So You Want to Be a Book Author? *Data Based Advisor,* 11(9):144-147, September 1993. [351b].

[Hol93] Alan Holub. The Power of Inheritance. *BYTE,* 18(6):221-227,May 1993. [291g].

[Hon94] Peter Honeyman. *Ditributed File Systems,* chapter 2, pages 27-44. Volume 1 of Khanna [Kha94a], 1994. [128].

[Hop93a] Casey Hopson. Make the Database Connection. *Data Based Advisor,* 11(11):88, November 1993. [353a].

[Hop93b] Casey Hopson. ODBC Does Distributed Joins Between Databases. *Data Based Advisor,* 11(12):109-111, December 1993. [354c].

[Hop94] Casey Hopson. Developing With ODBC. *Data Based Advisor,* 12(1):49-50, January 1994. [355f].

[Hor93a] Michael Horwith. PowerBuilder 3.0: New and Improved. *Data Based Advisor,* 11(9):97-102, September 1993. [351c].

[How88] J.H.Howard et al. Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems,* 6(1):55-81, February 1988.[209+].

[HS94] David Harvey and Richard Santasala. Wireless Gets Real. *BYTE,* 19(5):90-98, May 1994. [303c].

[Hub92] Mary Hubley. GUIs, Applications and Unix. *BYTE,* 17(10):186-189,October 1992. [283c].

[Hun92a] Craig Hunt. *TCP/IP Network Administration.* O'Reilly & Associates, Sebastopol, CA, 1992. [123].

[Hun92b] Bruce H. Hunter. Surveying Fur-Flang Networks: Distributed Network Monitors. *BYTE,* 17(8):204-220, August 1992. [281c].

[Hur93a] Judith S. Hurwitz. The Business Decision About Object Orientation. *Glient/Server Toolwatch,* 2(5):2-3, July 1993. [055a].

[Hur93b] Judith S. Hurwitz. COSE: Unix's Best Hope? *Client/Server Toolzwatch,* 2(1):3-4, March 1993. [58b].

[Hur93c] Judith S. Hurwitz. Getting Started With Client/Server: Dividing the Work into Incremental Tasks. *Client/Server Toolwatch,* 2(9):1-3, November 1993. [052a].

[Hur93d] Judith S. Hurwitz. Intelligent Planning for Client/Server Development. *Client/Server Toolwatch,* 2(7):1-2, September 1993. [053a].

[Hur93e] Judith S. Hurwitz. Managing Client/Server Applications: The Next Challenge. *Client/Server Toolwatch,* 2(9):3, November 1993. [052b].

[Hur93f] Judith S. Hurwitz. Oracle's Tools Strategy: Beyond the Engine. *Client/Server Toolwatch,* 2(3):3-10, May 1993. [057].

[Hur93g] Judith S. Hurwitz. Rapid Application Development: Know thy Tool. *Client/Server Toolwatch,* 2(1):1-2, March 1993. [058a].

[Hur93h] Judith S. Hurwitz. Software Scalability: Looking Behind the Veneer. *Client/Server Toolwatch,* 2(4):2-3, June 1993. [056a].

[Hut93] Tony Hutchings et al. Process Improvement That Lasts: An Integrated Training and Consulting Method. *Communications of the ACM*, 36(10):104-113, october 1993.

[JM90] Celia A. Joseph and Kundi H. Mualidham. Integrated Network Management in an Enterprise Environment. *IEEE Network Magazine,* pages 7-13, July 1990. [103].

[Joh94] Philip A. Johnson. *Domestic and International Open Systems Interconnection Management Standards,* chapter 4, pages 122-135. Volume 1 of Aidarous and Plevyak [AP94b], 1994. [155].

[Jur92] Ronald K. Jurgen. Technology 1992. *IEEE Spectrum,* pages 26-35, January 1992. [201].

[KA90] Setrag Khoushafian and Raznik Abnous. *Object Orientation: Concepts, Languages, Databases, User Interfaces*. John Wiley, New York, NY, 1990.

[Kas92] Mohammed Kasmi. Gestion de Reseaux: Le Standard SNMP - Simple Network Management Protocol. D.E.S.S. I.R.S.,1992.

[Kay94a] Russell Kay. Distributed and Secure. *BYTE*, 19(6):165-180, June 1994. [304d].

[Kay94b] Russell Kay. Objects in Use. *BYTE*, 19(4):99-104, April 1994. [302c].

[Keg92] Harris Kegan. Objects in Real Time. *BYTE*,17(8):187-190, August 1992. [281e].

[Keh92] Brendan P. Kehoe. *Zen and The Art of the Internet: A Beginner's Guide to the Internet*. Prentice-Hall, Widener University, Chester, PA, January 1992. [143].

[Kha94a] Raman Khanna, editor. *Distributed Computing: Implementation and Management Strategies*, Volume 1. Prentice-Hall, Englewood-Cliffs, NJ, 1994. [031]

[Kha94b] Raman Khanna. *Introduction,* chapter l, pages 1-23. Volume 1 of [Kha94a], 1994. [165].

[Kla93] Todd C.Klaus. Creating Motif Applications. *UnixWorld*, 10(10):60-66,October 1993. [257b].

[Kle88] S.Mark Klerer. The OSI Management Architecture : An Overview. *IEEE Network Magazine,* mar, 1988.

[Kob91] Shiz Kobara. *Visual Design with OSF/Motif.* Addison-Wesley, Reading, MA, 1991. [011].

[Koe93] Scott Koegler. Backing Up a Backup Policy. *UnixWorld*, 10(10):67-70, October 1993. [257c].

[Kor93] Paul Korzeniomski. Make Way for Data. *BYTE,* 18(7):113-120, June 1993. [292c].

[KP92] Ashfag A.Khokar and Viktor Prassanme et al. Heterogeneous Computing: Challenges and Opportunities. *IEEE Computer,* June 1992. [113].

[KW93] Jeffrey O.Kaphart and Steve R.White. Computers and Epidemiology. *IEEE Spectrum,* pages 20-26, May 1993. [203].

[Lam91] Buther Lampson. Authentication in Distributed Systems: Theory and Practice. *ACM SIGOPS,* pages 165-182, 1991. [114].

[Lan94] Alex Lane. Optimizing Your Today's CPU. *BYTE*, 19(2):81-90, February 1994. [300b].

[LaP93] Alice G.LaPlante. Can You Trust Your Consultant. *UnixWorld*, 10(10):111-114, October 1993. [257a].

[LD93b] Mike Lewis and Debbie Dickstein. How to Choose a Database Management System. *Data Based Advisor*, 11(5):40-57, May 1993. [350a].

[Leo94] Jonh Leong. *Project Andrew,* chapter 8, pages 203-220.Volume 1 of Khanna [Kha94a], 1994. [134].

[Lin94a] C.Links. Universal Wireless LANs. *BYTE*, 19(5):99-112, May 1994. [303d].

[Lit93] Jonathan Littman. Commerce on the Internet: the Digital Gold Rush. *UnixWorld's Open Computing*, 10(12):42-47, December 1993. [263a].

[Lit94] Jonhatan Littman. Seeing is Believing. *Unix World's Open Computing*, 11(1):40-50, January 1994. [259a].

[LM92] Eva Lakatos and Marina Marconi. *Metodologia do Trabalho Científico.* Atlas, São Paulo, SP, 4th edition, 1992. [121].

[LS90] Eliezer Levy and Abraham Silberschatz. Distributed File Systems: Concepts and Examples. *ACM Computing Surveys,* 22(4):321-374, December 1990. [091].

[Lu92] Gary Lu. Objects for End-Users. *BYTE,* 17(14):142-152, December 1992. [285a].

[Mae87] Mamoru Mackawa. *Operating Systems: Advanced Concepts.* Benjamin-Cumrnings, Menlo Park, CA, 1987. [026].

[Mal92a] Carl Malamud. *Analyzing SUN Networks.* Van Nostrand Heinhold, New York, NY, 1992. [024].

[Mal92b] Carl Malamud. *STACKS: Interoperability in Today's Computer Networks.* Prentice-Ha11, Englewood-Cliffs, NJ, 1992. [005].

[Man92] Jim Manzi. The Productivity Macguffin. *BYTE,* 17(8):360, August 1992. [281d].

[Mar91] Brian D.Marsh. First Class User Level Threads. *In Proceedings of the thirteenth ACM Symposium on Operating Systems,* pages 110-121, October 1991. [116].

[Mar93a] Bill Martorelli. The Sombers Group's Netwave Extends OLTP. *Client/Server Toolwatch,* 2(7):10-12, September 1993. [053c].

[Mar93b] Roger Martin. Changing the Mind of The Corporation. *Harvard Business Review.* 71(6):81-96, Nov-Dec 1993. [215c]

[Mas92] Tony Mason. OSF's DCE. *Unix Review*, 10(1):31-34, January 1992. [093].

[May94] Thornton A. May. Shakesperian Wisdom. *BYTE,* 19(1):312, January 1994. [299f].

[McL92a] Spencer J. McLimurray. Client/Server Architecture: Direction for the 90's. Conference Presentation, Gartner Group, Stamford, CT, 1992. [035].

[McL92b] Spencer J. McLimurray. Operations Automated Systems: Redefining Business Process Support. Conference Presentation, Gartner Group, Stamford, CT, 1992. [041].

[McL93] Ephrain McLean et al. Converging End-User and Corporate Computing. *Communications of the ACM,* 36(12):7893, december 1993.

[Mel93] John P. Mello. Future Communications. *BYTE,* 18(9):94-I10, August 1993. [294a].

[Men93] David Menninger. In Search of the Perfect Server. *Data Based Advisor,* 11(9) :103-107, September 1993. [351d].

[Men94a] David Menninger. Under the Hood of SQL. *Data Based Advisor*, 12(1):111-115, January 1994. [355d].

[Men94c] David Menninger. Select * From RDBMS. *Data Based Advisor,* 12(4):76-88, April 1994. [358d].

[Mez93] Dan Mezick. Data Modelling: the Building Blocks to Better Applications. *Data Based Advisor*, 11(10):79-89, October 1993. [352e].

[MIC93] MICROSOFT. Microsoft Windows NT and Client/Server Computing: An overview. Microsoft Co., Redmond, WA, May 1993. [051].

[Mil87] Milan Milenkovic. *Operating Systems: Concepts and Design.* McGraw-Hill, New York, NY, 1987. [149].

[Mil91] Michael D.Millikin. *Distributed Computing Futures: DCE, ONC and Beyond.* Conference Presentation/Unix Expo, Gunstock Hill Associates, Gilford, NH, October 1991. [044].

[Mil93] Michael J.Miller. Making Your Applications Work Together. *PC Magazine,* 12(6):108-138, March 1993. [385b].

[Mil94a] Glenn M.Miller. Data server. *UnixWorld's Open Computing,* 11(2):24, February 1994. [260d].

[Mil94b] Glenn M.Miller. How to Help Users Understand MISs Mission. *UnixWorld's Open Computing*, 11(2):22-24, February 1994. [260c].

[Mil94c] Michael D.Millikin. DCE: Building the Distributed Future. *BYTE*, 19(6):125-138, June 1994. [304a].

[Moc93] Kevin A.Mocklin. Put Your Objects Front-End Center. *UnixWorld*, 10(11):119-124, November 1993. [258d].

[Mor86] J.Morris et al. Andrew: A Distributed Personal Computing Environment. *Communications ofthe ACM,* March 1986. [208+].

[Mos93] Walter Mossberg. They Just Don't Have IS. *BYTE,* 18(9):268, August 1993. [294d].

[MP94] Carlos Machado and Sônia Penteado. É Hora de Recomeçar. *Informatica Exame,* 9(96):64-71, March 1994. [214].

[MR88a] K.McCloghrie and M.Rose. Management Information Base for Network Management of TCP/IP-Based Internets. *Request For Comments 1066, August,* 1988.

[MR88b] K. McCloghrie and M. Rose. Structure and Identification of Management Information for TCP/IP-based internets. *Request For Comments 1065,* aug, 1988.

[MR91] K. McCloghrie and M. Rose. Common Management Information Services and Protocol Over TCP/IP (CMOT). *Request For Comments 1189,* mar,1991.

[MS94] R.L. Morgan and Roland Schemers. *Migration Strategies,* chapter 15, pages 393-423. Volume 1 of Khanna [Kha94a], 1994. [139].

[MSS93] Masoud Mansouri-Samani and Morris Sloman. Monitoring Distributed Systems. *IEEE Network Special Issue: Integrated Network Management,* 7(6):20-31, November 1993. (118].

[Mul90] S.J. Mullender et al. AMOEBA: a Distributed Operating System for The 1990s. *IEEE Computer,* 23(5):44-53, May 1990. [206+].

[Mul94] Craig S. Mullins. The Great Debate. *BYTE,* 19(4):85-98, April 1994. [302b].

[Mur94] Kundi H. Muralhidar. *Knowledge Based Network Management,* chapter 7, pages 200-231. Volume 1 of Aidarous and Plevyak [AP94b], 1994. [157].

[Mye93a] Marc Myers. Getting the Help You Need: Client/Server Training. *Data Based Advisor,* 11(10):72-73, October 1993. [352c].

[Mye93b] Marc Myers. SQL Tools Speed Development. *Data Based Advisor,* 11(12):97-100, December 1993. [354a].

[Mye94] Marc Myers. Get Your Data Together. *Data Based Advisor,* 12(4):104-111, April 1994. [358e].

[Nad94a] Michael Nadean. Distributed Computing Resource Guide. *BYTE,* 19(6):205, June 1994. [304g].

[Nad94b] Michael Nadean. Remote Connections. *BYTE,* 19(6):197-205, June 1994. [304f].

[Nan93a] Barry Nance. LAN Remote Control. *BYTE,*18(4):210, April 1993. [289c].

[Nan93b] Barry Nance. Stackin up TCP/IP for Windows. *BYTE,* 18(2):215-218, February 1993. [287a].

[Nan93c] Barry Nance. Troubleshooting Your LAN: Cable Detectives. *PC Magazzne,* 11(13):335-350, July 1993. [389a].

[Nel90] V.P. Nelson. Fault Tolerant Computing: Fundamental Concepts. *IEEE Gomputer,* 23(6):19-25, July 1990. [211+].

[Ols94] Douglas K. Olson. Developing for Multiple Platforms. *BYTE,* 19(2):91-96, February 1994. [300c].

[O'M93] Cristopher O'Malley. E-Mail Unplugged by Wireless LANs. *BYTE,* 18(12):28, November 1993. [297a].

[O'R88] O'Reilly & Associates. Xlib Programming Manual for Version 11 Release 2 of the X Window System. Volume I. Manual de Referência. O'Reilly & Associates, Sebastopol, Ca, 1988.

[OSF90a] OSF. A Look at Computing in The 1990's. A White Paper, Open Software Foundation, Cambridge, MA, August 1990. [077].

[OSF90b] OSF Distributed Computing Environment: Rationale. A White Paper, Open Software Foundation, Cambridge, MA, May 1990. [078].

[OSF90c] OSF. Security in The OSF/1 Operating System. A White Paper, Open Software Foundation, Cambridge, MA, November 1990. [082b].

[OSF91a] OSF. Answers To Commonly Asked Questions About the OSF Architectural Neutral Distributed Format: Rationale. A White Paper, Open Software Foundation, Cambridge, MA, February 1991. [083b].

[OSF91b] OSF. *Guide to OSF/1: A Technical Synopsis.* O'Reilly & Associates, Sebastopol, Ca, 1991. [020].

[OSF91c] OSF. Interoperability: A Key Criterion for Open Sysytems. A White Paper, Open Software Foundation, Cambridge, MA, November 1991. [076].

[OSF91d] OSF. OSF Architectural Neutral Distributed Format: Rationale. A White Paper, Open Software Foundation, Cambridge, MA, June 1991. [083a].

[OSF91e] OSF. Remote Procedure Call in a Distributed Computing Environment. A White Paper, Open Software Foundation, Cambridge, MA, August 1991. [080].

[OSF92a] OSF. An Analyses of the OSF/1 Operating System and Unix V Release 4. A White Paper, Open Software Foundation, Cambridge, MA, January 1992. [082a].

[OSF92b] OSF. From UNCOL to ANDF: Progress in Standard Intermediate Languages. Technical Paper, Open Software Foundation, Cambridge, MA, January 1992. [083d].

[OSF92c] OSF. OSF Distributed Computing Environment: An Overview. A White Paper, Open Software Foundation, Cambridge, MA, January 1992. [079].

[OSF92d] OSF. Security in a Distributed Computing Environment. A White Paper, Open Software Foundation, Cambridge, MA, janeiro 1992. [081].

[OSF92e] OSF. The Structure of ANDF: Principles and Examples. A White Paper, Open Software Foundation, Cambridge, MA, January 1992. [083c]

[OT92] Tim O'Reilly and Grace Todino. *Managing UUCP and Usenet*. O'Reilly & Associates, Sebastopol, CA, 10th. edition, January 1992. [029]

[OV91] Tamer Ozsu and Patrick Valduriez. *Principles of Distributed Database Systems*. Prentice-Hall, Englewoods-Cliffs, NJ, 1991. [034].

[PA92] Tekla S. Perry and John A. Adler. E-MAIL: Pervasive and Persuasive. *IEEE Spectrum*, pages 22-33, October 1992. [202].

[PB92]Dick Poutain and John Bryan. All Systems Go: Parallel Processing Appreciation. *BYTE*, 17(8):112-136, August 1992. [281a].

[Pen93] Ed Penatore. IBM Makes MP Promises for OS/2. *BYTE*, 18(12):114, November 1993. [297c].

[Per93a] Edwin C.Perkins. Supercharged Sybase Application Development. *UnixWorld*, 10(10):30, October 1993. [257e].

[Per93b] Edwin C. Jr. Perkins. On-line Reliability With RAID. *UnixWorld*, 10(6):113-116, July 1993. [254c].

[Pet92] Mary Petrosky. Routing: What's Around the Band? *LanTechnology*, pages 50-64, August 1992. [107].

[PK91] M.L.Powell and S.R.Kleiman et al. Sun OS Multi-threaded Architecture. USENIX Conference Presentation, 1991. [101].

[PL93] Susan Perscheck and Michael Liczberski. Is OOP in your future? *Data Based Advisor*, 11(10):49-54, October 1993. [352a].

[Poo93a] Gary Andrew Poole. In the Line of Fire. *UnixWorld*, 10(11):46-53, November 1993. [258a].

[Poo93b] Gary Andrew Poole. Its Crunch Time in Cyberspace. *UniXWorld's Open Computing*, 10(12):52-55, December 1993. [263b].

[Poo94] Gary Andrew Poole. Open Systems Fall Guy. *UnixWorld's Open Computing*, 11(5):46-49, May 1994. [262a].

[Pou93a] Dick Pountain. The Multiprocessor Solution. *BYTE*, 18(7):185-192, June 1993. [292e].

[Pou93b] Dick Pountain. Oberon: a Glimpse at the Future. *BYTE*, 18(6):111-117, May 1993.[291a].

[Pou94a] Dick Pountain. The Chorus Microkernel. *BYTE*, 19(1):131-138, January 1994. [299b].

[Pou94b] Dick Pountain. Microprocessor Trends. *BYTE,* 19(1):74-82, January 1994. [299d].

[Pro93] Steven E.Prokesch. Mastering Chaos at the High-Tech Frontier. *Harvard Business Review,* 71(6):134-144, nov/dec 1993. [215b].

[PS94] Dick Pountain and Clement Szyperski. Extensible Software Systems. *BYTE,* 19(5):57-62, May 1994. [303b].

[Pyl94] Raymond H. Pyle. *Applyinf Objetc-Oriented Analysis and Design to the Integration of Network Management Systems,* chapter 6, pages 136-158. Volume 1 of Aidarous and Plevyak [AP94b],1994.[156].

[Qua90] John S. Quaterman. *THE MATRIX: Computer Networks and Conferencing Systems Worldwide.* Digital Press, 1990. [027].

[Rad93a] Alan Raddings. Foundation for Cooperative Processing. *Client/Server Toolwatch,* 2(7):3-10, September 1993. [053b].

[Rad93c] Alan Raddings. SQL Windows. *Client/Server Toolwatch,* 2(4):2-3, June 1993. [056b].

[Ram93] Lakshmi Raman. CMISE Functions and Services. *IEEE Network Magazine,* may,1993.

[Ras92a] Daniel W. Rasmus. Object-Oriented CASE. *BYTE,* 17(14):160, December 1992. [285c].

[Ras92b] Daniel W. Rasmus. Relating to Objects. *BYTE,* 17(14):161-166, December 1992. [285d].

[Ray94] Gordon Rayney. *Configuration Management,* chapter 8, pages 234-266. Volume 1 of Aidarous and Plevyak [AP94b], 1994. [158].

[RB92] David C. Rine and Bharat Bhargava. Object-Oriented Computing. *IEEE Computer,* October 1992. [108].

[Red92a] Bill Redman. Corporate Network Operating Systems. Conference Presentation, Gartner Group, Stamford, CT, 1992. [039].

[Red92b] Bill Redman. Local Area Communications Scenario. Conference Presentation, Gartner Group, Stamford, CT, 1992. [037].

[Red92c] Bill Redman. Network Connectivity: The Physical Network. Conference Presentation, Gartner Group, Stamford, CT,1992. [038].

[Rei93] Andy Reinhardt. Smarter E-Mail is Coming. *BYTE,* 18(3):90-108, March 1993. [288a].

[Rei94a] Andy Reinhardt. Building the Data Highway. *BYTE,* 19(3):46-74, March 1994. [301a].

[Rei94b] Andy Reinhardt. Managing Storage. *BYTE,* 19(6):153-162, June 1994. [304c].

[Ric94] Jane Richter. Distributing Data. *BYTE,* 19(6):139-150, June 1994. [304b].

[Rin93] Martin L.Rinehart. When to Use Application Generator - and When Not To. *Data Based Advisor,* 11(10):65, October 1993. [352b].

[Ros90] Wayne Jr.Rosh.Getting Bigger Groupware. *BYTE,* 15(13):93-96, December 1990. [280c].

[Ros91] Marshall T.Rose.*An Introduction to Management of TCP/IP Based Internets.* Prentice-Hall, Englewood-Cliffs, NJ, 1991. [033].

[Ros93] Marshall T.Rose.Challenges in Network Management.*IEEE Network Special Issue: Integrated Network Management,* 7(6):16-19, November 1993. [117].

[Ros94] Wallace A.Ross. *Hewlett-Packard's Migration to Client/Server Architecture,* chapter 11, pages 274-298. Volume 1 of Khanna [Kha94a], 1994. [137].

[RS93] Marc Rettig and Gary Simons. A Project Planning and Development Process for Small Teams. *Communications of the ACM.* 36(10):44-55, october 1993.

[RT94] Bob Ryan and Tom Thompson. RISC Grows Up. BYTE, 19(1):91-100, January 1994. [299g].

[Rum91] James Rumbaugh et al. *Object Oriented Modelling and Design.* Prentice-Hall, Englewood-Cliffs, NJ,1991.

[Ryd93] John Rydberg. Barnyans Streettalk for Netware. *BYTE,* 18(6):199-200, May 1993. [291f].

[Sah94] Veli Sahin. *Telecommunications Management Networks: Principles, Models* and *Applications,* Chapter 3, pages 72-119. Volume 1 of Aidarous and Plevyak [AP94b], 1994. [154].

[Sal93] Joe Salemi. *Guide to Client/Server Databases.* Ziff Davis Press, Emeryville, Ca, 1993. [001].

[Sau94] Jacques Sauvé. Unix não é um Sistema Aberto. Byte Brasil, Novembro 1994.

[Sch92a] Jeff Schullman.Client/Server Keynote: A Framework for Client/Server Processing. Conference Presentation, Gartner Group, Stamford, CT, 1992. [036].

[Sch92b] Jeff Schullman. Enterpise Systems Management: Bottleneck or Enabler? Conference Presentation, Gartner Group, Stamford, CT, 1992. [042].

[Sch92c] Jeff Schullman.The Software Management Strategies: Keynote and Five Year Scenario. Conference Presentation, Gartner Group, Stamford, CT, 1992. [040].

[Sch94] Jeffrey Schiller.*Distributed System Security,* chapter 4, pages 73-106. Volume 1 of Khanna [Kha94a], 1994. [130].

[Sen94] Subhabrata Sen Sen. *Network Performance Management,* chapter 10, pages 302-335. Volume 1 of Aidarous and Plevyak [AP94b], 1994. [159].

[Set86] Valdemar W.Setzer. *Bancos de Dados: Conceitos, Modelos, Gerenciadores, Projeto Logico, Projeto Fiisico.* Edgard Blucher Editora, Sao Paulo, SP, 1986. [147].

[Sey89] Jim Seymour. The GUI: an Interface You Won't Outgrow. *PC Magazine,* 8(15):43-54 September 1989. [380].

[Sey93] Patricia Seybold. The Learning Organization. *BYTE,*18(4):264, April 1993. [289a].

[SG92a] S.Simmel and I.Godard. Object Oriented Database Managers. *BYTE,* 17(14):172, December 1992. [285f].

[SG92b] S.Simmel and I.Godard.Objects of Substance. *BYTE,* 17(14):167-171, December 1992. [285e].

[SH93] Jeannette Sill-Holeman. RDBMS's Vendor Update: Who's Who? *Data Based Advisor,* 11(5):132-134, May 1993. [350f].

[Sha94] Oliver Sharp. Compilers for Paralell CPUs. *BYTE,*19(2):97-101,February 1994. [300d].

[She93a] Mark Sherman. Distributed Transaction Processing in a DCE Environment with Encina. Presentation Script. Transarc Co, Pittsburgh, PA, 1993.

[She93b] Mark Sherman. An Introduction to Programming the Encina Monitor. Tech Report. Transarc Co, Pittsburgh, PA, 1993.

[Shi92] Alok Sinha. Client-Server Computing:Current Technology Review. *Communications of the ACM.* 35(7):77-97, july 1992.

[Sin94] Harvinder Singh.*Distributed Database Management,*chapter 3,pages 45-72. Volume 1 of Khanna [Kha94a], 1994. [129].

[Sit93] Richard L. Sites. Digital's Alpha Chip Project. *Communications of the ACM.* 36(2):33-44, february 1993.

[Ski94] Richard A.Skinde. The Metamorphosis of Oracle. *Data Based Advisor,* 12(2):84-88, February 1994. [356c].

[Skr94] Richard Skrinde.How will Oracleware Fare Against the Might of Microsoft? *Data Based Advisor,* 12(3):92-95, March 1994. [357b].

[Smi94) Ben Smith. Client/Server Made Easy. *BYTE,* 19(3):143-144, March 1994. [301c].

[SOIEC89]International Standards Organization / International Electrotechnical Commission. Information Processing Systems - Open Systems Interconnection - Basic Reference Model - Part 4: Management Framework. *ISO/IEC 7498-4,* nov, 1989.

[SOIEC90]International Standards Organization / International Electrotechnical Commission. Information Technology - Open Systems Interconnection - Common Management Information Protocol Specification. *ISO/IEC 9596,* may, 1990.

[SOIEC9la]International Standards Organization / International Electrotechnical Commission. Information Technology - Open Systems Interconnection - Systems Management Overview.*ISO/IEC DIS 10040,*may,1991.

[SOIEC91b]International Standards Organization / International Electrotechnical Commission. Information Technology - Open Systems Interconnection - Structure of Management Information - Part 1: Management linformation Model.*ISO/IEC DIS 10165-1,*mar,1991.

[SOIEC91c]International Standards Organization / International Electrotechnical Commission. Information Technology - Open Systems Interconnection - Structure of Management Information - Part 4: Guidelines for the Definition of Managed Objects. *ISO/IEC DIS 10165-4,*mar,1991.

[SOIEC91d] International Standards Organization / International Electrotechnical Commission. Information Technology - Open Systems Interconnection - Common Management Information Service Definition.*ISO/IEC 9595,*apr,1991.

[Sol94] Richard Mark Soley. *Role of Object Technology in Distributed Systems,* App B., pages 469-478. Volume 1 of Khanna [Kha94a], 1994. [141].

[Sos94] Barry Sosinsky. Building Applications that Travel. *Data Based Advisor,* 12(1):49-50, January 1994. [355a].

[Sov92] John Sovereign. The Power of POSIX Thinking. *Unix World,* 9(7):93-104, July 1992. [250b].

[Spe91] Spectrum. A Guide to Engineering Workstations. *IEEE Spectrum,* pages 31-67, April 1991. [102].

[SS94] William Stallings and Ben Smith. SNMP Versao 2. *BYTE,* set, 1994.

[Sta93a] William Stallings. FDDI Speaks. *BYTE,* 18(4):197-200, April 1993. [289b].

[Sta93b] William Stallings. *SNMP, SNMPv2, and CMIP: The Practical Guide to Network Management Standards.* Addison-Wesley, Reading, MA, 1993. [127].