CARACTERIZAÇÃO DE UM SISTEMA OPERACIONAL DE REDE PARA INTEGRAR AMBIENTES HETEROGÊNEOS

Márcio Lannes Duarte de Souza

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENEGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS.

Aprovada por:

Prof. Valmir Carneiro Barbosa, Ph.D.
(Presidente)

Prof. Aloysio de Castro Pinto Pedroza, Dr.

Prof. Julius César Barreto Leite, Ph.D.

RIO DE JANEIRO, RJ - BRASIL
ABRIL - 1990

Souza, Márcio Lannes Duarte de

Caracterização de um Sistema Operacional de Rede(Rio de Janeiro) 1990

x, 114 p. 29,7 cm (COPPE-UFRJ, M.Sc., Engenharia de Sistemas, 1990)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Sistemas Operacionais para Ambientes Heterogêneos

I. COPPE/UFRJ II. Título (Série)

Agradeço ao Professor Gerhard Schwarz pelo apoio dado no início deste trabalho.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.).

# CARACTERIZAÇÃO DE UM SISTEMA OPERACIONAL DE REDE PARA INTEGRAR AMBIENTES HETEROGÊNEOS

Márcio Lannes Duarte de Souza

Abril 1990

Orientador: Prof. Valmir Carneiro Barbosa.
Programa: Engenharia de Sistemas.

Este trabalho trata do problema da integração de sistemas heterogêneos. É proposto como solução para o problema um elemento de software, denominado Sistema Operacional de Rede - SOR, cuja função é dotar o ambiente composto por unidades autônomas de um sistema operacional.

As principais características de um SOR são:
- Composição heterogênea
- Preservação do ambiente de operação dos susbsistemas componentes.
- Preservação da autonomia local.

É feita uma conceituação do sistema sob três pontos de vista: o dos requisitos a atender, o da composição e o da distribuição de funções e uso do sistema. São enfocados ainda problemas de administração, com ênfase nos aspectos de controle de acesso e gerência dos conjuntos de usuário, onde é proposto um mecanismo baseado em "capabilities" como solução. Por fim, é analisado um mecanismo para a construção dos serviços a serem

disponibilizados ao ambiente - RFE. A principal característica desse mecanismo é a capacidade de reprogramação de interfaces, possibilitando além de uma implementação gradual, o acesso por elementos com graus de complexidade variáveis.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M. Sc.).

CARACTERIZATION OF A NETWAORK OPERATING SYSTEM
TO INTEGRATE HETEROGENEOUS ENVIRONMENTS

Márcio Lannes Duarte de Souza

April, 1990

Thesis Supervisor: Prof. Valmir Carneiro Barbosa.
Department: Engenharia de Sistemas.

This work treats the problem of integrating heterogeneous systems. A software entity, named Network Operating System, is proposed as a solution. Its function is to be the Operating System of the environment composed by the interconnected autonomous unit.

Its main caracteristics are the following:

- Heterogeneous composition
- Preservation of the native operating environment of each component subsystem.
- Preservation of local autonomy.

The system is defined from three points of view: the requirements to fulfill, its composition and function distribution and system usage. Problems concerning system administration are focused, with enphasis on aspects of access control and on the management of the sets of users. Then a mechanism based on capabilities is proposed as a solution. Finally, a mechanism for constructing the services to be made avalable in the environment - named RFE - is analyzed. Its main caracteristic is its fitness to build interfaces that can be reprogrammed. In this way, besides

making a stepwise software construction possible, systems with different complexity degrees can be mixed in the same environment without overloading smaller elements or imposing poor performance to the more complex ones.

# ÍNDICE

# CAPÍTULO I

# INTRODUÇÃO

## 1.1) APRESENTAÇÃO DO ASSUNTO

Ao observarmos as atividades desenvolvidas em um sistema de computação, podemos notar a existência de uma série de ambientes de operação. Cada ambiente é construído para suportar da maneira mais adequada as atividades desempenhadas por seus usuários. Por princípios de modularidade, tem sido prática a criação de ambientes novos na forma de valor agregado. Existe assim uma estratificação, onde o software de camadas mais básicas fornecem serviços para camadas superiores, de forma que as mesmas possam, por sua vez, desempenhar suas atividades.Essa estruturação pode ser notada na hierarquia existente para a operação de um programa aplicativo. O quadro na seqüência, por exemplo, evidencia este tipo de hierarquização.

| |
|---|
| o ambiente oferecido pelo aplicativo |
| a linguagem oferecida para a sua programacao |
| o metodo de acesso às bases de dados |
| as regras de acesso ao sistema de arquivos |
| as primitivas do Sistema Operacional |
| as interfaces de acesso aos dispositivos |

Cada um dos pontos destacados pode constituir um ambiente relativamente independente dos demais. Dentro dessa estratificação, vamos nos ater àquele ambiente mais básico que fornece a primeira interface para o acesso ao sistema computacional: o seu sistema operacional.

Classicamente tem sido função de um sistema operacional organizar o acesso aos recursos físicos de um sistema qualquer.São considerados recursos físicos: a memória principal, o espaço em memória secundária, unidades processadoras, dispositivos de entrada e saída, etc. Na verdade, e até em decorrência do fato de que para organizar é preciso gerenciar, as funções de um sistema operacional extrapolam em muito a simples função de mero alocador de recursos. São funções também normalmente associadas a um sistema operacional: a gerência do conjunto de usuários existentes; o controle de acesso dos objetos sob sua guarda; a definição de um conjunto de regras para o acesso aos recursos. Este conjunto de regras define o ambiente básico de operação dos diversos tipos de "usuários". Este é um aspecto fundamental para os objetivos desta tese, uma vez que carcteriza o Sistema Operacional como um elemento Normativo.

Uma vez mostrada essa visão de um sistema operacional convencional, vamos estendê-la agora para a de um Sistema Operacional de Rede. Primeiramente, deve-se chamar a atenção para um fato que a cada dia se faz mais presente, pela contínua informatização das sociedades modernas. Este fato é o do uso, dentro de uma mesma organização, de equipamentos de computação de diversos tipos - tanto modelos diferentes quanto marcas diferentes. Dentro mesmo de nossa universidade tem-se exemplo disto.No NCE existem em operação pelo menos os seguintes tipos de computadores: dois A9, um VAX 780, um VAX 8810, 2 micro VAX, um IBM 4381, micro-computadores IBM PC compatíveis, etc. Pode-se imaginar uma série de razões para o uso de máquinas diversificadas: a disponibilidade financeira;

situação mercadológica; a existência de facilidades desejáveis específicas em cada tipo de equipamento; visão de administração do sistema; maior independência com relação a fornecedores; treinamento mais diversificado de pessoal; aspectos de custo/benefício dentro de uma organização hierarquizada; etc. Sejam lá quais forem os motivos, o que importa no momento é a existência dessa tendência à heterogeneidade.

Uma vez reconhecida esta realidade, não passa despercebida a ocorrência de um problema: o conjunto constituído pela diversidade de equipamentos existentes, encarado como um único sistema computacional, já não possui um Sistema Operacional. Não pelo menos como havíamos definido anteriormente. O sistema computacional agregado não dispõe por um lado de mecanismos para uma gerência global de seus objetos, e por outro, não existem normas que uniformizem a operação dentro de cada ambiente. As soluções normalmente existentes para a interligação dos sistemas cuidam de aspectos de acesso ou de troca de informações - Emulação de terminal, FTP, correios eletrônicos, etc. Um retrato dessa situação pode ser observado na figura (1).

O Sistema Operacional de Rede surge então com duplo enfoque. Em primeiro lugar ele é o mecanismo através do qual alguma ordem pode ser conseguida no ambiente desagregado existente. Este enfoque tem um lado positivo e outro curioso. O lado positivo é a oportunidade de se resolver um problema que gera muita ansiedade, qual seja o da integração do ambiente heterogêneo. O lado curioso é aquele que parece afastar boa parte do interesse que esse assunto poderia despertar junto a comunidade acadêmica. Existe toda uma sorte de dificuldades de ordem técnica e de implementação para construção de um Sistema Operacional com as virtudes desejadas. Existem mesmo opiniões que consideram, a priori, uma tarefa inglória a tentativa de construção de um Sistema Operacional, com as características a serem descritas posteriormente, que tenha

por objetivo por ordem nesse caos. Isto, segundo TANEMBAUM e RENESEE [1], simplesmente porque os Sistemas Operacionais nativos existentes não foram construídos antevendo os problemas decorrentes da heterogeneidade e da atuação cooperada. Essa controvérsia será abordada mais adiante. Por ora basta lembrarmos da tendência à heterogeneidade como um fato - um problema a resolver.

O segundo enfoque diz respeito precisamente ao reconhecimento da tendência à heterogeneidade - precisamos saber construir elementos novos distintos, mas que possam integrar-se da forma mais produtiva a outros já existentes. O estudo de Sistemas Operacionais de Rede traz o benefício de permitir identificar os pontos relevantes para a integração de sistemas heterogêneos, permitindo que esta tarefa se torne no futuro mais fácil pela melhor compreensão dos problemas e pela análise das soluções adotadas para os mesmos. Outro benefício de seu estudo é a identificação dos pontos críticos para a integração. Seria possível a construção de Sistemas Operacionais que previssem as situações associadas, trazendo embutidas consigo as facilidades necessárias para a sua integração.

## 1.2) MOTIVAÇÕES PARA A TESE

A primeira motivação para realização dessa tese já foi em parte apresentada. Quando mencionamos a tendência para a formação de ambientes de computação heterogêneos, foram dados, na verdade, dois motivos para a realização de um estudo sobre o assunto. O primeiro é a disponibilidade de casos-exemplo vivos. Nesses ambientes todos os problemas, ansiedades e expectativas estão ali, prontos para que alguém se disponha a analisá-los. Um estudo sistemático da situação e das soluções caso a caso criadas lançam luzes sobre um problema que é chave para o desenvolvimento de qualquer sistema: A especificação dos requisitos a que ele deva atender. Num congresso sobre sistemas operacionais realizado em 1985, registrado por

SVOBODOVA [2], reconheceu-se que pouco se sabia a respeito dos requisitos reais para a construção de um Sistema Operacional Distribuído. Além da discussão de temas de interesse técnico específicos, o conhecimento das necessidades a que os futuros sistemas operacionais devam atender foi assunto de um polêmico painel por um motivo simples: o direcionamento das pesquisas realizadas.

Este direcionamento constitui, na verdade, o segundo motivo para a realização dessa tese. Outra vez a motivação vem da existência dos "casos vivos" acima mencionados. Agora passa-se a encará-los, não mais como um objeto para a observação, mas como um campo para a aplicação dos conhecimentos adquiridos com o seu estudo.

Com relação à motivação técnica para a escolha desse tema como assunto de pesquisa, poderiam ser mencionados os seguintes tópicos específicos: a organização de um ambiente com as características específicas do sistema em estudo; a identificação dos serviços necessários; a definição dos serviços e protocolos para a realização do sistema buscado; estudo dos trabalhos de padronização desenvolvidos. Alguns desses tópicos serão abordados no decorrer dessa tese.

Outra motivação técnica existente diz respeito à natureza do assunto de pesquisa. Por ser um assunto de caráter muito abrangente, trabalhos direcionados na área forneceriam campo para pesquisas numa série de áreas afins. Essas pesquisas, outra vez, teriam motivadores práticos para serem realizadas. Por exemplo, enquanto um projeto de pesquisa em área de sistemas, trabalhos em engenharia de software poderiam ser levados conjuntamente. Ferramentais novos poderiam ser desenvolvidos - compiladores. Novas arquiteturas de sistemas poderiam ser pesquisadas. Assim também novos sistemas operacionais que atuassem dentro do novo ambiente poderiam ser desenvolvidos. Haveria um campo de estudo para protocolos e implementações de protocolos

formidável. Enfim um projeto de pesquisa abrangente poderia fazer convergirem esforços de várias áreas, dando o elemento prático e a direção fundamentais.

## 1.3) A CONTROVÉRSIA SOD x SOR

Encarando-se o problema de integração de sistemas heterogêneos, parecem existir, a princípio, duas formas de se solucionar o problema. Uma primeira seria a de esquecermos os sistemas operacionais já existentes e construir sistemas operacionais novos a partir do suporte básico do hardware. A justificativa parece simples: Como, na verdade, os sistemas operacionais existentes não foram construídos prevendo sua integração com outros elementos, deve ser impossível conseguí-la. Por esse ponto de vista, esta forma de abordagem tem similar naquela de se acabar com a pobreza matando-se todos os pobres.

O outro tipo de abordagem implicaria na criação de mecanismos que não só preservassem a individualidade de cada elemento constituinte, mas que procurassem valorizar essa individualidade. Isso devido ao reconhecimento de que a preservação do ambiente natural de cada sistema traz consigo um fator chave para um sistema de computação, paradoxalmente muito importante nesse mundo de rápida evolução: O aproveitamento de software já existente e da cultura de uso de cada sistema. Uma solução que ignore a importância deste fato estará fadada ao ostracismo. Analisando-se aspectos de custos vê-se que essa segunda abordagem tem três componentes atenuadas em relação à primeira alternativa, ou seja, a da introdução de um novo Sistema Operacional. A primeira é a preservação do investimento em software realizado. Há algum tempo o peso principal dos custos de um sistema se deslocou do Hardware para o Software do mesmo.É interessante observar que faz parte dos requisitos de análise de viabilidade de projetos e de lançamento de novos produtos de muitas empresas o

estudo do impacto que uma nova linha - isso de um mesmo fabricante - tem sobre a base de software desenvolvida. Se por acaso pouco ou nada pode ser aproveitado qualquer continuidade do mesmo fica seriamente comprometida. É fundamental que os investimentos já realizados no setor de software sejam preservados.

A segunda componente que é atenuada é a que está ligada ao treinamento de recursos humanos. Uma vez que o objetivo é valorizar o ambiente de operação original de cada subsistema componente, o usuário deve sentir tão pouco quanto possível o impacto da integração ao novo ambiente. O aprendizado de uso do novo sistema pode ser gradativo não criando choques que poderiam inviabilizar a implantação do novo sistema. Do lado do usuário parece pouco provável que ele venha a abrir mão de facilidades que ele tenha disponível em favor de um novo sistema único. Deve-se mesmo observar que quando se fala em usuário, na verdade quer-se dizer uma comunidade de usuários, que certamente farão coro contra perturbações em seu ambiente de trabalho.

A terceira componente que seria atenuada com relação a uma implementação "absolutamente inovadora", seria a de construção do novo sistema. Imagine-se o que é construir um sistema homogêneo para um ambiente com suporte heterogêneo. Observa-se que ainda assim lidar-se-ia com o problema de heterogeneidade - só que desta vez a nível de hardware. O problema fica, com relação apenas a este ponto, em se saber o que é mais complicado: trabalhar com um sistema menos complexo porém menos maleável e mais próximo ao hardware ou com um sistema mais complexo porém mais maleável - a maleabilidade aqui é com relação as facilidades de programação. A nós nos parece mais fácil, e portanto demandando menos esforço, essa última. Outro problema que ocorre com a primeira seria o de se querer mexer com uma parte via de regra mais carente de documentação: os hardwares individuais e particulares de cada sistema.

Na verdade essa controvérsia toda resulta muito mais da não compreensão da realidade já mencionada: a de que o mundo da informática tem caminhado para a heterogeneidade. É preciso que se busquem soluções para a operação dentro desse contexto.E mais, soluções que integrem de fato os subsistemas. Assim como se reconheceu um dia a necessidade de construção de sistemas que estivessem preparados para evoluir - ou seja, serem alterados - é necessário que se tenha uma compreensão análoga. Num primeiro passo deve-se aprender como viabilizar a convivência com a heterogeneidade. Num segundo estágio deve-se estar aprendendo como projetar sistemas que se integrem a ambientes heterogêneos. Fica mesmo fácil imaginar o atrativo que teria um novo sistema com essa característica: aquele que oferece todas essas novas funcionalidades e se comporta assim e assim dentro de um contexto existente.

A área de pesquisa em sistemas operacionais distribuídos também é vasta. Exemplos de trabalhos desenvolvidos nessas áreas temos em [1], [3] e [4]. Ela não tem o objetivo de resolver o problema de heterogeneidade existente, mas sim o de ser o sistema operacional de toda uma sorte de arquiteturas que estão surgindo. Existem problemas técnicos específicos que precisam ser ainda resolvidos ou mais bem elaborados. Isso sem falar a respeito da futura necessidade de sua integração com sistemas já existentes. Então aí seria a hora dele exibir características que suavizassem essa entrada. Certamente existem assuntos que são abordadns nas duas linhas de pesquisa. O que varia, no entanto é o enfoque dado ao problema. Por exemplo, a nomeação dos objetos no sistema, a distribuição de carga, a duplicação de objetos, e mais uma série de outros assuntos são estudados em uma e outra área, mas certamente com condições de contorno e objetivos diferentes. A pesquisa realizada em uma área acaba por gerar subsídios para a outra.

## 1.4) ANÁLISE DE TRABALHOS NA ÁREA

A literatura existente reflete em parte a confusão mencionada no ítem anterior quanto aos objetivos de um sistema operacional de rede. Integrar-se sistemas heterogêneos; adequação de sistemas existentes a situações novas; criação de novos sistemas. Esse tatear de caminho, apesar de inicialmente provocar uma dispersão de esforços, acaba por ter conseqüências interessantes.Ao coletarmos o material gerado até aqui, notamos que uma série de tópicos já foi de alguma forma abordada permitindo estabelecer um ponto de partida consistente.

O trabalho de FORSDICK [5] é um trabalho interessante realizado já há algum tempo. O interesse que esse trabalho desperta está relacionado a coerência demosntrada na conceituação sobre a interconexão de Sistemas através de um SOR. Nesse trabalho são apresentadas duas abordagens para construção de serviços num ambiente formado por elementos computacionais autônomos. Conquanto alguns pontos da comparação já não façam muito sentido, outros são bastante interessantes, destacando-se a observação final quanto a tecnologia emergente dos computadores individuais.

Os trabalhos realizados na área podem ser divididos em três grupos: um primeiro grupo elegeu o Unix como um sistema padrão e trabalha para dotá-lo de características para a operação em rede. Exemplo desse tipo de abordagem temos em trabalhos publicados por PANZERI e RANDELL [6] sobre o Unix-United, NFS da SUN Microsystems por WALSH et all[7] e o LOCUS por BUTTERFELD e POPEK[8]. Ainda que esse tipo de trabalho não objetive a integração de sistemas heterogêneos, a exceção do NFS, eles são interessantes por abordarem aspectos importantes do problema com relação a mecanismos de transparência e de extensão de uso. Existe um trabalho de adequação do sistema para novas condições de operação.O grau de transparência

que é fornecido a aplicativos e usuários.A compatibilidade com o software já existente. Que partido se pode tirar do novo sistema em termos de incremento na qualidade do serviço - desempenho, a distribuição de carga, duplicação de objetos, confiabilidade, etc.O tipo de alterações que viabilizaram ou viabilizariam a sua implementação. Existe toda uma sorte de conhecimentos que servem para traçar uma diretriz para o trabalho que se pretende encetar.

O segundo grupo de trabalhos procura fazer uma abordagem a partir de uma análise de arquiteturas possíveis para a solução do problema de integração de sistemas heterogêneos. Existem alguns exemplos que, contudo, não possuem continuidade, pelo menos na literatura disponível. Um trabalho que se encaixa nesse grupo é o "DCNA Higher Level Protocols" por TODA [9]. Esse trabalho procura modelar o ambiente heterogêneo de uma rede de computadores. Basicamente existe um modelo subdividido em outros três que buscam reproduzir os seguintes aspectos de um sistema computacional: os aspectos relativos aos recursos diponíveis - os tipos de estações participantes, os recursos ligados a ela; os aspectos relativos à gerência do sistema - o controle de acesso, diagnóstico, inicialização, manutenção do status; os aspectos relativos ao uso - como se fazer para se acessar os recursos disponíveis. É uma visão consistente que procura endereçar o problema de se criarem interfaces para a construçao de protocolos que viabilizariam a integração dos sistemas participantes.

Duas questões se colocam: a primeira é a forma como os procedimentos eventualmente elaborados são agregados aos sistemas participantes. O ponto onde a partir do qual passamos a considerar os problemas como detalhes de implementação é sempre sujeito a questionamentos. Este é porém, na nossa forma de ver, um problema crucial, uma vez que está ligado à natureza do trabalho que se vai poder desenvolver e o que se pode esperar do sistema integrado. A segunda questão diz respeito ao seguinte problema: uma vez

que se modelou o sistema identificando-se explicitamente tipos de elementos participantes, o que ocorre com o mesmo quando novos elementos, que não se encaixarem nos modelos disponíveis, vierem a compor o sistema. É claro que todo e qualquer projeto cai, uns mais cedo que os outros, em obsolecência. Mas a questão é interessante, uma vez que o modelo não parece admitir, confortavelmente, por exemplo, microcumputadores.

Um outro trabalho pertencente a esse grupo é o realizado por TSAY e LIU [10]. O que se destaca nesse trabalho é a idéia de se construir um sistema que pode ser globalmente administrado a partir da interligação de forma cooperativa dos sistemas participantes. O trabalho desenvolve ainda alguns outros conceitos interessantes - como o de se criar, a partir dos próprios mecanismos de compartilhamento, controles para regular o acesso aos objetos. Para fornecer um ambiente uniforme para a construção do MIKE, o próprio hardware de comunicação é considerado. Esta é uma idéia interessante contudo de aplicação questionável, que por certo não elimina o problema de se interagir com cada Sistema Operacional participante.

O terceiro grupo de trabalhos é o daqueles que procuram estudar tópicos específicos da integração de sistemas, sejam eles propriamente heterogêneos ou não. Dentre esses trabalhos destacaríamos, até porque será abordado com mais detalhes em capítulo a parte, o trabalho de FALCONE [11] que aborda o problema da definição de mecanismos para a construção e a evolução de um sistema com as caracterísitcas que vamos descrever.

## 1.5) REALACIONAMENTO AOS TRABALHOS DE PADRONIZAÇÃO ISO-OSI

Vamos aqui analisar quais as relações entre os trabalhos de padronização desenvolvidos na ISO para a Interconexão de sistemas abertos e os estudos para o desenvolvimento de um Sistema Operacional de Rede. Essa análise é interessante, uma vez que as duas formas de enfoque estão dedicadas a uma mesma causa: a integração de sistemas heterogêneos.

Primeiro é necessário que se entenda corretamente os objetivos dos trabalhos ISO-OSI. Para isso nada melhor que a própria definição de objetivos constante nos documentos da organização [12]: "OSI is concerned only with interconnection of systems. All other aspects of systems which are not related to interconnection are outside the scope of OSI". Por essa afirmativa poderíamos pensar que os objetivos na integração fossem mais modestos,contudo logo em seguida o mesmo documento acrescenta: "OSI is concerned not only with the transfer of information between systems,i.e.,transmission,but also with their capability to interwork to achieve a commom (distributed) task.In other words, OSI is concerned with the interconnection aspects between systems, which is implied by the expression 'systems interconnection'".Esta última colocação não deixa dúvidas quanto à amplitude dos objetivos nos trabalhos ISO-OSI.

A própria forma de abordagem desenvolvida incialmente nos trabalhos ISO-OSI indica uma certa dualidade de objetivos entre os dois enfoques. Primeiramente foi desenvolvido um modelo - o modelo de referência ISO-OSI - que define uma base comum para o desenvolvimento dos trabalhos. De acordo com essa metodologia, primeiro são considerados os vários aspectos dos "sistemas reais" com o

fito de se apreender os aspectos envolvidos na sua interconexão. Após essa fase, apenas os aspectos externos seriam considerados para o seguimento dos trabalhos de padronização. Assim, qualquer sistema que externamente se comportasse de acordo com o sistema real especificado seria capaz de integrar-se ao ambiente ISO-OSI.

Esta fase de busca de elementos para a definição de um modelo constitui uma parte fundamental para o sucesso ou não dos trabalhos de padronização, uma vez que, ao se construir um modelo inadequado, os trabalhos desenvolvidos a partir do mesmo teriam sua aplicabilidade automaticamente comprometida. A própria dualidade de objetivos contida no documento, conforme a citação, dá margem a questionamentos quanto à sua plena realização. O primeiro trecho diz respeito ao objetivo primeiro do documento citado, ou seja, a construção do modelo de referência. Enquanto o segundo trecho diz respeito aos trabalhos de padronização como um todo. Ora, apesar de não ser objetivo do modelo definir como os sistemas implementariam os protocolos e serviços definidos, parece que não se extraiu adequadamente algumas caracterísitcas básicas dos "sistemas reais" com vistas aos objetivos mais amplos definidos. Na medida em que se identifica toda uma sorte de funções, algumas considerações deveriam ter sido feitas a respeito de como os "sistemas reais" desenvolvem essas mesmas funções. Isso porque, uma vez que se deseja uma integração mais ampla, ainda que não se queira definir como se implementam determinadas funções, vão se criar conflitos com as implementações existentes de uma série delas, individualmente, nos sistemas envolvidos.

Como foi anteriormente mencionado, o ambiente de interação entre um (programa) usuário é basicamente aquele definido pelo Sistema Operacional. Do ponto de vista de cada sistema real participante, qualquer novo trabalho a ser desempenhado pela estação deve ou alterar ou se sujeitar às regras do ambiente. Isso com impactos - leia-se custos - tanto maiores quanto sejam as alterações decorrentes.

Há uma certa corrente que vê os trabalhos de padronização ISO-OSI como um mecanismo para se conseguir a homogeneização dos sistemas computacionais. Raciocínio: Uma vez que todo mundo vai refazer tudo de acordo com as normas então teremos sistemas homogêneos. Isso não é o objetivo dos trabalhos de padronização e além do que, pelas razões já expostas, não é nem mesmo viável. Para se advogar a homogeneização tem sido levantado que ainda que não seja objeto específico dos trabalhos ISO-OSI, existe sempre um outro subcomitê que estude o tal assunto com vistas a normalização, por exemplo, LININGTON [13] faz comentários a esse respeito. NOTKIN [14] faz alguns questionamentos quanto a essa ótica homogeneizadora de se encarar os trabalhos de padronização. Na verdade o que se termina padronizando é uma série de opções. Os próprios trabalhos de padronização ISO-OSI formam um conjunto fabuloso de opções. Encarar os trabalhos de padronização como elementos homogeneizadores e se criar a discussão em torno do assunto parece uma situação análoga àquela da disputa entre pesquisar sistemas operacionais de rede e sistemas operacionais distribuídos. Ambas fruto da incompreensão de objetivos buscados.

Para a consecução dos objetivos amplos definidos anteriormente teria sido necessário considerar os aspectos de ambiente dos sistemas envolvidos. Aí surge o primeiro ponto de contato claro entre a padronização e o estudo de Sistemas Operacionais de Rede. Esse estudo fornece, senão a solução pelo menos os subsídios para a compreensão dos problemas pertinentes a integração de sistemas heterogêneos. Nesse sentido, o ambiente característico dos sistemas participantes não poderia ter escapado à definição do modelo de referência.

Uma outra forma de se analisar as relações existentes entre os dois enfoques é a da via em sentido contrário, ou seja, o uso dos protocolos ISO-OSI para

viabilizar os aspectos de interconexão dos sistemas envolvidos. No estágio atual os protocolos definidos cumprem perfeitamente esse objetivo. Conforme já mencionado, enquanto área de aplicação, o assunto dessa tese fornece um campo interessante para a aplicação dos trabalhos de padronização.

## 1.6) PRINCIPAIS CONTRIBUIÇÕES

Como principais contribuições desta tese podemos destacar os seguintes pontos:

- A organização conceitual dos elementos de um sistema operacional de rede.

- A identificação da necessidade de se recriar o conceito de usuário de um sistema heterogêneo em rede.

- A descrição do interrelacionamento de entidades autônomas de forma cooperativa.

- A definição de mecanismos para viabilizar o controle de acesso no sistema.

- O estudo de mecanismos para o desenvolvimento dos serviços no sistema de forma gradual.

## 1.7) ORGANIZAÇÃO DA TESE

No capítulo seguinte abordamos aspectos conceituais do sistema: o que é um sistema operacional de rede; os elementos participantes; as funções por eles desempenhadas; o conceito de transparência. No terceiro capítulo é analisado o problema de administração do sistema com enfoque na manutenção do conjunto de usuário e dos elementos que regulam o seu acesso aos recursos do sistema. No quarto capítulo estudamos a proposta de mecanismos para a construção dos serviços e acesso aos mesmos encontrada em [11]. Resultados de uma aplicação experimental dessas idéias são apresentados. No quinto capítulo, finalmente, apresentamos conclusões sobre o trabalho e propostas para a seqüência do mesmo.

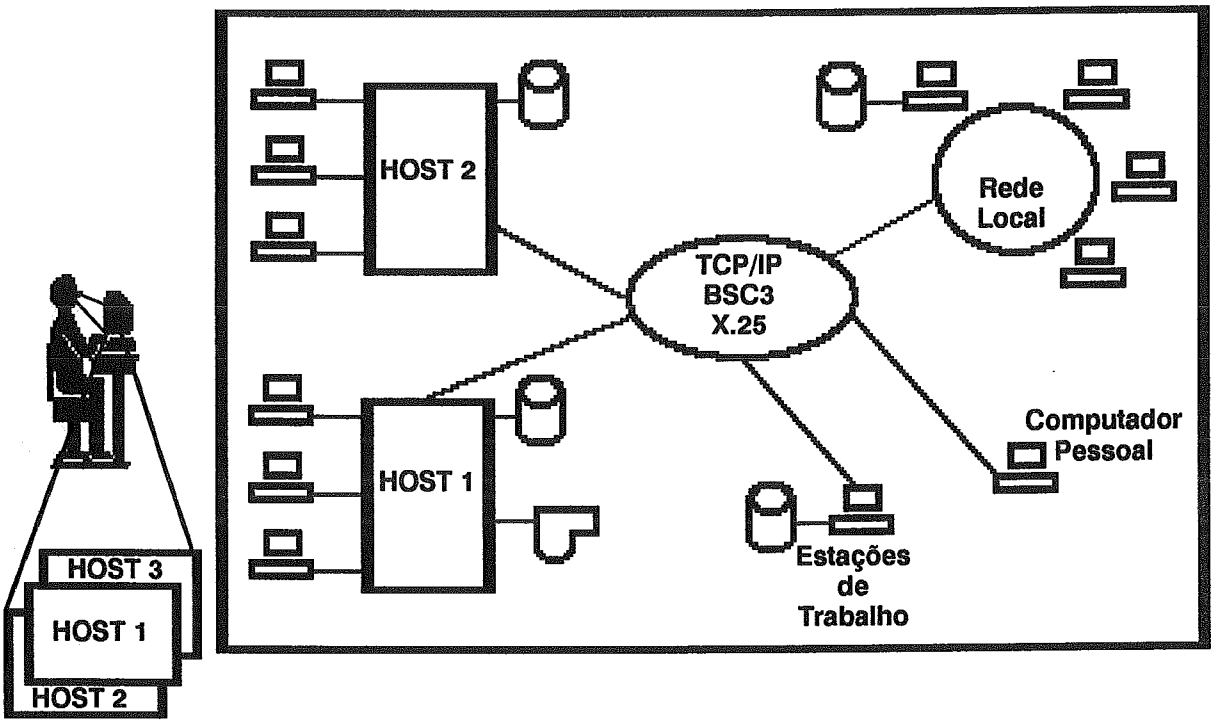


**Fig.1 Ambiente heterogêneo com mecanismos de interconexão**

# CAPÍTULO II

# ASPECTOS CONCEITUAIS DE UM SOR

Trataremos agora de aspectos conceituais importantes do sistema que procuramos definir. Para tanto dividiremos o capítulo em três partes. Primeiramente caracterizaremos o Sistema Operacional de Rede por um conjunto de assertivas relacionadas aos requisitos que ele deve atender, a sua composição e aos mecanismos de integração utilizados. Este conjunto faz a caracterização básica do sistema, podendo ser considerado como uma declaração de princípios que um SOR deva seguir. A segunda parte descreve os elementos participantes procurando caracterizá-los sob ponto de vista funcional. Por último trataremos de um conceito fundamental, ligado principalmente ao uso do Sistema: transparência. Resumindo-se, o SOR é enfocado sob três ângulos: o dos requisitos a atender; o dos tipos de elementos participantes; e finalmente o da forma de uso do sistema.

## 2.1) CARACTERIZAÇÃO DE UM SOR

Passamos a listar a seguir os requisitos básicos que um SOR deve atender. Alguns desses requisitos, se levados individualmente às últimas conseqüências, são mutuamente contraditórios. Alguma solução de compromisso deve ser adotada quando da realização do sistema.

### 2.1.1) COMPOSIÇÃO HETEROGÊNEA

Apesar de já termos mencionado o aspecto de heterogeneidade, vamos estabelecer mais precisamente o seu significado e suas implicações. O primeiro aspecto de heterogeneidade diz respeito à heterogeneidade de hardware. Quer-se com isto dizer que os sub-sistemas componentes possuem não apenas elementos de hardware distintos, mas que

possuem também potencialidades dissimilares. Umas estações são inerentemente mais limitadas que outras. Uma estação possui, por exemplo, dez vezes menos memória que uma outra, ou dez vezes menos capacidade de processamento que uma outra, etc. Ou seja o sistema é, a partir de seus elementos componentes, desbalanceado. Em que pese a obviedade de tal afirmação, é ela contudo essencial quando se pensa na distribuição de tarefas dentro do sistema.

Outro aspecto relevante de heterogeneidade diz respeito às diferenças de sistemas operacionais dos elementos constituintes. O principal fato nesse aspecto de heterogeneidade não são tanto os bits e bytes, mas as diferenças conceituais básicas nas definições dos objetos de cada sistema operacional. Por exemplo, o que é um usuário, um arquivo, etc.

O grau de convivência pacífica e produtiva que se pode obter de um sistema heterogêneo é sempre uma questão em aberto. Isto deriva do balanceamento nas soluções de compromisso adotadas. O ganho de produtividade que se obtém com uma forma mais coesa de operação deve ser compatível com o esforço dispendido para tanto.

### 2.1.2) PRESERVAÇÃO DE AMBIENTE

Este princípio tem relação direta com a escolha da forma de se resolver o problema de integração de sistemas heterogêneos. No primeiro capítulo já se discorreu sobre as razões para tal escolha. Aqui consideraremos suas implicações. A primeira é que devemos encarar cada possível participante como uma unidade vista a partir de seus recursos e limitações de seu sistema operacional. Devemos definir um patamar adequado que cada tipo de estação deva atender. No entanto, devemos estar cientes de que se a mesma originalmente não o fizer, uma camada de software deverá prover o nivelamento adequado. Quão mais elaborado for o patamar mínimo para uma estação participar do sistema

maior será o esforço requerido para estações menos complexas. Outra implicação é que a estrutura do software deve destacar esses requisitos de tal maneira a permitir qua sua implementação seja modular.

Para se atender a esse princípio, o trabalho inicial de estudo de casos para se compor um modelo para o sistema deve ser amplo. Obteríamos dessa forma a identificação dos conceitos básicos dos sistemas existentes. A esse respeito é bom lembrarmos que há não muito tempo atrás discutia-se conceitos que hoje temos por certo como a necessidade de diretórios, arquivos, etc.

### 2.1.3) CAPACIDADE DE EVOLUÇÃO

Num sistema com o qual se pretende a interligação de elementos heterogêneos, este requisito é duplamente importante. Por um lado deseja-se poder agregar novas estações ao sistema à medida que as mesmas vão surgindo. O que isso traduz é a capacidade do sistema para absorver novos conceitos que eventualmente venham a surgir. Novas versões devem poder ser implantadas minimizando-se o esforço dessa atualização.

O outro lado é que para sua própria implantação, o trabalho deve poder ser realizado paulatinamente. Isso traz custos reduzidos de implantação permitindo ainda priorizar o trabalho atendendo a situações mais prementes ou de maior disponibilidade.

### 2.1.4) GRAU DE INTEGRAÇÃO VARIÁVEL

Devemos buscar patamares mínimos para a integração de novos elementos. Isto decorre de que a integração de um elemento em particular só se torna viável se puder ser feita a um custo que o usuário se disponha a pagar por ela. Assim devem-se criar condições para que algum proveito possa ser tirado a partir de condições mínimas de operação obtidas. A

relação custo/benefício no grau de integração realizado deve ser aceitável, ainda que uma determinada estação não esteja "completamente integrada ao sistema".

Isso se reflete também na escolha dos pontos para a realização de determinado serviço. Deve-se ter em mente que uma distribuição de tarefas muito liberal pode inviabilizar a participação de novos elementos. Por exemplo, poderíamos optar por esquemas totalmente descentralizados para a implementação de determinado serviço. O resultado disso poderia ser uma carga excessiva para alguns tipos de estações. Não queremos dizer que a opcão diametralmente oposta - a centralização -seja a aconselhável. Mas sim, que alguma forma de organização, que permita uma distribuição racional das tarefas, deva ser empregada. Por exemplo, a distribuição de determinadas tarefas entre sistemas potencialmente equivalentes, o que permitiria tirar proveito do sistema distribuído sem o malefício mencionado.

### 2.1.5) PRESERVAÇÃO DA QUALIDADE DE UM SERVIÇO

Um sistema operacional de rede deve surgir como elemento de integração de recursos que originalmente estão dispersos e desagregados num determinado ambiente. Os efeitos que ele deve trazer sobre o ambiente devem ser aditivos tanto em termos de qualidade - melhora de características gerenciais, por exemplo - quanto de funcionalidade - acesso a novos recursos anteriormente indisponíveis a uma série de usuários no ambiente. Um determinadn elemento que venha a participar deste ambiente não deve ter degradada a qualidade de serviços que ele antes prestava a seu próprio conjunto de usuários. Essa degradação pode tomar a forma de uma piora em termos de desempenho, por exemplo, ou o que pode ser eventualmente mais crítico, uma fragilização de seus mecanismos de controle de acesso. As opções feitas no projeto não devem contribuir diretamente para essa degradação de qualidade. É

evidente, por exemplo, que uma má configuração de um sistema pode sempre degradá-lo. Sobretudo pelo fato do mesmo estar exposto a situações novas. Tal seria o caso do aumento do número de requisições em decorrência de sua disponibilidade a um maior número de usuários. Contudo isso seria um problema de configuracão e administracão do sistema e não um problema específico do SOR. Mais adiante, quando detalharmos melhor determinadas partes do SOR, ficará mais clara a aplicação desse princípio.

### 2.1.6) PRESERVAÇÃO DA AUTONOMIA LOCAL

De certa forma esse princípio é um corolário do anterior e do segundo, uma vez que trata especificamente da preservação de contexto de operação e da manutenção da qualidade do serviço de gerência sobre um conjunto de objetos que um sistema tenha sob sua guarda.

Este princípio ressalta o objetivo de não se reduzir a disponibilidade do sistema diante de falhas que por ventura venham a ocorrer, seja em determinados centros privilegiados, seja em enlaces de comunicação dos quais o sistema global pudesse depender para a sua operação. O fato de darmos o controle em última instância a administração local não quer dizer que se venha a prescindir de outros mecanismos que o sistema traga consigo. Isto porque o fato de estar conectado a um sistema mais amplo traz realidades novas com as quais o sistema original poderia não estar preparado para lidar. Assim, uma estrutura com níveis para controle seria a forma mais apropriada, que garantisse pelo menos o tratamento das condições novas a que um elemento participante estaria sujeito.

### 2.1.7) ADEQUAÇÃO DOS COMPONENTES

O uso de um SOR para a integração de subsistemas autônomos pode muita das vezes conduzir a situações de conflito entre a obtenção de um objetivo e características intrínsecas aos componentes ou à arquitetura global do sistema.. Em outras palavras, existem limitações inerentes ao sistema que não deveriam ser resolvidas apenas no escopo do SOR. Algumas dessas limitações podem ser decorrentes da escolha da preservação de cada Sistema Operacional Nativo, o que constitui meta primeira do assunto. A título de exemplo achamos conveniente citar algumas situações.

Situações desse tipo são facilmente exemplificadas em sistemas que integrem microcomputadores. Normalmente elas estão associadas a políticas de controle de acesso a informação. Por exemplo, pode-se querer resolver através de mecanismos de software a possibilidade de um determinado usuário copiar para si arquivos a que por ventura ele possa ter acesso em virtude da estação pela qual ele interage com o sistema possuir unidades de disco removíveis. Outro exemplo seria o fato de se criarem estações com sistemas de arquivos compartilhados e não se ter em atenção que seu usuário imediato tem poder de vida e morte sobre a estação - botão de liga/desliga.

Esses tipos de situações podem ser facilmente equacionados dentro da própria arquitetura do SOR. A segurança de qualquer sistema, por exemplo, é uma questão que transcende ao domínio de controle do sistema operacional. Outros aspectos como desempenho, custo, políticas de organização, etc, devem ser conjuntamente analisados para a adoção de uma solução adequada.

## 2.2) A ORGANIZAÇÃO DO SISTEMA

### 2.2.1) SERVIÇOS

Dois fatores principais criam os atrativos para a elaboração de um SOR. O primeiro é a possibilidade de criação de mecanismos para a gerência do ambiente com seus usuários e sub- sistemas computacionais. O outro fator é a disponibilização dos recursos dispersos a um número maior de usuários. Desse segundo fator é que nos ocuparemos agora.

Serviço é uma palavra que vem sendo utilizada em diversas áreas para a caracterização das atividades realizadas por um elemento em favor de outro. Assim é, por exemplo, na área de padronização de protocolos, onde o conceito de serviço engloba as atividades que as entidades de uma determinada camada realizam para as da camada imediatamente superior.

Num SOR define-se serviço como o conjunto de atividades que um determinado componente, detentor de recursos específicos, realiza em favor de um conjunto de "usuários" do sistema. Para que um determinado serviço possa ser acessado, é necessário que uma série de ações segundo protocolos específicos sejam realizadas.

Cada serviço é, em princípio, completamente independente dos demais. Um componente pode fornecer um ou mais serviços a partir da disponibilidade de recursos e do desejo de oferecê-los a outros usuários do sistema global. A independência entre o fornecimento dos diversos serviços possíveis contribui para a modularidade do sistema. Vale observar que essa forma de organização está de acordo com a que vem sendo adotada em sistemas operacionais convencionais. Nestes, existe um núcleo mínimo - onde se procuram reunir as "primitivas básicas" - que fornece

suporte a um grupo de processos de sistema que implementam as formas de acesso aos recursos da estação.

### 2.2.2) COMPONENTES

Foi mencionado anteriormente a necessidade de se fazer um levantamento dos sistemas existentes com vistas a se modelar adequadamente o sistema, em virtude da preservação de contexto buscada. Nesse sentido poderíamos agrupar os equipamentos segundo duas classificacões. A primeira identifica características atuais do sistema e a segunda identifica papéis a serem desempenhados pelos sub-sistemas componentes dentro do SOR. Consideraremos essas duas clasificações nos dois próximos itens.

#### 2.2.2.1) TIPOS DE SUBSISTEMAS COMPONENTES

A grosso modo poderíamos classificar os subsistemas componentes, quanto a sua capacidade administrativa, em sistemas multiusuários e sistemas monousuários. Preferiu-se adotar este tipo de classificação em virtude de se reconhecer o problema administrativo como de importância capital dentro do contexto de integração de sub-sistemas autônomos.

Os sistemas multiusuários implementam, sob as mais variadas formas, mecanismos que permitem aos mesmos gerenciar um conjunto de usuários que podem acessá-lo. A presença desses mecanismos se fez necessária em decorrência de pontos para acesso simultâneo aos recursos do sistema - os terminais. Num sistema monousuário, apesar de ser teoricamente concebível a existência e a necessidade de mecanismos análogos, na prática tal não ocorreu. A seqüencialidade de acesso aliada à maior simplicidade de programação contribuiram para a eliminação do conceito de usuário - na realidade foi tornado implícito. O conceito de usuário será analisado adiante mais detidamente.

Voltando à nossa caracterização dos sistemas, vamos identificar com maiores detalhes cada um dos dois tipos de sistema. Além da existência do controle de um conjunto de usuários, o sistema multiusuário distingue-se pelas seguintes características:

- Sistemas individualmente mais caros

- Maior complexidade do sistema tanto a nível de hardware quanto de software.

- Existência de suporte para a multiprogramação - existem primitivas no sistema que permitem desenvolver atividades concorrentes (Criação e destruição de processos, mecanismos de sincronização, etc).

- Recursos físicos do sistema são mais abundantes - processamento, memória primária, memória secundária - e o software de seu SO arbitra a disputa por seu acesso.

- Distanciamento do usuário do sistema de computação - a cultura de uso geralmente restringe-se a identificação de um terminal com a interface fornecida via aplicativos.

Os sistemas monousuários poderiam ser caracterizados pelos seguintes pontos:

- Sistemas de baixo custo

- Menor complexidade do sistema tanto a nível de hardware quanto de software.

- Via de regra não possui suporte para a multiprogramação.

- Recursos físicos geralmente mais escassos, não existindo concorrência para seu acesso.

- A cultura de uso é a da interação direta do usuário com o sistema de computação.

Essa caracterização dá bem uma idéia da problemática envolvida no assunto. De uma forma geral, tem-se que construir um sistema que lida com os seguintes antagonismos:

- O sistema deve lidar com aspectos de custo díspares.

- O esforço de desenvolvimento é grande, uma vez que deve ser feito sobre sistemas de graus de complexidade muito dissimilares.

- O ambiente de programação é absolutamente distinto em termos de suporte.

- Recursos a disposição quantitativamente diferentes. É bom lembrar que não foram poucos os casos onde se verificou, após a construção do software de sistema, contrário a todas as previsões, que poucos recursos ficaram a disposição de aplicativos. Estes últimos são a razão de ser de qualquer sistema de software básico.

- O conjunto de usuários do sistema global é bastante heterogêneo. Possui cada um a respeito do outro impressões as mais discrepantes.

É necessário ainda fazermos uma distincão entre os tipos de sistemas multiusuários. Existem, por questões de custo, tipo de aplicação, etc, sistemas com diferentes níveis de segurança e disciplinas de acesso. Considerando-se que os sistemas são também guardas de objetos a eles confiados e que possuem facilidades diferenciadas que auxiliam na implementação de políticas de controle de acesso a esses objetos, o fato desse sistema estar inserido num contexto mais abrangente não deve, de acordo com um princípio já mencionado, degradar o grau de segurança com

que o referido sistema executa essa função. Mecanismos devem ser implementados dentro do SOR que garantam a manutenção do nível de segurança dos elementos participantes. Contudo, é bom lembrar que a solução desse problema vai além dos aspectos meramente ligados ao software do SOR.

2.2.2.2) TIPOS DE FUNÇÕES

Uma segunda forma de classificar o conjunto de elementos que participarão do sistema diz respeito à função que esperamos que cada um deles venha a desempenhar dentro do mesmo. Uma vez que complexidades, custos e recursos são bastante dissimilares, devem ser definidos papéis que cada um dos componentes possa desempenhar a contento.

Um primeiro tipo de papel que um determinado elemento pode desempenhar dentro do sistema do ponto de vista do SOR é o de usuário. Aqui a parte que lhe cabe é a de usufruir dos "novos recursos" que lhe foram disponibilizados. Como se verá mais adiante a forma do usuário ganhar acesso a um determinado recurso pode ser algo muito simples, como o fato de ser o primeiro a ser apresentado pode fazer supor, como tornar-se um trabalho bastante árduo. Em todo caso esse trabalho terá sempre a guiá-lo a luz da relação custo/benefício.

Um segundo papel que um elemento pode desempenhar dentro do SOR e' o de constituir um fornecedor de serviços ao ambiente. Aqui o papel desempenhado é exatamente o oposto ao anterior. Uma estação entra no sistema com o fito de tornar disponíveis ao sistema capacidades e/ou características que lhe sejam próprias.

Um terceiro e último papel em nossa classificação é aquele em que um elemento participa do sistema global usufruindo e fornecendo recursos do mesmo.Seria a forma mais completa de integração.

Com essa classificação maleável podemos conceber um determinado componente desempenhando cada um desses três papéis para serviços distintos. Pretende-se assim deixar bem claro o que se deseja realizar com o Sistema Operacional de Rede: integrá-lo com recursos e conhecimentos desde já disponíveis a um custo que o viabilize. Isso, evidentemente, de tal maneira que o sistema integrado seja mais produtivo que o mesmo em seu estado anterior.

### 2.2.3) MECANISMOS DE COMUNICAÇÃO

Neste ponto vamos considerar uma peça que é chave, até mesmo por razões históricas, qual seja a comunicação entre os sistemas computacionais. Procuraremos enfocar o assunto sob dois ângulos: o primeiro como elemento viabilizador de sistemas distribuídos. O segundo como o do problema para se conseguir uma uniformização das interfaces e protocolos. Isto é capital uma vez que o tipo ou mecanismo de comunicação utilizado, dada a sua ubiqüidade, pode comprometer a viabilidade do sistema.

#### 2.2.3.1) TIPOS DE ENLACE

Podemos distinguir basicamente dois tipos de enlace: os lentos (tipicamente até 64 Kbits/s) e os rápidos (tipicamente acima de 1 Mbit/s). A distinção se faz assim mais por razões de ordem históricas, sendo realmente arbitrária. Essa divisão, baseada em velocidade do enlace de comunicação, surgiu em virtude dos tipos de aplicação aos quais originalmente se destinaram. Algum tempo atrás só existiam mesmo dois tipos de mecanismos de comunicação: os lentos seriais e os rápidos paralelos. A desvantagem desses últimos era que a distância entre os elementos interligados tinha que ser muito pequena - tipicamente da ordem de metros. As aplicações básicas às quais se destinava a comunicação serial era o acesso a dispositivos de E/S e a

transferência de arquivos. Nessa época efetuava-se, quando muito, um preprocessamento dos dados com sua posterior submissão a uma unidade central onde seria de fato processado.

Com o aparecimento da tecnologia de redes locais e os enlaces seriais de comunicação rápidos, novos tipos de aplicação começaram a se tornar viáveis. Inicialmente o seu uso foi o de meramente permitir o acesso a recursos na época caros. Com o uso das unidades interligadas através de uma rede local todos os elementos participantes ganhavam acesso aos referidos dispositivos. Posteriormente, com a evolução do software, tornaram-se viáveis aplicativos antes somente disponíveis em sistemas multiusuários: o acesso concorrente a bases de dados compartilhadas, correios eletrônicos, etc.

É interessante observar que durante um certo tempo, dada a rapidez da evolução tecnológica que vinha se processando, houve mesmo uma certa confusão dentro dos organismos internacionais que cuidavam dos aspectos de normalização dos protocolos de comunicação. O novo tipo de uso que se vinha fazendo de sistemas interligados tinha peculiaridades a serem devidamente consideradas. A despeito de suas particularidades, havia a necessidade de realizar-se a integração desses sistemas ao ambiente alvo dos trabalhos de padronização. Quando isso foi compreendido, os trabalhos passaram a considerar os novos sistemas também dentro de seu contexto, objetivando uma ampla integração.

### 2.2.3.2) INTERFACE DE COMUNICAÇÃO UNIFORME

Observando-se a diversidade de equipamentos, interfaces, protocolos, etc, disponíveis para a interligação, um problema chama a atenção. Uma vez que um sistema interligado em rede possui naturalmente mecanismos de comunicação os mais diversos possíveis, como se obter

algum grau de uniformidade que torne mais racional a tarefa de se desenvolver programas que operem nesse ambiente? Como em última instância desejamos construir programas de sistemas - SOR - que forneçam o ambiente de programação, temos que conseguir alguma facilidade que reduza o esforço de tal empreitada.

Dois pontos são particularmente críticos. O primeiro diz respeito aos mecanismos locais a cada sistema que permitem o acesso aos serviços de transporte de mensagens. Nessa descrição estão embutidos: a qualidade do serviço fornecido; o tipo de serviço; como é o mecanismo de reporte de situações de exceção; enfim toda a mecânica local que os programas eventualmente fariam uso. O segundo ponto tem relação com a forma com que uma entidade lógica consegue se comunicar com outra no ambiente, ou seja como se consegue localizá-la. A questão fundamental aqui é a de como a entidade de alto nível consegue endereçar uma outra com a qual ela deseja se comunicar.

Em PANZERI e RANDELL [6] é feita uma observação sobre a forma como ocorreu o desenvolvimento dos protocolos de comunicação, que julgamos pertinente. Os autores salientam que uma abordagem "bottom-up" foi utilizada para a definição das interfaces de comunicação: primeiro foram definidos os protocolos e depois então as interfaces. Essas interfaces exibem como conseqüência detalhes ligados à enegenharia do problema. Isto faz com que essas mesmas interfaces, pela complexidade associada, se distanciem de seu objetivo fim, qual seja, o de serem usadas para a comunicação entre programas.É bem verdade que assim tenha ocorrido dada a natureza dinâmica de uma tecnologia emergente. Contudo, talvez já fosse tempo de se direcionar o desenvolvimento para a definição de interfaces de comunicação mais apropriadas aos objetivos da programação. Somente assim é possível a construção e o porte de programas que operem nesse ambiente.

Em que pese a especificação de mais uma interface nesse nível, ela é contudo essencial para tornar o SOR viável. Na referência citada é mostrada uma abordagem "top-down" para o problema de interconexão de sistemas com vistas à construção de aplicações distribuídas. Essa forma de se atacar o problema poderia ser aqui empregada. 2.2.4) TRANSPARÊNCIA

Dentro do leque de possíveis recursos que se vai tornar disponíveis dentro de um ambiente coordenado por um SOR, uma parte considerável dos mesmos possui alguma forma de similar funcional na estação requisitante, isto é, o sistema requisitante já trabalha com objetos análogos. Isso se traduz na existência de uma quantidade considerável de programas que poderiam imediatamente fazer uso dos referidos recursos. Transparência é um conceito que surge então na funcionalidade de um SOR. Ele está associado à possibilidade de determinado sub- sistema, que funcionava anteriormente de forma isolada, continuar operando dentro do novo ambiente. Isto sem sofrer alterações em sua rotina normal de operações, quer no acesso a recursos localmente disponíveis, quer no acesso a recursos dispersos no sistema global. O conceito de transparência está ligado à continuidade de operação, à preservação de contexto.

### 2.2.4.1) NÍVEIS DE TRANSPARÊNCIA

A fim de viabilizar o SOR como um mecanismo que resolva de forma ampla o problema da integração de sistemas heterogêneos é necessário adotarmos posturas abertas quanto aos possíveis tipos de solução. Assim, encarando-se mais abertamente a questão de transparência, vamos fazer algumas considerações. Inicialmente observamos que normalmente a programação em qualquer sistema é estruturada em camadas. Isto é, existem vários pontos onde se estabelecem interfaces de uso. Essas interfaces, que constituem regiões onde existem regras de acesso bem definidas, são elementos naturais para se obter o efeito de transparência. Já que

existe uma linha que estabelece uma descrição comportamental de dois elementos que através dela interagem, é possível substituir um dos dois elementos sem que o outro se aperceba do fato. Trata-se de se tentar enganar o parceiro. Essa tarefa é tão mais simples quanto mais bem estabelecidas forem as regras de interoperação.

A grosso modo, poderíamos identificar dois níveis onde isso poderia ser conseguido. Como buscamos manter a capacidade de operação de programas existentes nas estações que passem a usar serviços e recursos colocados a sua disposição pelo SOR, vamos nos ater àquelas interfaces que permitiriam a esses mesmos programas continuarem a operar normalmente sem alterações relevantes em seus procedimentos. As duas interfaces principais que permitiriam o uso vantajoso da transparência seriam: primeiro a nível de seu sistema operacional; a segunda a interface a nível da linguagem de programação. A primeira interface tem a vantagem de ser global. Uma vez que o sistema operacional de uma estação possui o duplo papel de gerente de acesso aos recursos e disciplinador da sua forma de acesso, a sua interface se torna uma candidata natural. A sua vantagem principal é a de se conseguir o melhor resultado em termos de transparência. Os programas continuariam a operar com compatibilidade a nível de código objeto. Nenhuma modificação precisaria ser efetuada nos programas. Todo ferramental disponível para o desenvolvimento de programas seria exatamente o mesmo. A dificuldade maior para esse tipo de abordagem é a de que, primeiramente, nem sempre é possível efetuar modificações no ponto de entrada do sistema operacional. Outra desvantagem é que às vezes o uso de interfaces não documentadas - não bem definidas - tornam esse trabalho dificultoso.

O segundo ponto onde poderíamos obter o efeito de transparência - a nível de linguagem de programação - permitiria que os programas continuassem a operar com transparência estabelecida a nível de código fonte.

Normalmente bastaria uma operação de "linkedição" ou de susbstituição de um módulo de bibliotecas do sistema operacional para se conseguir o efeito desejado. Em sistemas que disponham desse último recurso, é bom observar que conseguiríamos também o efeito de compatibilidade a nível de código objeto. Uma primeira desvantagem desse tipo de solução é que nem sempre o código fonte está a disposição do usuário. Outra desvantagem é o fato de uma mesma linguagem possuir uma série de versões, cada uma com interfaces eventualmente diferentes. Não obstante, esse tipo de abordagem tem sido alvo de estudos.

A respeito do conceito de transparência talvez seja bom lembrar a discussão de um workshop sumarizada por SVOBODOVA [2]. Lá além de se debater o conceito também se discutiram tópicos como níveis de transparência. Por exemplo, transparência de acesso - os mesmos mecanismos de acesso tanto local como remotamente - transparência de localização - a localização é invisível com relação ao método de acesso - transparência de controle - todas as informações descrevendo um sistema são idênticas para as aplicações - transparência de execução - balanceamento de carga, migração de programas, comunicação entre processos são elementos relevantes. Ainda que se questione essa exarcebação do efeito de transparência - uma vez que para se escolher é preciso distinguir, e dado o nosso estágio atual de software de sistema, muitas escolhas ainda tem que ser feitas de forma não automáticas - o seu uso como elemento para facilitar o uso do sistema foi ponto de concordância.

## 2.3) VISUALIZAÇÃO DO SISTEMA

Ao final desse capítulo e também dos subsequentes vamos procurar mostrar graficamente a arquitetura do sistema que estamos definindo - o SOR. O objetivo é apresentar uma visão "atualizada", destacando-se aspectos estruturais e

funcionais do mesmo que tenham sido abordados em cada capítulo. Assim a figura 2 mostra o mesmo sistema da figura 1 integrado por um SOR. Chamamos a atenção para os seguintes pontos:

- Para a programação do SOR foi adotada uma interface de comunicação uniforme. Não importa qual o "meio físico" utilizado (TCP/IP, X.25, etc), os mecanismos usados pelos programas de sistema são uniformes.

- Cada estação preserva seu ambiente nativo de operação. Os elementos continuam a existir independentemente uns dos outros.

- Do ponto de vista de um "usuário", tem-se acesso aos recursos globais do sistema a que está integrado. No entanto, por efeito de transparência, a visão obtida é aquela típica de seu sistema em particular.

Na figura 3 destacamos os papéis que podem ser desempenhados por cada estação dentro do sistema. Assim uma estação multiusuária - Host 1, por exemplo - oferece e usa diversos recursos do sistema. Uma estação monousuária agrega-se ao sistema apenas como usuária do mesmo.

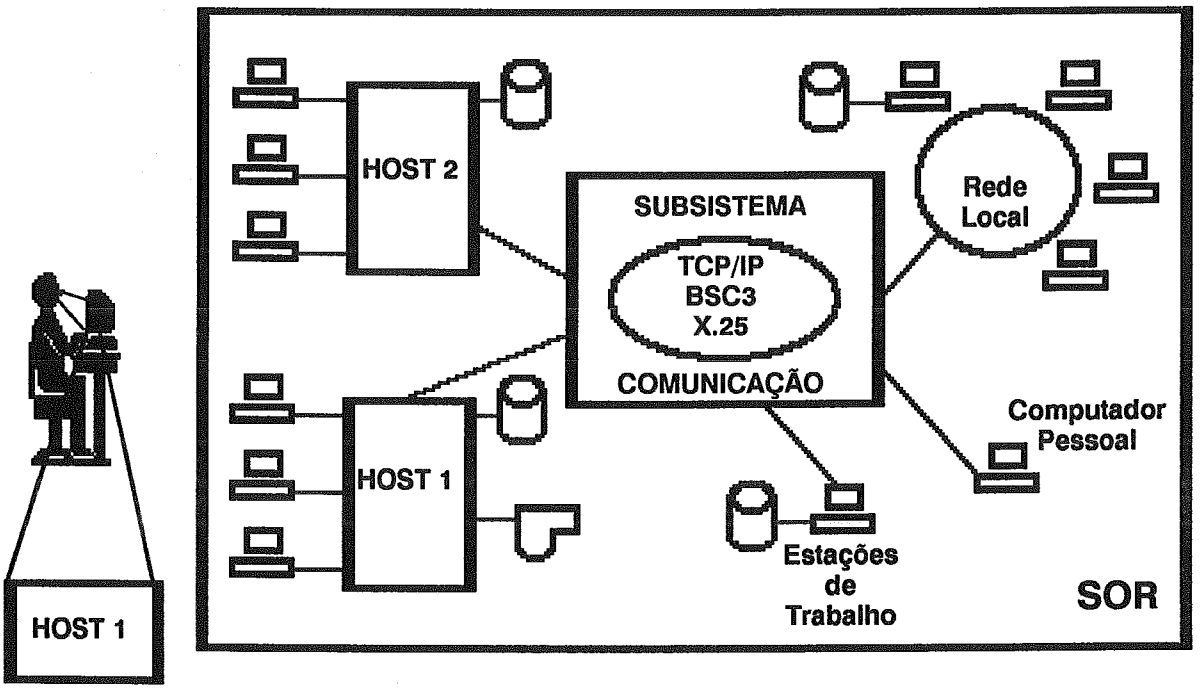


**Fig.2 Sistema Heterogêneo Integrado para SOR**

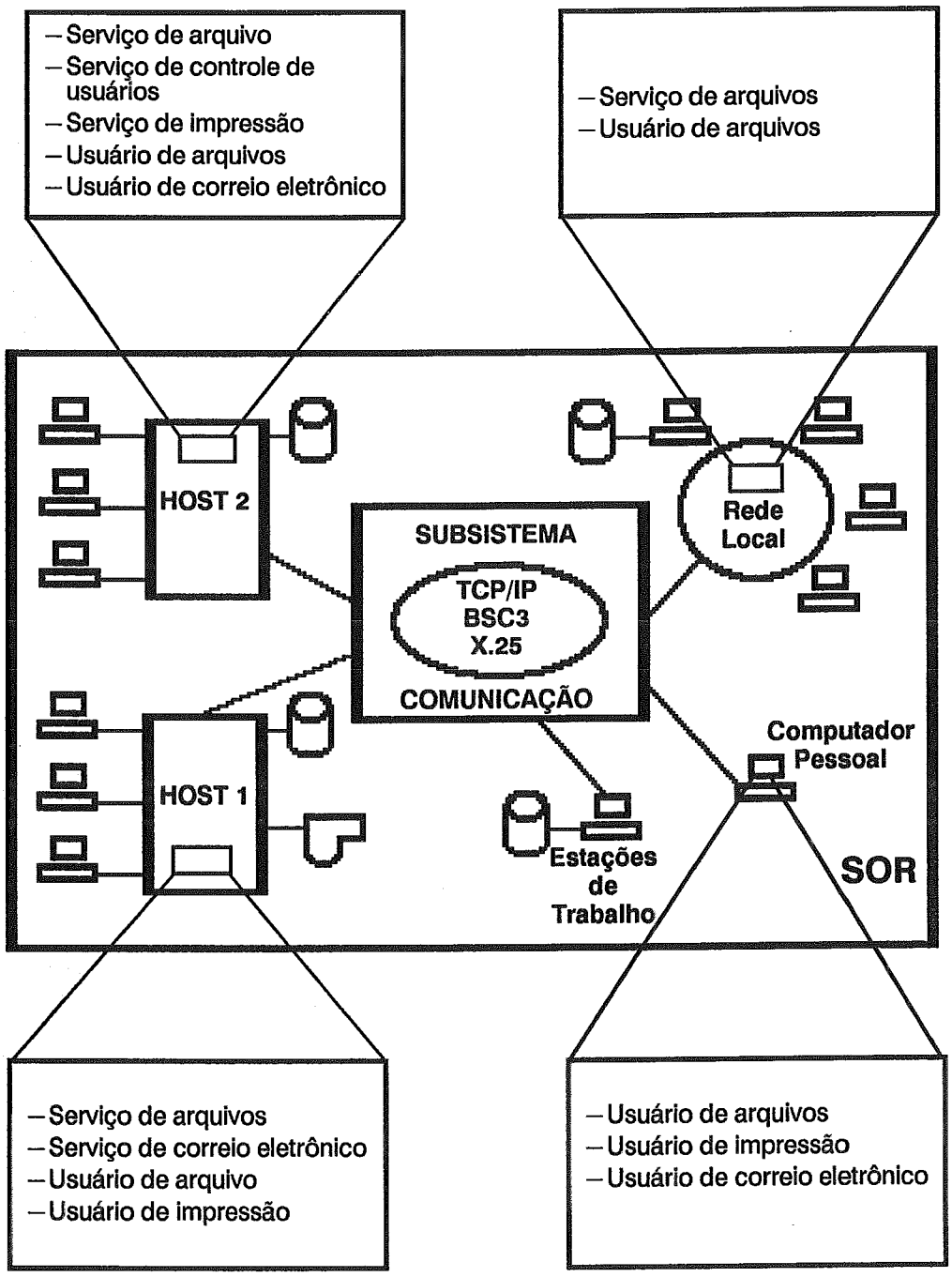


**Fig.3 Distribuição de funções no sistema**

# CAPÍTULO III

# ADMINISTRAÇÃO EM UM SOR

## 3.1) INTRODUÇÃO

Alguém já disse que os aspectos da definição do escopo da administração de um sistema computacional podem ser muito traiçoeiros [15]. Num extremo, poderíamos definí-la como a parte do sistema que não sabemos projetar. Assim, não teríamos mecanismo algum a construir, por definição. Num outro extremo a definiríamos como tudo o que tem relação com o controle e a estruturação do mesmo, possuindo assim implicações em todas as questões técnicas do sistema.

À medida que os sistemas em rede vão se tornando mais complexos, a dificuldade de sua implementação, manutenção e operação aumenta.Se não se puder dispor de mecanismos que automatizem e/ou auxiliem as tarefas mencionadas teremos sérios empecilhos para a sua expansão. Num sistema distribuído heterogêneo esta complexidade pode ser vista como um problema com três dimensões:

- A complexidade individual dos sub-sistemas componentes - complicador para a implementação dos programas do SOR.

- A quantidade de elementos componentes - complicador para a administração do SOR.

- A diversidade de tipos - complicador para a manutenção e evolução do SOR.

Como exemplos de tarefas normalmente associadas à administração de um sistema distribuído, podem ser citadas as seguintes:

- Nomeação e acesso aos recursos do Sistema
- Comunicação entre processos
- Controle de concorrência
- Atomicidade de operacões
- Duplicação de recursos
- Escalonamento e balanceamento de carga
- Gerência e manutenção dos enlaces de comunicação
- Controle de acesso e autenticação

Num sistema centralizado, uma série de problemas correlatos também existe. Fatores como disponibilidade de conhecimento global de informação e limitação intrínseca dos recursos - aqueles que a unidade centralizadora comporta - facilitam o tratamento desses problemas. Uma alternativa para um sistema distribuído seria a de centralizar-se a administração do mesmo de maneira a criarmos uma situação similar a de um sistema centralizado. Contudo as limitações do elemento centralizador face à potencialidade explosiva de crescimento, os problemas de obtenção e a atualização das informacões necessárias, e requisitos de continuidade de operação na ocorrência de falhas no sistema tornam a escolha de um elemento centralizador uma opção inadequada. São buscadas soluções que viabilizem a administrção do sistema através da cooperação entre os elementos partícipes.

Um princípio nessa hora é chave para a definição dos mecanismos de administração que deverão ser aplicados a um SOR, qual seja o de encararmos os componentes a partir da visão oferecida por seu sistema operacional nativo. Intrinsecamente o que se resolve com esse princípio é o problema da continuidade de operação mesmo quando de eventuais quedas de elementos do sistema não diretamente envolvidos. Do ponto de vista da administracão do sistema

isso implica na necessidade de se usar ou, por outro lado, não ferir conceitos próprios de cada sistema operacional nativo. A ênfase deve ser na coordenação das atividades.

Por outro lado é preciso não se abstrair de que, como exposto no capítulo anterior, é preciso gerar um ambiente com algum grau de uniformidade de tal maneira a se facilitar a construção dos programas que irão compor o SOR. Procura-se com isso reduzir o esforço para o porte dos programas desenvolvidos. A seguir analisaremos cada uma das tarefas de administração mencionadas acima no ambiente de um SOR.

## 3.2) TÓPICOS DE ADMINISTRAÇÃO

### 3.2.1) ACESSO AOS RECURSOS DO SISTEMA E NOMEAÇÃO DE OBJETOS

Basicamente existem duas formas de se efetuar processamento num sistema distribuído. A primeira forma seria a de se trazer o dado para onde ele deve ser processado. Por natureza essa forma exigiria um maior grau de coerência das estruturas do sistema, uma vez que, em se tratando de um sistema distribuído, todas as estações devem ter uma mesma visão dos objetos manipulados. A segunda forma seria a de se efetuar o processamento junto ao dado. As operações possíveis sobre um determinado elemento são aquelas definidas pelo elemento operador. Este é o típico modelo Cliente-Servidor.

Num SOR deve-se optar preferencialmente pelo segundo modelo, uma vez que facilita a interoperabilidade dos sistemas operacionais componentes. As estruturas de dados são encapsuladas e manipuladas apenas segundo as regras definidas pelas primitivas de acesso, isolando-se as particularidades de cada subsistema componente.

É de se notar, contudo, que o processamento num SOR, visto a partir do aplicativo,muitas vezes acaba por

envolver os dois tipos de processamento. No alto nível - aplicativo - os programas efetuam o processamento localmente. Ou seja manipulam diretamente o dado. Enquanto isso, num nível mais baixo - nos programas do SOR - um modelo cliente-servidor é utilizado. Esta situação decorre do objetivo de se obter um sistema operando com o efeito de transparência.

Quanto à nomeação de objetos no sistema, podem-se mencionar os seguintes problemas associados:

- Distinção entre acesso a recursos locais e remotos
- Localização de recursos

Cada estação componente de um SOR possui suas próprias regras para a nomeação dos objetos por ela gerenciados. Esse tem sido um ponto crítico na integração de sistemas heterogêneos, uma vez que as regras de nomeação são sempre diversas, às vezes conflitantes entre si. Por exemplo, o conjunto de caracteres válidos na composição dos nomes dos objetos e o próprio tamanho desses nomes podem ser diferentes para cada um dos subsistemas componentes. Uma solução para esse tipo de problema pode, por exemplo, se valer da existência de um conjunto interseção dos conjuntos de nomes dos sistemas envolvidos. Este tipo de abordagem permite um mapeamento direto dos nomes de um sistema em outro, ainda que incompleto. Uma solução mais genérica seria a de se construir um dicionário onde um nível intermediário permitiria o cadastramento dos recursos disponíveis. Os problemas associados seriam o de se manter este dicionário atualizado, e resolver a possível formação de gargalos para o cadastramento e acesso aos mecanismos de conversão.

Quanto à localização de um determinado recurso, podemos tomar dois tipos de atitude: não deixar a localização do mesmo disponível a um usuário; permitir que

um usuário especifique onde um objeto deva ser guardado. Existem alguns pontos a se considerar num SOR. Primeiramente, a diversidade de elementos que fazem parte do sistema possibilita o fornecimento de um mesmo serviço com custos, qualidade, desempenho, disponibilidade os mais variados. Logo, dar visibilidade da localização de um recurso ao usuário é de uma forma direta permitir que o mesmo escolha que tipo de serviço, dentre um leque a disposição, ele deseja utilizar. Existe, por outro lado, a possibilidade de um sistema distribuído que faça parte do SOR oferecer, dentro de seus domínios, essa invisibilidade de localização de um determinado recurso. Um outro ponto que se considera é o da existência de uma uma série de recursos do SOR para os quais uma estação componente já coloque a disposição de seus usuários recursos locais equivalentes. Com mecanismos que possibilitem seu uso de forma transparente é possível estabelecer-se uma configuração da estação onde a localização dos recursos não lhe seja mais visível. Isto, obviamente, não no sentido em que o sistema os guarda onde melhor lhe convenha, mas em que não é uma tarefa de responsabilidade do usuário ou que seja sua a escolha. Portanto, não parece objetivo primeiro de um SOR oferecer num nível mais elementar transparência na localização dos recursos, ainda que se possa oferecer algumas formas desse tipo de facilidade.

Tem sido opção para se solucionar o problema da localização de objetos no sistema o uso de um serviço denominado Serviço de Nomes. A função básica deste serviço é a de fornecer, a partir do nome de um determinado recurso, a sua localização dentro do sistema. Existem vários níveis onde esse tipo de conversão pode ser empregado. Por exemplo, um nome de um determinado recurso pode, por necessidade de se garantir unicidade num determinado ambiente, ter uma forma completamente inadequada para a manipulação por um usuário. Formas abreviadas para referência a esses recursos, ou sub-domínios reduzidos podem ser técnicas empregadas para esse fim. O

dicionário mencionado pode ser considerado um serviço desse tipo.

### 3.2.2) COMUNICAÇÃO ENTRE PROCESSOS

É fundamental, tendo em vista os trabalhos de implementação e de manutenção dos programas do SOR, que se consiga um grau de uniformidade dentro do sistema. Assim cumpre definir-se uma interface uniforme para a troca de mensagens entre os elementos componentes do sistema. É interessante que a definição dessa interface a torne o mais simples possível no que diz respeito a semântica das operações, isso com os seguintes objetivos: simplificar a escrita dos programas de sistema; uniformizar a escrita dos mesmos; minimizar o esforço de implantação dos mecanismos de comunicação para estações quaisquer. Assim , em princípio, devem ser separados, nesse nível, quaisquer aspectos referentes a escalonamento de processo, controle de acesso com vistas a mecanismos de proteção. Os mecanismos de comunicação são elementos presentes em todos os elementos componentes,e devido a essa ubiquidadde, devem ser os mais simples possíveis.

### 3.2.3) CONTROLE DE CONCORRÊNCIA E ATOMICIDADE

O acesso concorrente a objetos disponíveis num ambiente multiusuário dá origem a uma série de situações onde o sistema deve prover mecanismos que garantam a integridade dos dados. A atualização simultânea, atualização e consulta sumultâneas, a inadequação de aplicativos que operam em ambientes estendidos, são exemplos de situações que o sistema deve poder suportar. A escolha dos mecanismos de coordenação e controle possuem implicações no grau de disponibilidade e de autonomia do sistema. Por exemplo, pode-se definir que enquanto um usuário bloqueia um objeto para atualização nenhum outro tipo de operação seja permitida sobre o mesmo. Ou seja, ele

fica indisponível durante o processo. Se considerarmos a possibilidade de falhas durante a operação, podemos imaginar as complicações decorrentes.

Existem duas formas de se considerar o problema do controle de concorrência. O primeiro é no fornecimento de um serviço onde esse aspecto encontra-se resolvido já no próprio elemento hospedeiro. Por exemplo, o sistema em questão já é do tipo multiusuário e possui, por si só, suporte para o acesso concorrente. Nesse caso bastaria que o elemento servidor efetuasse um mapeamento funcional entre o que define a interface de acesso ao SOR e a interface local de acesso ao recurso - salvo complicações decorrentes da semântica das operações. O outro seria o do conjunto de extensões do SOR fornecer o nivelamento adequado. De qualquer forma, o problema de controle de concorrência é uma atividade ligada ao fornecimento de serviços específicos disponibilizados ao sistema e nesse âmbito deve ser elaborado.

A atomicidade nas operações está associada à necessidade de se garantir que determinado conjunto de operações seja indivisível. Isto quer dizer que, ou todo conjunto de operações é efetuado, ou nenhuma das operações do mesmo é efetuada. Isto possibilita a definição de estados consistentes, e que o sistema evolua de acordo com os mesmos. Com relação a atomicidade e o SOR podemos dizer que valem considerações análogas às feitas sobre o problema de controle de concorrência.

### 3.2.4) DUPLICAÇÃO DE RECURSOS

Valem as mesmas considerações do item anterior. Contudo, seria bom acrescentar que um SOR pode ser um ambiente para estudos interessante para a duplicação de recursos. Primeiramente porque a princípio a heterogeneidade não parece ser um fator impeditivo para o estabelecimento dessa característica dentro do ambiente.

Ainda assim, não parece ser uma suposição falsa a de que um grupo de elementos do sistema possa pertencer a um mesmo tipo de componente. Essa, aliás, é outra caracterísitica de um sistema heterogêneo: ele não é absolutamente heterogêneo. Tirar partido de pontos de homogeneidade para o ambiente heterogêneo parece ser um ponto a ser estudado. Uma outra forma de se obter essa característica, analogamente aos itens anteriores, seria a de um elemento participante do sistema fornecer um serviço com o grau de redundância desejado. Resumindo existem três formas de se atacar o problema da duplicação de recursos dentro de um SOR:

- De forma genérica
- Aproveitando-se de situações de homogeneidade
- Fornecimento a partir de elementos que já disponham dessa característica

### 3.2.5) ESCALONAMENTO E BALANCEAMENTO DE CARGA

A solução de uma forma genérica do problema de escalonamento e balanceamento de carga dentro de um SOR parece ser inviável. Não só diferenças de conceitos sobre o que seja um processo dentro de cada sistema estão envolvidas, mas a própria heterogeneidade do hardware do sistema inviabilizam o seu uso de forma ampla - conjunto de registradores; definição do que compõe o contexto, etc. Contudo podemos identificar as seguintes situações onde esse assunto poderia ser estudado:

- Grupos de sistemas homogêneos. Entre esses elementos formas bastante genéricas de escalonamento distribuído poderiam ser elaboradas.
- Subsistemas que tornassem disponíveis "pools" de processadores onde a sua alocação seria de responsabilidade do sistema.Vide TANENBAUM e RENESEE [1]
- A disponibilidade de programas em vários elementos poderia permitir que um sistema de balanceamento

estático de carga fosse elaborado. Não poderia, é claro, ocorrer migração de processos entre essas estações, mas o disparo de operações no sistema em locais mais convenientes, permitindo um balanceamento da carga do sistema quando da entrada de novas tarefas.

### 3.2.6) GERÊNCIA E MANUTENÇÃO DOS ENLACES DE COMUNICAÇÃO

Este é um problema físico da operação em rede. Em sistemas multiusuários esse tipo de problema, numa escala bem menor, já se fazia presente. Já havia o problema de manutenção da rede de terminais, do RJE, dos subcentros remotos, que precisavam ter acesso à unidade central. Esse tipo de ambiente possuía algumas características próprias que não mais se fazem presentes. Por exemplo, podia-se contar com um certo suporte de hardware no controle de acesso, pois as conexões físicas permitiam certas garantias com relação a fonte requesitante do acesso. A unidade central era a destinatária última do acesso, logo se ela por ventura sofresse alguma pane o resto da rede perdia o sentido. Num ambiente distribuído, deve-se levar em conta a descentralização do sistema, com relação à potencialidade de operação diante de falhas em alguns de seus elementos. Pode ser requisitado na especificação que essa facilidade seja aproveitada. Por outro lado, ocorre uma maior permissividade com relação ao acesso "físico" aos recursos de computação.

Outro ponto é que nos sistemas distribuídos atuais ocorre uma grande diversidade de tipos de enlaces de comunicação. A tendência é uma grande pluralidade de equipamentos para a construção física da rede implicando por um lado em custos mais baixos de construção e por outro em custos de manutenção potencialmente mais elevados. Esses sem dúvida são problemas de sistemas distribuídos por excelência e que vêm recebendo uma atenção crescente.

### 3.2.7) CONTROLE DE ACESSO E AUTENTICAÇÃO

Os problemas de controle de acesso e autenticação estão relacionados à capacidade de o sistema impor regras para a convivência entre os elementos componentes. A autenticação é o procedimento pelo qual o sistema procura garantir que uma determinada fonte geradora de requisições é reconhecida como associada a uma entidade lógica por ele conhecida. Ou seja, se quem requisita uma atividade é quem diz ser.

Em um sistema distribuído, como se mencionou no item anterior, a explosão dos tipos de mecanismos para a interconexão tornaram pouco confiáveis as informações daí advindas sobre à localização do elemento requisitante. O que se deduz daí é que esta é uma informação frágil quanto à identificação do elemento requisitante.

A garantia da identidade da fonte de uma requisição é chave para que o controle de acesso possa ser efetuado. O sistema deve poder decidir que operações válidas um determinado programa-usuário pode fazer sobre os objetos existentes. O sistema pode inclusive dar suporte à modulação dos direitos de operação dos programas em execução. Isto visa a implementação de políticas de operação em nível de menor conjunto de privilégios necessários.

Formas de controle de acesso e autenticação podem já estar implementadas nos diversos subsistemas componentes. Na análise desse problema alguns princípios são chaves.Primeiro a estação é aquela vista pelo seu sistema operacional. Logo devem ser providos mecanismos que viabilizem o trabalho local na realização dessa tarefa. Deve-se ter em especial atenção à não degradação da qualidade de serviço com que a estação executa suas funções. Em segundo lugar devem ser fornecidos mecanismos uniformes para que os serviços tenham reduzidos os custos

associados à construção, à manutenção e à evolução do sistema.

Este tópico é de particular interesse neste trabalho. Primeiro porque até agora, com o enfoque dado, nenhum trabalho foi por nós encontrado. Em virtude disso e da importância do assunto vamos nos ater daqui até o final deste capítulo especificamente ao estudo desse problema no ambiente de um SOR.

## 3.3) ASPECTOS DO CONTROLE DE ACESSO NUM SOR

### 3.3.1) O CONCEITO DE USUÁRIO

À medida que os sistemas foram sendo interconectados para que os dados pudessem ao menos ser acessados, passamos a ter um ambiente de computação sem sistema operacional. Essa ausência de um sistema operacional é refletida principalmente pela falta de regras para a coordenação do acesso aos recursos do sistema. Uma série de tarefas tem que ser realizadas de forma não automática. Dados para serem tratados pelos programas têm que ser transferidos e/ou reformatados para poderem ser processados pelos subsistemas componentes. Com isso múltiplas cópias dos objetos precisam ser armazenadas cabendo a regras extra-ambiente garantir a sua coerência. Outro ponto onde esse estado de coisas se reflete é na interação do usuário com o sistema. Um usuário que queira fazer um acesso a recursos que estejam dispersos no ambiente precisa saber uma série identificações e de senhas, quando não saber também como interagir com uma diversidade de equipamentos.

Quando se pensa em adotar um SOR como solução para a integração de um ambiente heterogêneo, alguns autores acreditam que o problema de um usuário ter que efetuar múltiplas conexões é um problema inerente. Na verdade, juntamente com o sistema operacional, desapareceu um

conceito que é comum a qualquer sistema de computação: o usuário. Num sistema operacional convencional, o usuário é aquela entidade à qual associamos a responsabilidade pelas tarefas desempenhadas por um elemento ativo dentro do sistema. Essa responsabilidade possui atributos para a contabilização dos recursos utilizados do sistema. Ela possui atributos que a individualizam com relação a outras entidades de tal forma a permitir que se discrimine o acesso a objetos guardados pelo sistema. Isso é feito a fim de se modelar dentro de um sistema de computação situações similares àquelas encontradas no dia a dia das organizações. Normalmente a entidade usuário é associada a um indivíduo, ou grupo de indivíduos que fazem uso do sistema de computação. Dado que um sistema possui recursos e uma comunidade-imagem de pessoas, instituições, etc, e que o sistema é o guardião de valores - informações - para essa comunidade, é preciso que o guardião possa modelar os relacionamentos existentes dentro dessa comunidade. No entanto, seja qual for o esquema a se elaborar para o SOR, ele deve garantir a cada sistema individualmente a capacidade de gerir seus próprios objetos.

Quando se interconectam diversos sub-sistemas de computação num SOR, é preciso que os usuários de cada um dos referidos sub-sistemas possam ter acesso controlado aos recursos da comunidade global. A forma mais natural como isso pode ser feito é através de mecanismos que viabilizem a transparência. Num certo aspecto é uma situação análoga ao trânsito de pessoas entre dois países. As autoridades competentes de um país têm que fornecer o passaporte e as correspondentes do outro o visto de entrada. Assim o componente do SOR de uma estação deve cuidar para que um usuário consiga a entrada em outro sistema baseado em elementos administrados por ele. A analogia perde um pouco o sentido quanto ao tratamento que o sistema destino do acesso dá ao elemento externo. O sistema alvo do acesso apenas reconhece elementos definidos em seus domínios. Ele apenas age em favor de seus usuários, guardando objetos

para elementos estáveis - os seus usuários. Seria como, ao recebermos o visto de entrada, nos tornássemos cidadãos do país visitado. De alguma forma deve ser possível a um sistema operacional mapear o seu conjunto de usuários em usuários de um sistema remoto. Logo o mecanismo que se pretende para criar o conceito de usuário de um Sistema Operacional de Rede não é aquele de se criar uma central de acesso global para controle, mas o de se criarem mecanismos para permitir a mediação do trânsito de objetos e demanda de processamento entre as estações componentes.

Neste momento é interessante considerarmos um ambiente de computação, que foi fator decisivo para a explosão do uso de sistemas informatizados: o aparecimento do computador pessoal e sua evolução para o conceito de estação de trabalho. O uso disseminado do computador foi possível graças ao baixo custo dessas estações. Como características, elas possuem um Sistema Operacional nativo simples, cuja função principal é a de criar uma interface de acesso mais adequada a seus recursos que o uso direto dos mesmos. Como essas estações de trabalho são de uso individual, o conceito de usuário ficou implícito - existe apenas um usuário: aquele que tem acesso a ela. Logo, a integração desses sistemas em rede, integração essa que é capital, deve tornar explícito outra vez o conceito de usuário. Destacamos esse ponto pela necessidade de impormos disciplinas diferentes para esses ambientes de trabalho. Um outro ponto curioso é que muitos desses sistemas vêm se agregar ao ambiente, perdendo os recursos clássicos normalmente associados - discos e impressoras- retendo apenas sua capacidade local de processamento. O seu Sistema Operacional ganha com isso contornos mais nítidos de normalizador do acesso. Outros pontos deveriam ser considerados quanto à definição de requisitos que o SOR deva atender para a integração. Por exemplo, o ponto principal responsável pela explosão do uso da informática foi a capacidade de trazer a ferramenta de solução dos problemas mais para perto do beneficiado. Ou seja,

aproxima-se a programação do usuário - capacidade do usuário resolver seus próprios problemas - em oposição aos gargalos criados por CPDs com procedimentos centralizadores.

### 3.3.2) USUÁRIOS E OS SERVIÇOS

Muitos dos serviços que se vai implementar no sistema vão efetuar seus procedimentos através do acesso a objetos que os subsistemas componentes colocam à disposição de seus usuários. Por outro lado, é preciso ter em atenção que muitos desses subsistemas não estão preparados para tratar adequadamente situações novas a que sua integração a um sistema global, fisicamente mais permissivo, possa dar origem. Assim como os serviços que se vai disponibilizar no sistema têm que lidar com esse problema, seria interessante que um tratamento uniforme pudesse ser dado.

Seria interessante que pudéssemos ter um serviço do SOR que, através de um conjunto uniforme de primitivas, realizasse as tarefas relacionadas à validação dos elementos para controle de acesso e tradução desses mesmos elementos para o ambiente onde a tarefa fosse ser executada. Abaixo identificamos algumas funções desse tipo de serviço:

- Prover mecanismos que permitam ao implementador do serviço discriminar que tarefas ele pode ou não desempenhar em favor de um determinado requerente.

- Permitir ao executor personalizar um usuário dentro do sistema hospedeiro. Assim, durante a execução da tarefa requisitada garantiríamos que o sistema operacional nativo não violaria alguma de suas regras.

A última função é uma alternativa àquela opção em que um elemento executor onipotente dentro do hospedeiro realizasse as tarefas. Tal opção possui dois perigos: O

primeiro seria o de inviabilizarmos uma implementação uniforme dos serviços. Cada sistema pode possuir seus próprios conceitos sobre o que e como deve ser protegido. Para reproduzirmos os controles que o sistema nativo efetuasse sobre seus usuários teríamos que individualizar a sua implementação comprometendo sua portabilidade. O outro perigo seria o de fragilizarmos o sistema hospedeiro. Isto em decorrência de uma implementação incompleta de seus conceitos, ou por falhas que potencialmente podem estar presentes no software do SOR. Uma maneira consistente de se garantir a integridade do Sistema Operacional hospedeiro, e por conseguinte dos objetos sob sua guarda, é a de se executar os procedimentos para um usuário encarnando um elemento de direitos convenientemente restringidos dentro do mesmo. Esse tipo de abordagem segue o princípio da limitação de privilégios ao nível suficiente para a execução de uma tarefa conforme SALTZER e SCHROEDER [16].

Para a implementação de um sistema com as características citadas pensou-se em mecanismos baseados em "capabilities". Sobre o uso desses mecanismos em rede é bom ver MULLENDER e TANEBAUM [3] e NEEDHAM [17]. Escolheu-se esse caminho pelo desacoplamento inerente que existe entre o elemento que executa uma tarefa e o objeto que contém a informação sobre os direitos de operação do requerente e o próprio requerente. Outro desacoplamento que se procurou obter foi entre o mecanismo que valida uma chave - o elemento que carrega informações sobre a identificação - e o tipo de serviço a que ele está associado. Com isso permitiu-se a construção de um suporte genérico aos demais serviços do sistema, contribuindo para a uniformização da construção dos serviços no ambiente. O próprio serviço que mantém o conjunto de chaves pode ser mantido em boa medida através desses mecanismos.

## 3.4) PROPOSTA DE UM MECANISMO PARA CONTROLE DE ACESSO EM UM SOR

Para se viabilizar uma implementação uniforme de mecanismos para controle de acesso no SOR pensou-se no uso de "capabilities" - chaves. Dois pontos são importantes nesse mecanismo: o primeiro é o conteúdo de informações que a chave proporciona; o segundo ponto é a capacidade de garantirmos que essa chave é um elemento fidedigno. Devemos observar que as chaves não estão intrinsecamente associadas a objetos, num sentido amplo, dentro do sistema - cada subsistema possui mecanismos próprios de controle de acesso. Elas são usadas como um mecanismo global para modular a operação de cada serviço.

Para se garantir que uma chave é fidedigna pensamos na construção de um serviço que efetuasse, de maneira uniforme, o procedimento de validação. Optamos por esse tipo de abordagem, ao invés de, por exemplo, construir-se a chave e criptografar suas informações, pelas seguintes razões:

- simplicidade
- informações associadas à chave que não seria conveniente constar da própria chave.
- facilidade de manipulação da chave pelo subsistema usuário.

Vamos agora mostrar o tipo de informações que estariam a ela associadas.

### 3.4.1) DESCRIÇÃO DE UMA CHAVE

Apesar do uso do termo campo para designar um destaque de informacão, não se quer dizer com isso que a chave carregue consigo essa informação. Ela pode estar, na

verdade, indiretamente associada através de tabelas mantidas pelo serviço de autenticação da chave.

### 3.4.1.1) CAMPO DE IDENTIFICACÃO DE SERVIÇO

Com a finalidade de diminuir o número de interfaces, e conseqüentemente a interdependência entre um serviço e o sistema que lhe dá suporte é conveniente que se possa deduzir, através da chave o serviço a ela associado. O objetivo desse campo é o de fornecer um meio para que o próprio serviço verifique se uma determinada chave correta dá realmente direitos de acesso sobre algum objeto por ele gerenciado. Este campo não tem, em princípio, relacão alguma com mecanismos para o roteamento de uma requisicão dentro de uma estação. Surge então um problema administrativo, qual seja o de gerenciar o espaço de identificações entre os diversos possíveis serviços.

O problema na verdade pode ser simplificado, com a definição de domínios administrativamente independentes. Procuramos com isso manter o princípio de dar autonomia local para o maior número de decisões possíveis. Essa possibilidade decorre do fato de que este campo, para os objetivos de controle dos direitos de acesso, só precisa ser interpretado localmente. Apenas o executor do serviço e o serviçn de controle de acesso precisam concordar quanto à sua interpretação.

Quanto ao uso, cabe a cada sistema escolher as chaves convenientes, para o acesso. Como cada serviço é fornecido indepentemente uns dos outros, cada gerente de uso de uma estação, para um determinado serviço, deve ter sua base de chaves particular e manipulá-la apropriadamente.

### 3.4.1.2) CAMPO DE IDENTIFICAÇÃO DE DIREITOS

Este campo permite que se faça uma distinção mais específica sobre que operações podem ser efetudas a partir de uma determinada requisição. Existem duas operações básicas sobre este campo: a primeira é a validação de seu conteúdo. Esta é uma tarefa realizada por uma primitiva do sistema proposto e que dá segurança ao executor do serviço. Para se permitir uma gama variada de políticas e tipos de controle de direitos define-se um esquema, ainda assim simples, onde: bit ligado => direito permitido. É permitido que uma cópia de uma chave numa requisição tenha direitos no máximo iguais ou mais restritivos que a forma original.

A segunda operação diz respeito à interpretação de cada um dos bits do campo. Esta parte define que operações específicas de um serviço em particular são permitidas. Esta é uma tarefa de cada executor e que, uma vez efetuada a validação, pode ser realizada sem maiores problemas.

Deve-se notar que para a execução de uma determinada operação, pode-se definir uma ampla gama de combinações de formas de uso, inclusive com a exigência de mais de uma chave .isto parece permitir a implantação de políticas bastante variadas.

### 3.4.1.3) CAMPO DE GARANTIA CONTRA FALSIFICACÕES

Este campo permite uma maior liberalidade na manipulação das chaves.Este campo possui um número gerado randomicamente que impede que se forge uma chave. Escolhendo-se um tamanho conveniente para esse campo pode-se trabalhar com uma probabilidade de falsificação tão pequena quanto se queira. Menor, por exemplo, que a probabilidade de uma falha de hardware num esquema mais convencional de "capabilities".

Para se evitar que a partir de um computador conectado ao sistema, termine-se por descobrir uma chave que permita acesso a um serviço por tentativas, pode-se monitorar requisições de caráter suspeito. Medidas de resolução automáticas ou com a intervenção de um operador poderiam ser tomadas.

3.4.1.4) CAMPO DE IDENTIFICAÇÃO DE USUÁRIO NO SISTEMA HOSPEDEIRO

A finalidade desse campo é a de suprir informações para que o executor de um determinado serviço que age sobre objetos guardados pelo sistema operacional nativo possa restringir o seu acesso. Como foi dito, isso permite a execução do serviço sem se degradar o grau da qualidade do serviço - com relação à segurança - fornecido pela estação hospedeira.

É bom notar que existem serviços que não operam sobre objetos previamente existentes para o SO nativo da estação hospedeira. O próprio serviço de controle dessas chaves para o controle de acesso é um exemplo desse tipo de serviço. Para esses serviços e outros em que considerações diferentes - complexidade, simplicidade, etc. - indicarem uma solução que não implique na personificação de usuários locais ao sistema hospedeiro esse campo ficaria sem sentido.

Esse campo particularmente não deve estar presente na chave. O que indica esse tipo de procedimento é a multiplicidade de estruturas e atributos para caracterizar um usuário dentro de cada sistema.

### 3.4.1.5) CAMPO DE IDENTIFICAÇÃO DE ESTAÇÃO

Este campo contém informações necessárias para que se possa identificar para qual estação a chave foi inicialmente gerada. Para que se possa controlar que tipo de operações administrativas uma estação pode efetuar sobre objetos de uma outra estação, é necessário poder-se associar as chaves a seus donos. É possível, por exemplo, construirem-se mecanismos que permitam realizar operações administrativas a partir de qualquer elemento do sistema de forma controlada. Pretendemos com isso possibilitar a implementação de políticas que limitem o acesso a partir da fonte da requisição. Num sistema único interligado a um computador de grande porte a sua própria rede de terminais garantia a estrutura física de interconexões. Num sistema interligado por uma rede podemos definir graus de confiança quanto à capacidade do subsistema de transporte garantir a fonte de determinada requisição. Outra alternativa, seria a desse campo não representar um endereço, que pudesse ser forjado, mas a uma identificação que algum protocolo apropriado pudesse verificar.

### 3.4.2) DESCRIÇÃO DOS SERVIÇOS PARA SUPORTE AO CONTROLE DE ACESSO

O serviço que se quer especificar possui dois grupos básicos de funções: O primeiro compreende as primitivas de suporte para a validação das chaves para acesso aos diversos serviços e as funções que auxiliam na manutenção da base de dados formada pelo conjunto de chaves. O segundo grupo compreenderia àquelas funções que permitiriam o mapeamento do usuário de um subsistema componente no conjunto de chaves a ele disponível. Será objeto de reflexão apenas o primeiro conjunto.

Vale observar que se busca definir o próprio serviço de manutenção do conjunto de chaves usando os

próprios elementos do serviço. Contudo, existem situações onde outros mecanismos têm que ser empregados. Essas situações ocorrem basicamente durante a fase de "start" do serviço. Admitem-se essas formas apenas durante essa fase , seguindo-se a partir daí a operação normal.

### 3.4.2.1) CADASTRAMENTO DE SERVIÇO

Como vimos anteriormente, existe a necessidade de associarmos a cada serviço um determinado identificador. Temos, portanto que administrar um domínio constituído por esses identificadores. Como a sua interpretação tem significado apenas em âmbito local, é natural que o acesso a essa função esteja restrito aos elementos servidores locais. Definimos esse nível de indireção para dar independência entre os serviços que vierem a ser implementados e o suporte dado pelo sistema aos mecanismos de controle de acesso.

Um elemento executor de um serviço deve, no momento de sua instalação, requerer o seu cadastramento junto ao sistema. Um identificador é alocado para o executor, de forma que futuramente, quando do acesso, ele possa saber as chaves que se referem a objetos por ele gerenciados.

Além dessa necessidade para controle de direitos, é necessário prover alguma forma de identificação que faça sentido em termos globais no sistema. Essa necessidade surge do fato de que, para que alguém possa fazer uso de um serviço, é necessário que possa referenciá-lo. Em outras palavras, é necessário que o serviço possua um nome de forma a ser por todos reconhecido.

3.4.2.1.1) FUNÇÃO PARA CADASTRAMENTO DE SERVIÇO

CADASTRA_SERVIÇO

parâmetros de entrada

NOME_DO_SERVIÇO - Nome do serviço a cadastrar

parâmetros de saída

IDENTIFICADOR

3.4.2.2) CRIAÇÃO DE CHAVES

A criação de novas chaves no sistema deve tomar em consideracão uma série de fatores. Um primeiro fator diz respeito à distinção de dois tipos de executores de serviços. O primeiro operaria sobre tipos de objetos que o próprio sistema operacional da estação provedora já gerencia. Assim sendo é importante que se possa mapear o requisitante do acesso remoto num usuário reconhecido pelo sistema nativo. Um outro grupo de serviços, dos quais este serviço seria um exemplo, operam sobre objetos que estão fora do escopo do SO nativo da estação. A não introdução de uma operação adicional, o mapeamento do usuário remoto num local, possibilita uma simplificação interessante, embora não necessária.

Outro fator a se considerar é o problema de competência administrativa para gerir o cadastramento de usuários e chaves no sistema. Com relação ao sistema que requisita a criação de uma chave três tipos de procedimentos podem ser realizados: O primeiro permitiria que um usuário no sistema nativo hospedeiro fosse criado quando da requisição de criação da chave. Um problema que existiria nesse tipo de procedimento seria o de dar permissão para esse tipo de operação a uma outra estação

remota. O outro seria o problema relativo à diversidade de conceitos e descrições do que seria um usuário dentro de cada sistema. Talvez a definição de perfis default pudesse constituir uma alternativa. De qualquer forma a criação de usuários remotamente poderia ser interessante para a operação de um subconjunto de estações heterogêneas e/ou situações onde os ganhos com tais procedimentos fossem relevantes.

Onde não se pudesse ou não se quisesse criar um usuário no sistema operacional nativo a operação deveria ser divida em fases. Primeiro um conjunto de usuários seria criado localmente na estação provedora dos serviços. Esses usuários seriam disponibilizados num conjunto associado a uma determinada chave. Esta chave deveria ser posteriormente apresentada quando do pedido de criação de uma nova chave para o acesso. Uma forma de descrição dos usuários disponíveis seria necessária de tal forma que o mapeamento pudesse ser convenientemente efetuado. A disponibilização poderia obedecer a uma série de políticas,por exemplo, o serviço poderia garantir ou não que usuários estivessem associados a apenas um conjunto.

Outro ponto é a definição de que direitos um determinado susbsistema poderia definir para que constassem de suas chaves. Seria como uma matriz que definiria um espectro máximo para as chaves derivadas. Nos serviços onde um mapeamento a um usuário local não fosse estabelecido só esse controle estaria presente. Deve-se observar que este seria um ponto de contato, no que tange à conformidade de interpretação, entre o serviço de controle de acesso e o executor do serviço.

Um último fator que se considera agora diz respeito à criação e à distribuição das chaves de controle do próprio serviço de manutenção do conjunto de chaves. Uma vez que não existe nenhuma chave para acesso a este serviço inicialmente, é necessário prover mecanismos outros para a

criação dessas mesmas chaves e seu posterior acesso por estações remotas. A essas chaves estariam associados basicamente os seguintes direitos:

- Direito de criação de um usuário
- Direitos de eliminação de chaves
- Direitos para alteração de direitos associados

Como cada chave está associada a um determinado conjunto de usuários ou a uma matriz de usuário o controle para se listar o conjunto e o controle sobre a associação a um usuário que foi disponibilizado são feitos automaticamente.

3.4.2.2.1) FUNÇÃO PARA CRIAÇÃO DE UMA CHAVE GERADORA

CRIA_CHAVE_GERADORA

parâmetros de entrada

NOME_DO_SERVIÇO - Nome do serviço para se cadastrar o usuário.

REQUISITANTE - Identificação de um futuro requisitante.

DIREITOS_CHAVES - Operações possíveis através do serviço de controle de direitos.

MATRIZ_USUARIO_LOCAL - Se direito permitir operação

DIREITOS_ESPECÍFICOS - Operacões possíveis através do serviço associado a esta chave.

parâmetros de saída

CHAVE

3.4.2.2.2) FUNÇÃO PARA A DISPONIBILIZAÇÃO DE UM USUÁRIO

DISPONIBILIZA_USUARIO

parâmetros de entrada

CHAVE_GERADORA - Chave criada a partir da operação anterior.

CONJUNTO_DE_USUÁRIOS - Conjunto de usuários a serem acrescidos a um possível conjunto existente.

parâmetros de saída

STATUS_DA_OPERAÇÃO

3.4.2.2.3) FUNÇÃO PARA CRIAÇÃO DE CHAVE DE ACESSO A SERVIÇO

CRIA_CHAVE_SERVIÇO

parâmetros de entrada

CHAVE_GERADORA - Chave gerada por função anterior com direitos eventualmente restringidos.

USUÁRIO_SO_NATIVO - Se para criar ou identificação de um dos usuários disponíveis.

parâmetros de saída

CHAVE_SERVIÇO

3.4.2.3) VALIDAÇÃO DE UMA CHAVE

A validação de chaves é uma tarefa desempenhada pelo SOR em cada estação prestadora de serviços de forma que os mesmos possam ter a certeza de operar a partir de elementos confiáveis - as chaves. A interpretação particular do campo de direitos é de responsabilidade de cada executor de serviço, sujeito às regras mencionadas.

A execução dessa função define o momento para o fornecimento, ao serviço que requisita a verificação, de informações adicionais associadas que não estejam diretamente presentes na chave. Por exemplo, o código de identificacão do serviço; informações que permitam identificar o usuário dentro do sistema operacional nativo; etc.

A forma de uso dessa função está ligada a aspectos de implementação de cada serviço em particular. Por exemplo, pode-se executar a validação uma única vez quando do estabelecimento de uma sessão entre o usuário e o servidor. Outras situações implicariam em uma ou mais operações de validação para cada transação usuário-servidor.

3.4.2.3.1) FUNÇÃO PARA A VALIDAÇÃO DE UMA CHAVE

VALIDA_CHAVE

parâmetros de entrada

CHAVE - a chave que se quer validar.

parâmetros de saída

CÓODIGO_DO_SERVIÇO - Identificador do serviço.

IDENTIFICAÇÃO_USUÁRIO - Identificação do usuário local.

ENDEREÇO_ESTAÇÃO - Endereço da estacão possuidora da chave.

3.4.2.4) OBTENÇÃO DE CHAVES

Como podemos ter uma série de tipos de relacionamentos entre as estações, no que tange à capacidade administrativa, devemos prover funções que permitam a obtenção de informações estabelecidas por outros centros, talvez mais privilegiados. Esta função permitiria que uma estação tivesse acesso a um conjunto de chaves que

foi estabelecido por uma outra. Algo como direito de apenas leitura sobre um determinado conjunto de chaves.

O sistema operacional local poderia gerir o conjunto de chaves que lhe foi disponibilizado liberalmente entre o seu conjunto de usuários. Contudo não lhe seria facultado expandir o conjunto de elementos de acesso em uma estação hospedeira.

Poderíamos estabelecer medidas que disciplinassem o uso desse tipo de função. Por exemplo, seria admitido o uso dessa função apenas uma vez para cada chave.

Com relação à obtenção da chave de controle de acesso inicial pode-se pensar num mecanismo de autenticação de começo que permita que um conjunto de chaves gerenciadoras primitivas pudessem ser obtidas. Para tanto um protocolo de autenticação entre as partes envolvidas teria que ser utilizado para se garantir a identidade das partes.

3.4.2.4.1) FUNÇÃO PARA OBTENÇÃO DE CONJUNTO DE CHAVES PARA ADMINISTRAÇÃO

OBTÉM_CHAVES_ADMINISTRATIVAS

parâmetros de entrada

REQUISITANTE - Informação oriunda do protocolo de autenticação.

parâmetros de saída

CONJUNTO_DE_CHAVES_ADMINISTRATIVAS

3.4.2.4.2) FUNÇÃO P/ OBTENÇÃO DE CONJUNTOS DE CHAVES

OBTEM_CHAVES

parâmetros de entrada

CHAVE_ACESSO - chave que dá dureito de acesso sobre um conjunto de chaves.

parâmetros de saída

CONJUNTO_CHAVE - conjunto de chaves p/ acesso

3.4.2.5) ELIMINAÇÃO DE CHAVES

Podemos identificar duas situações distintas para o uso desse tipo de operação: a primeira diz respeito a capacitação de uma estação para gerir um conjunto de chaves a ela associado. Outra situação seria a capacitação de determinadas estações como centros de operações administrativas do sistema. A partir desses centros seria possível gerir subconjuntos de chaves definidas para outras ou todas as demais estações do sistema. Como as chaves são por natureza unidades autônomas, com relação a definição de competência administrativa, a simples posse das mesmas controlaria a operação. Poderíamos impor algumas políticas que restringissem o espectro de ação dessas chaves, como,por exemplo, apenas proceder a execução da operação a partir de determinados centros.

Com relacão a requisição da operação é preciso ter em atenção que uma chave está associada a uma série de informações, sendo interessante que cada informação possa ser utilizada como chave (chave de acesso) para decidir a eliminação, ou mesmo uma combinação delas. Por exemplo, eliminarem-se chaves associadas a um determinado serviço, Ou então, as chaves associadas a um determinado usuário.

3.4.2.5.1) FUNÇÃO P/ A ELIMINAÇÃO DE CHAVES

ELIMINA_CHAVE

parâmetros de entrada

CHAVE_PERMISSÃO - chave que autoriza a operação

INDICADOR_CONJUNTO_CHAVE - Se se deseja eliminar uma chave de um conjunto especificamente.

INDICADOR_CHAVE - especifica uma chave dentro do conjunto.

SERVIÇO - Se se deseja eliminar um conjunto de chaves por essa característica.

IDENTIFICAÇÃO_USUÁRIO - Se se deseja eliminar um conjunto de chaves por essa característica.

parâmetros de saída

STATUS_DA_OPERAÇÃO

3.4.2.6) ALTERAÇÃO DE DIREITOS DE ACESSO ASSOCIADOS A UMA CHAVE

As observações e definições acima citadas valem para essa função, apenas se modificando o objetivo da operação.

3.4.2.6.1) FUNÇÃO PARA ALTERAÇÃO DE DIREITOS DE ACESSO

ALTERA_DIR

parâmetros de entrada

parâmetros de saída

3.5) VISALIZAÇÃO DO SISTEMA

A figura 4 mostra um dos procedimentos possíveis para permitir que um usuário de um sistema possa acessar recursos em outro. No exemplo, seria necessário que o

usuário U2x de Host 2 fosse cadastrado junto ao mediador local de acesso. O mediador consultaria a base de dados local (CHAVES) e emitiria uma requisição ao serviço de cadastramento em Host 1. Lá, o serviço de cadastramento efetuaria a validação da chave geradora e emitiria uma chave de acesso como resposta. Observe-se que o serviço de cadastramento acessa a mesma base de dados que o serviço de validação.

Na figura 5, temos um procedimento alternativo para permitir que o mesmo usuário possa acessar recursos em Host 1. Neste caso, seja por agrupamento, seja por pré-alocação, apenas o mediador local está envolvido no processo. Atendendo a uma requisição local, o mediador, por si só, faz a associação entre o usuário e a chave de acesso.

Na figura 6 exemplificamos um acesso a um serviço de arquivos remoto. O par identificador (U2x,H1) identifica univocamente uma chave de acesso. Uma requisição é montada com os parâmetros e passada em seguida para o serviço adequado em Host 1. Lá o serviço de arquivos efetua a validação através do serviço de suporte, o qual retorna a identificação do usuário local e demais parâmetros, para que o serviço de arquivos tenha seguimento.

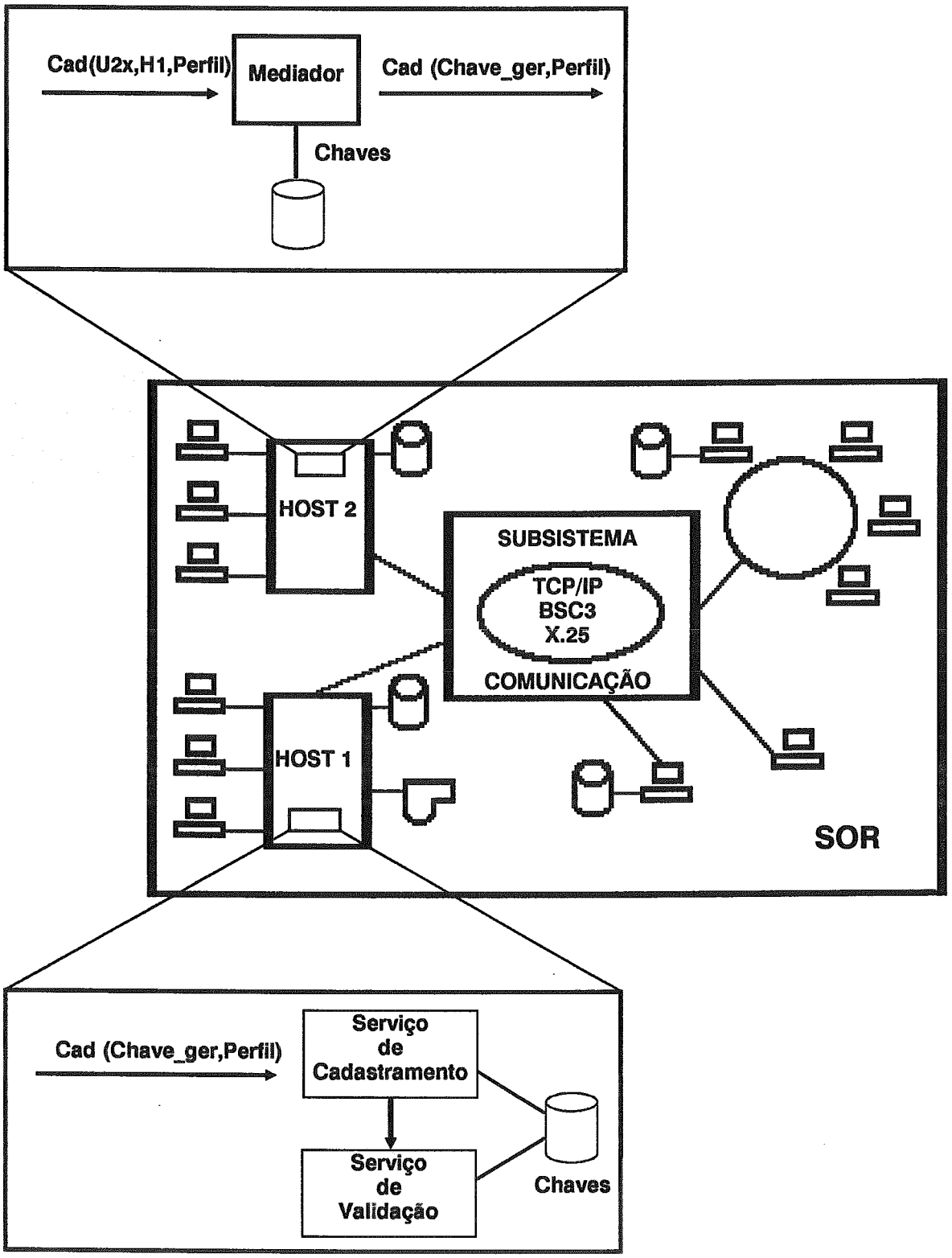


**Fig.4 Cadastramento de um usuário para acesso remoto com geração de uma nova chave**

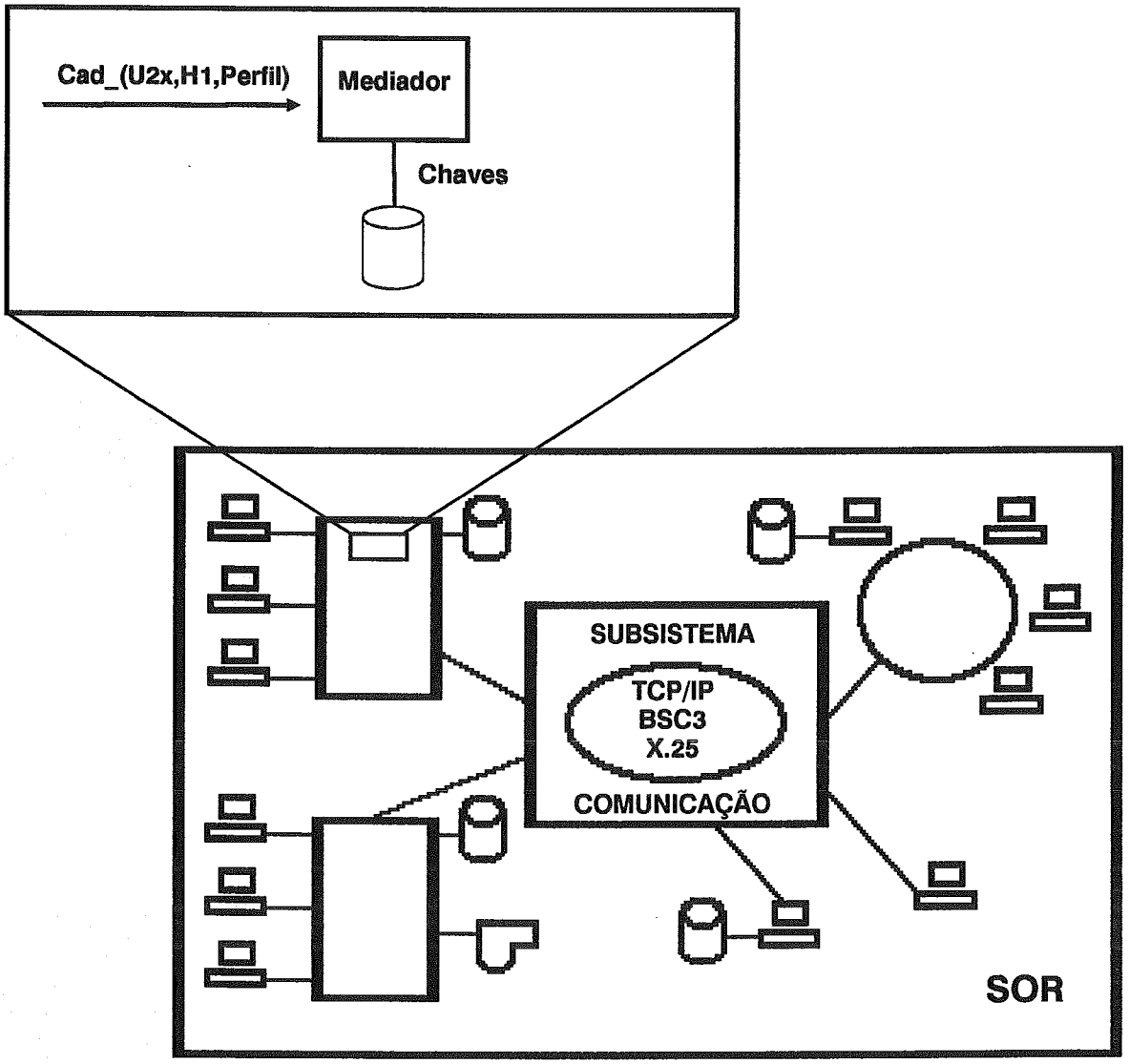


**Fig.5 Cadastramento de um usuário para acesso remoto sem geração de uma nova chave**

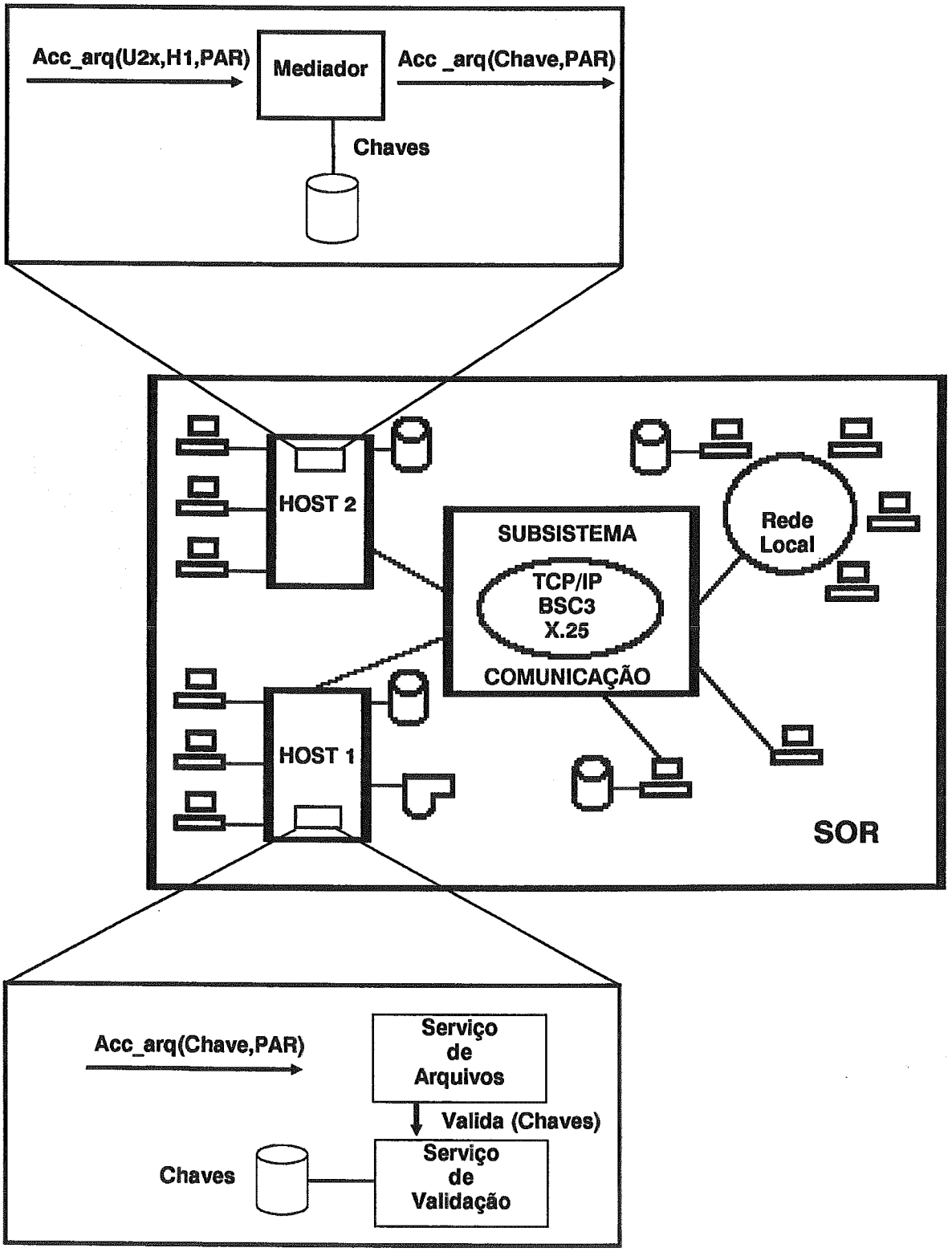


**Fig.6 Acesso de um serviço remoto**

# CAPÍTULO IV

# A CONSTRUÇÃO DOS SERVIÇOS NUM SOR

## 4.1) MODELO DE RELACIONAMENTO

A construção de serviços em um ambiente distribuído é um assunto estudado há algum tempo. A forma mais comum de se implementar um ambiente de processamento tem seguido o modelo Cliente-Servidor. Neste modelo Clientes são os elementos que desejam utilizar determinados recursos. O acesso a esses recursos é por sua vez regulado por outros elementos - denominados Servidores - que executam os procedimentos necessários em favor dos clientes.

Em decorrência desse modelo existe uma separação física entre o agente - o Cliente - e o objeto. Assim, ficam estabelecidos pontos, ou melhor interfaces, que definem uma distribuição de tarefas entre as partes envolvidas. Interfaces análogas, na verdade, já existem em qualquer sistema computacional. Temos, por exemplo, a interface de acesso a seu Sistema Operacional. Contudo, essa divisão num ambiente distribuído, tem a característica adicional do envolvimento de unidades físicas distintas, ou seja, potencialmente dissimilares quanto a capacidade de realização do processamento.

Os Servidores possuem geralmente as seguintes atribuições principais:

- definir as regras de acesso a seus serviços.
- implementar as políticas de controle de direitos de acesso sobre os objetos sob sua guarda.
- gerenciar o acesso concorrente dos possíveis clientes a esses objetos.

Dentro desse esquema é, em princípio, papel do cliente se acomodar às regras existentes para acessar os serviços disponíveis. Para a implementação do conceito de transparência, existe uma estratificação do software básico de suporte, de tal forma, que o usuário real dos serviços não tem consciência dos mecanismos envolvidos.

## 4.2) CHAMADA REMOTA DE PROCEDIMENTO: RPC

### 4.2.1) DEFINIÇÃO

Para que o processamento distribuído seja viável é necessário que as informações possam fluir entre os elementos participantes. A forma mais direta que se tinha, quando do início dessa forma de processamento, era a de um elemento enviar, explicitamente, uma mensagem para outro. Outro fator que tem contribuído para esse tipo de abordagem é a ênfase na uniformização dos mecanismos e protocolos de comunicação. Ocorre que, ainda que se possa interligar as estações envolvidas, a programação realizada diretamente sobre os mecanismos de comunicação geralmente está acima da capacidade da maioria dos programadores. É também problemática a definição de primitivas de alto nível para a troca de mensagens, uma vez que ela implica em restrições quanto a sua forma de uso. Por exemplo, se as primitivas são bloqueadas ou não; o tratamento dado aos buffers de comunicação disponíveis, etc.

Dado que, para haver processamento distribuído é necessário a troca de informações, e o envio de mensagens diretamente de um elemento para outro pode ser complexo, surge a idéia de se tratar a comunicação entre dois elementos como uma chamada de rotina qualquer. Para o modelo Cliente-Servidor, essa forma de comunicação se adapta perfeitamente. A própria mecânica dos serviços principais disponibilizados num ambiente de um SOR se adapta, quanto a forma, nesse esquema. Por exemplo, o acesso a um sistema de arquivos é feito numa estação através de primitivas

disponíveis nos sistemas operacionais existentes. Ora, se ao invés de executarmos o serviço localmente, o fizermos em outro lugar e devolvermos os parâmetros desejados, teremos o efeito de transparência realizado de uma forma bastante direta. O conceito de Chamada Remota de Procedimento se aplica não só para o efeito de transparência, mas também para viabilizar a construção de aplicativos de forma distribuída. Estruturar um programa em sub-rotinas é um conceito já estável e difundido como técnica de programação. Esse mecanismo procura manter, tanto quanto possível, a mesma sintaxe de chamadas de rotinas normais. A tarefa de construção de aplicações distribuídas ganha assim um certo ar familiar para quem a realiza. Ou seja, faz parte da cultura de uso dos programadores.

### 4.2.2) ELEMENTOS DE UMA RPC

O objetivo desse mecanismo é viabilizar a comunicação entre programas, permitindo que o programador se abstraia de aspectos de codificação dos dados, detalhes de protocolos de transporte, etc. Em tempo de execução, o ambiente que suporta esse tipo de mecanismo é que é o responsável pela conversão da chamada realizada em ações convenientes através das primitivas disponíveis nos serviços de baixo nível.

Uma forma estruturada de se implementar esse mecanismo é a seguinte: Primeiramente, os elementos cliente e servidor são construídos como se fossem ser "linkados". Uma descrição dos serviços disponíveis do servidor é processada gerando dois "stubs". O objetivo de um desses "stubs" é o de fazer as vezes do servidor junto ao cliente e vice-versa quanto ao segundo junto ao servidor. O código dos dois "stub" são "linkados" junto ao cliente e ao servidor. Observa-se então que nem o cliente nem o servidor utilizam explicitamente qualquer mecanismo de comunicação para a troca de dados. Os "stubs" é que comunicam-se entre si dando transparência aos seus respectivos usuários.

A construção e uso de uma aplicação distribuída baseada em mecanismos tipo RPC possui três fases: compilação, associação aos mecanismos de suporte e execução. A primeira fase, compilação, diz respeito a descrição das interfaces existentes para acesso aos serviços disponíveis. Esta descrição pode ser feita numa linguagem adequada o que permite o uso de mecanismos automáticos para a geração do código dos "stub". Nessa linguagem são especificados o nome das rotinas e os nomes e tipos dos parâmetros de entrada e de saída. A fase de associação tem lugar quando o servidor disponibiliza seus serviços ao ambiente e o cliente, por sua vez, efetua os procedimentos necessários para que possa indicar o seu desejo de acessar os recursos do referido servidor. Até essa hora os aspectos de localização, escolha dos protocolos para acesso aos serviços tem que ser estabelecidos. Até a última fase - fase de execução - os mecanismos de transporte, de controle e de representação dos dados tem que ser estabelecidos. A vantagem de se postergar até o último momento possível a escolha entre as opções disponíveis tem por objetivo permitir o uso de um leque bem amplo de mecanismos de suporte: protocolos de comunicação, mecanismos de acesso, etc. Vide NOTKIN [14] para esse tipo de abordagem.

### 4.2.3) LIMITAÇÕES DO MECANISMO

Apesar do comentário em [] sobre o consenso do uso de RPC para se acomodar o problema de heterogeneidade, não devemos deixar de considerar algumas limitações nesse tipo de abordagem. Essas limitações começam a surgir quando queremos tornar um determinado tipo de serviço disponível a uma comunidade heterogênea de usuários. Mais especificamente quando se deseja implementar níveis mais elaborados de acesso a esses serviços. Como, por um lado, o grau de complexidade e conceitos variam enormemente de

sistema para sistema e, por outro, deseja-se obter um bom desempenho, o uso desse mecanismo pode levar a um impasse.

A dificuldade começa pela escolha do tipo de interface de serviço que deve ser feita disponível. Um bom exemplo é o da interface para a acesso a um sistema de arquivos. A definição do tipo, formas de acesso, restrições de acesso, etc, varia muito de um sistema para outro. Podemos optar por uma interface de serviço altamente elaborada. Isto causa dois tipos de efeitos: Pode-se simplificar a implementação de um cliente com essa necessidade e complicar o acesso para sistemas menos elaborados. Isso sem falar na própria complexidade que pode ser a construção do serviço em diversos tipos de possíveis unidades prestadoras. Podemos,por outro lado, optar por uma interface simples o que poderia acarretar uma operação ineficiente, em virtude de um número elevado de chamadas aos serviços básicos disponíveis. Uma outra opção seria a de dotarmos o servidor com um conjunto de opções de interface para acesso a seus serviços. Ainda que se pudesse pensar em soluções de compromisso do tipo: um conjunto com alguns tipos de interfaces imaginemos o trabalho para o suporte e evolução dos programas servidores e dessas interfaces.

Claramente precisaríamos de mecanismos que pudessem ter o lado amigável de uma RPC quanto a aspectos de programação, mas que por outro lado permitissem que o grau de complexidade das interfaces existentes no sistema não fosse o produto da complexidade de cada interface particular existente. No que se segue veremos uma abordagem de um mecanismo que potencialmente soluciona adequadamente esse tipo de problema.

## 4.3) AVALIAÇÃO REMOTA DE FUNÇÕES: RFE

### 4.3.1) DEFINIÇÃO

O uso de linguagens de programação de alto nível viabilizou o uso de sistemas computacionais em larga escala. Primeiro porque permitiu a concentração em aspectos lógicos da solução dos problemas. Um outro ponto importante foi a possibilidade de se portar um programa desenvolvido para um determinado sistema (hardware + software básico) para outros completamente diferentes entre si e do sistema alvo original. A condição necessária para que isso possa ocorrer é a existência, em cada tipo diferente de sistema, de elementos - compiladores e/ou interpretadores - que interpretem para o novo ambiente o procedimento a ser realizado. Essa possibilidade de se executar um mesmo programa em diversos tipo de estações é que deu origem à idéia de se usar RFE - Remote Function Evaluation - para a solução do problema de se construir os serviços de uma forma estruturada em ambientes heterogêneos.

A razão para seu uso é a mesma de uma linguagem de programação: ter a capacidade de executar uma infinidade de procedimentos diferentes a partir de um vocabulário disponível reduzido. Como se verá adiante, na descrição de seus elementos, é possível resolver de forma elegante o problema da identificação de onde situar as interfaces e como distribuir as tarefas entre usuário e servidor. Nesse tipo de abordagem a troca de informações é realizada através da programção mútua entre as estações cliente e servidora. Por exemplo, uma estação cliente constrói uma expressão que traduz um determinado procedimento a ser realizado. Essa expressão é, em princípio, uma concatenação lógica de funções de um conjunto previamente disponível no servidor. O servidor decodifica a expressão executando o procedimento pedido nesse processo. Como resposta o servidor monta uma expressão resposta que pode inclusive conter

procedimentos a serem realizados pela estação cliente. Essa abordagem dá uma perspectiva animadora quanto a formas bastante elaboradas para o tratamento de situações de exceção. Outra boa perspectiva diz respeito a capacidade de evolução do sistema, isto porque proporciona um grau de independência entre os sistemas envolvidos. As idéias aqui discutidas tomam como referência o trabalho de FALCONE [11]

### 4.3.2) ELEMENTOS DA RFE

Um mecanismo do tipo de RFE possui três elementos básicos. O primeiro deles é a utilização de uma linguagem para a construção das expressões a serem trocadas entre as estações envolvidas. Essa linguagem é que estabelece o grau de maleabilidade do sistema. Um mesmo servidor pode realizar os trabalhos para uma comunidade de clientes heterogêneos, sem que se tenha definido previamente os elos ou interfaces entre os módulos envolvidos.

As expressões escritas na linguagem definida serão submetidas ao segundo elemento dessa arquitetura. Esse segundo elemento é o avaliador das expressões. Em cada estação, seja ela servidora seja ela cliente, deve existir pelo menos um desses elementos. Esse é o elemento responsável pela execução das expressões escritas na linguagem de comando. O processo de decodificação é realizado através das regras de concatenação existentes e através da invocação das rotinas do terceiro elemento envolvido. Esse terceiro elemento compreende a biblioteca de rotinas existente. Essa bibliioteca é que caracteriza o tipo de serviço a ser realizado.

Na realidade existem dois tipos de bibliotecas de rotinas disponíveis: O primeiro tipo diz respeito as rotinas para construção das expressões. Ela contém as primitivas para se permitir o encadeamento lógico das primitivas disponíveis. Essas rotinas estão relacionadas com a capacidade de se construir um trecho iterativo de programa,

por exemplo. O outro tipo de biblioteca é aquele que contém as primitivas para acesso a um serviço específico. Por exemplo, num serviço de arquivos, a biblioteca englobaria as primitivas para acesso ao sistema de arquivos da estação. Existe ainda um terceiro tipo de biblioteca, que seria aquela que um determinado cliente poderia, dinamicamente, passar a um servidor. Esse tipo de funcionalidade seria extremamente interessante, pois permitiria que as expressões pudessem ser reduzidas. Num ambiente heterogêneo principalmente, um determinado tipo de cliente, com um tipo de serviço muito mais complexo que o conjunto de primitivas básicas poderia programar procedimentos repetitivos no servidor aliviando o tráfego de mensagens no sistema e o trabalho de construção das expressões.

### 4.3.3) RPC X RFE

Comparando os dois tipos de abordagens poderíamos ressaltar os seguintes pontos: O mecanismo tipo RFE adapta-se muito bem àquele conceito de se fazer um esforço progressivo para a integração de novos tipos de elementos no sistema. As interfaces podem ser mantidas num nível de complexidade mais baixo inicialmente, sem ter isso como conseqüência um ônus elevado em termos de desempenho para a integração de futuros elementos. Esse relaxamento dos requisitos para a interface de acesso a um determinado serviço é de importância capital. Num esquema do tipo RPC a solução se orientava segundo três tipos de enfoque: soluções caso a caso; soluções com tendência a uma visão homogeinizadora; soluções que implicavam numa fragmentação lógica do ambiente.

Como o nível de compatibilidade foi transferido para a linguagem de comandos a capacidade do sistema evoluir passa a ser um aspecto ligado principalmente à linguagem. O sistema como um todo passa a ser muito mais modular. Por exemplo a otimização do avaliador que interpreta a linguagem traz um benefício global. Uma melhora

funcional de uma determinada biblioteca permite que um serviço seja estentido para operar com novos tipos de objetos, etc.

Vale ressaltar que as técnicas desenvolvidas para o mecanismo de RPC podem vir a ser utilizadas para a RFE. Por exemplo, a linguagem de comandos, até por possuir uma sintaxe própria, provavelmente desconhecida do usuário, pode ser combinada, na construção dos sistemas com algumas técnicas desenvolvidas para a RPC. A metodologia de geração de "stubs" para acessar os serviços de "baixo nível" disponíveis poderia ser empregada..

## 4.4)IMPLEMENTAÇÃO PROSPECTIVA

### 4.4.1) OBJETIVOS

Decidimos realizar uma implementação, ainda que simplificada, de um sistema com uso de RFE com três objetivos básicos: Testar alguns conceitos, avaliar o grau de complexidade de construção e compreender melhor o problema.

### 4.4.2) AMBIENTE

O ambiente alvo proposto era composto por Microcomputadores IBM PC compatíveis com sistema operacional MSDOS. Esse ambiente alvo inicial permitiu que nos concentrássemos nos aspectos de construção do sistema de teste uma vez que já possuíamos placas e software de comunicação disponíveis.Para a programação do sistema foi utilizada a linguagem C.

### 4.4.3) ARQUITETURA

Para a definição e construção do protótipo vamos ainda tomar como referência o trabalho de FALCONE [11] já citado. Os autores tomam como paradigma para a definição da

linguagem de comando a linguagem de programação LISP. Um dos fatores que favoreceram a escolha desta linguagem foi a sua ortogonalidade quanto ao tratamento de expressões. Todas as expressões em LISP são encaradas como listas. Pouca distinção é mesmo feita entre uma função e outro elemento qualquer. Assim uma determinada função, com seus parâmetros, nada mais é que uma lista com elementos dispostos segundo algumas regras. A partir de outros estudos BORROW e CLARK [18] é também mostrada a possibilidade de se obter formas bastante compactas para a transação das expressões entre clientes e servidores.

Outro ponto interessante é que o próprio avaliador é um elemento já conceitualmente presente na estrutura da linguagem. O algoritmo básico do avaliador - um elemento basicamente interativo - na linguagem é algo do tipo:

```
Faça sempre
     aceita expressão de entrada
     avalia expressão
     apresenta resultados
Fim Faça
```

Ora, o algoritmo básico do elemento servidor é algo bem análogo. As expressões são recebidas através do subsistema de comunicação e processadas pelo avaliador que as interpreta, num estilo recursivo, devolvendo ao final os resultados ao gerador da requisição.

### 4.4.4) A IMPLEMENTAÇÃO

Foram construídos três elementos básicos.O primeiro desses elementos cumpria as funções do elemento servidor. O seu papel era o de, passivamente, aguardar as requisições vindas de um possível usuário. O segundo elemento básico era o que enviava as expressões para serem analisadas pelo servidor. O terceiro elemento foi um compilador para as expressões tipo LISP, que facilitou a

construção dos programas de teste. Descreveremos a seguir os pontos mais importantes na implementação do elemento servidor.

4.4.4.1) GERENCIAMENTO DE MEMÓRIA

Optamos nessa implementação por não adicionar a complexidade do tratamento do gerenciamento de memória para a avaliação de expressões que excedessem o tamanho de um pacote de mensagem - (1050 bytes no caso). Uma expressão que extrapolasse o tamanho máximo da mensagem além do problema de fragmentação da mensagem e da administrção dos buffers de comunicação, traria ainda problema de se concatenar "offsets" usados para o acesso às variáveis durante o processo de avaliação da expressão.

Assim o aspecto de gerência da memória diz respeito tão somente aos tipos e classe das variáveis disponíveis para a construção de expressões, a forma de sua alocação e liberação e o controle de acesso sobre as mesmas.

4.4.4.2) TIPOS E CLASSES

Com relação a classe dessas variáveis temos seis possibilidades:

- Valores de entrada - sua função é de conter valores constantes numa expressão para o transporte de dados.

- Variáveis Locais - são elementos alocados explicitamente, cuja função é a de armazenar valores produzidos no decorrer da avaliação da expressão.

- Parâmetros de retorno - São semelhantes às variáveis locais só que o seu resultado final é passado de volta na mensagem de retorno.

- Referência - Aponta um outro elemento na expressão. No caso de um tipo vetor evitaria a existência de múltiplas cópias de um mesmo elemento.

- Variáveis de sessão - São elementos que armazenam valores com tempo de vida maior que uma única expressão. Tanto a sua definição quanto seu acesso podem estar sujeitos a controle de direitos.

- Variáveis de ambiente - São elementos que armazenam informacões sobre o estado do servidor. Como no caso anterior podem existir mecanismos para controle de acesso.

Essas classes podem estar aplicadas a quatro tipos de objetos: Inteiro, Nil, Vetor, Lista.

Variáveis locais e parâmetors de retorno precisam ser explicitamente alocados para serem referenciados. Os outros dois tipos de variáveis podem ser alocados - dependendo de direitos para fazê-lo - ou podem já ter sido previamente definidos para o estabelecimento de contexto de avaliação da expressão.

Uma expressão pode conter além dos tipos citados um elemento que traduz a operação que deve ser realizada sobre um conjunto de parâmetros. Esse elemento caracteriza um tipo denominado de função.

O descritor básico de qualquer tipo possui 8 bytes de tamanho.Seu formato é o que se segue:

| CAMPO | TAMANHO |
|---|---|
| Classe | 1 byte |
| Tipo | 1 byte |
| Tamanho | 2 bytes |
| Complemento | 4 bytes |

Para os descritores de valores de entrada teríamos a seguinte configuração:

| CLASSE | TIPO | TAMANHO | COMPLEMENTO |
|---|---|---|---|
| VE | Inteiro | 4 | Valor de 32 bits |
| VE | NIL | 0 | 0 |
| VE | Lista | Total | Tam do Seg/Offset para elementos |
| VE | Func | 0 | Cod. Func./ Cod. Bib. |
| VE | Vetor | Total | Offset para elementos |

4.4.4.3) O AVALIADOR

Esse é o elemento chave na operação do mecanismo de RFE. O processo de avaliação se processa de forma recursiva num estilo semelhante ao avaliador numa linguagem Lisp. Uma diferença com relação ao algoritmo do avaliador da linguagem conforme WINSTON e HORN [19] é que a própria função é responsável por chamar o avaliador para a avaliação de seus parâmetros. Isto em oposição ao avaliador processar os parâmetros antes de invocar a função. Optamos por esse tipo de solução por permitir simplificar o gerenciamento de memória para armazenamento de valores intermediários para posterior fornecimento à função. Uma vez que esta sabe a natureza de seus parâmetros de entrada fica mais prático a mesma pedir sua avaliação e armazenagem intermediária.

Seu algoritmo é basicamente o que se segue:

```
avaliador (s)
  [
     Se (s é átomo)
          retorna s
     senão
          Se (s é Lista)
            Se (Primeiro elemento de s é função)
               retorna (rotina  (Bib[Prim  s],Func[Prim
s]))
            senão
               retorna (avaliador (Primeiro elemento de
s)
  ]
```

4.4.4.4) BIBLIOTECA PADRÃO

O conjunto de primitivas que permite a concatenação de funções para a construção das expressões compõe a biblioteca padrão da linguagem. Nessa biblioteca estão primitivas que permitem efetuar iterações, desvios, atribuições a variáveis, alocação de variáveis, etc. Abaixo segue uma lista com os nomes das funções e a sua finalidade.

1) Set (s)

Parâmetro de entrada - s é uma lista com pares de elementos.

Função - realiza a atribuição do segundo elemento de cada par no primeiro. O segundo elemento pode ser uma expressão qualquer e o primeiro tem que ser uma variável adequada.

2) IF (s)

Parâmetro de entrada - s é uma lista com três elementos.

Função - avalia o primeiro elemento e avalia o segundo ou o terceiro elemento de acordo com o resultado.

3) WHILE (s)

Parâmetro de entrada - s é uma lista com dois elementos.

Função - enquanto o primeiro elemento resultar verdadeiro o segundo elemento é avaliado.

4) DCL_VAR (s)

Parâmetro de entrada - s é uma lista com um elemento.

Função - A partir de informações extraídas do segundo elemento são alocadas as variáveis e inicializados seus valores.

5) MAIS (s)

Parâmetro de entrada - s é uma lista com dois ou mais elementos.

Função - os elementos são avaliados e seus valores, de tipo obrigatoriamente inteiro são somados.

6) MENOS (s)

Parâmetro de entrada - s é uma lista com dois elementos.

Função - os elementos são avaliados e o resultante do segundo é subtraído do primeiro. Ambos precisam ser do tipo inteiro.

7) VEZES(s)

Parâmetro de entrada - s é uma lista com dois ou mais elementos.

Função - os elementos são avaliados e seus valores, de tipo obrigatoriamente inteiro são multiplicados.

8) DIVIDE (s)

Parâmetro de entrada - s é uma lista com dois elementos.

Função - os elementos são avaliados e o resultante do primeiro é dividido pelo segundo. Ambos precisam ser do tipo inteiro.

9) PROG (s)

Parâmetro de entrada - s é uma lista com elementos que são outras listas.

Função - permitir a construção de comandos compostos, ou seja, em última instância o próprio programa.

4.4.4.5) BIBLIOTECAS DE SERVIÇO

As bibliotecas de seviço permitem, como o próprio nome indica, a definição das primitivas básicas de acesso a um determinado serviço. Em nosso trabalho definimos duas bibliotecas básicas: a primeira contém primitivas para permitir o acesso ao sistema de arquivos da estação. As primitivas básicas dessa biblioteca são as seguintes: ABRIR, CRIAR, FECHAR, LER, ESCREVER. A segunda biblioteca permite que se use a partir de uma estação um recurso específico de outra. No caso o recurso utilizado é um processador aritmético. Foram definidas as seguintes funções: ASIN, ACOS, ATAN, SIN, COS, TAN, EXP, LOG, LOG10, SQRT, RMAIS, RMENOS, RVEZES, RDIVIDE. Todas essas funções operam sobre valores reais. No caso de valores inteiros operados juntamente com reais, os primeiros são convertidos para valores reais.

4.4.4.6) MECANISMOS PARA CONEXÃO E DISCONEXÃO

Foram criados dois procedimentos para que os serviços disponíveis pudessem ser administrados. O procedimento de acesso ao serviço exige que uma conexão com o elemento servidor seja primeiro realizada. Nesse momento é estabelecido o ambiente inicial de operação da seqüência

do processamento. Não foi ainda implementado, mas esse seria o momento adequado, para serem armazenadas as informações quanto a direitos de acesso aos objetos do sistema, as informações quanto a individualização do usuário, etc.

#### 4.4.4.7) TRATAMENTO DE ERROS

Como a implementação se vale de recursividade para a avaliação das expressões, foi criado um mecanismo para que a detecção e recuperação de situações de erro não obscurecesse a lógica do programa complicando a sua escrita. Em qualquer ponto se um erro é detectado a avaliação é interrompida a memória alocada para o processamento dessa expressão é liberada e uma resposta com indicações sobre o ponto onde foi detectado o erro é passada de volta para o usuário. Com isso é possível se ter informações para a depuração do programa.

#### 4.4.4.8) OPERAÇÃO DO SISTEMA

Para facilitar o trabalho de geração de programas de testes foi construído um programa que a partir de expressões escritas numa forma "Lisp-like" o "programa objeto" pudesse ser obtido. Para a transação com o elemento servidor criou-se um programa que enviava as expressões geradas ao servidor para processamento. Nesta implementação não se usa um avaliador para processamento de possíveis expressões vindas do servidor. Nem este por sua vez as constrói como elementos de respostas. Num sistema real esse seria um dos grandes apelos para o uso de RFE. Isto porque permite formas elaboradas no tratamento de situações de exceção no sistema. Seria possível, por exemplo, que o servidor programasse o usuário para operar segundo novas regras. Este por sua vez poderia negociá-las com o servidor. Procedimentos mais simples de balanceamento de carga podem ser imaginados, por exemplo. Situações onde os

elementos do sistema "aprendessem" e "ensinassem" uns aos outros. Enfim esse é um ponto bastante interessante que não foi explorado.

### 4.4.5) AVALIAÇÃO DA IMPLEMENTAÇÃO

Além das dificuldades que já mencionamos - gerenciamento de memória, tratamento de erros - que resolvemos em boa medida através das simplificações efetuadas, encontramos duas dificuldades adicionais. A primeira está relacionada com a definição de tipos. O mecanismo de se usar um vetor para simplesmente trocar o dado entre o requisitante e uma biblioteca pode em muito retirar a elegância própria que a solução tem. Por exemplo, vamos supor que desejássemos ter funções que tratassem reais ou inteiros de 64 bits.

Outro ponto, que de certa forma também está associado à definição de tipos, é o uso do elemento interpretador. Optamos, por dar preferência, neste trabalho, à implementação estruturada, empregando largamente a recursividade. No entanto, determinadas operações que são tão naturalmente expressas numa linguagem de programação possuem um "overhead" bastante significativo. Por exemplo, I = I + 1 implica em que chamemos quatro vezes o avaliador e duas funções diferentes. Isto além da identificação de tipos e campos a serem alterados. Para alguns tipos de serviços este overhead é inteiramente desprezível. Contudo, como mecanismo para solução geral esses pontos precisam ser melhor elaborados

No anexo I encontram-se os algoritmos das principais rotinas do programa.

#### 4.4.5.1) MEDIDAS DE DESEMPENHO

Embora essa implementação não tivesse por objetivo a construção de um sistema prático, é sempre interessante analisarmos o comportamento do sistema. Para tanto vamos

analisar o desempenho do sistema elaborado seguindo a orientação de LUDERER [20]. Segundo o autor a análise do desempenho possui três dimensões principais:

- Capacidade - A taxa em que trabalho útil pode ser realizado.
- Responsividade - Qual o atraso antes que uma ação desejada ocorra.
- "Overhead" - Quantos recursos do sistema são usados para a realização de uma atividade especificada.

Para que pudéssemos avaliar esses três aspectos nos valemos dos seguintes materiais:

- 2 microcomputadores tipo AT de 8 Mhz - Est1, Est2.
- 1 microcomputador tipo XT de 8 Mhz - Est 3.
- 3 Placas de rede local com velocidade de 2Mbits/s

Criamos dois programas de teste com os seguintes algoritmos:

```
prog1
        declara i inteiro
        declara j inteiro
        i = NUM_MAXIMO de ITERAÇÕES
        j = 0
        equanto (i diferente 0)
             j = j + 1
             i = i - 1
        fim equanto
fim prog1

prog2
        1 + 2
fim prog2
```

Para servidor foi usada Est1 ficando as outras duas para uso como fontes de requisição. Com os dois programas obtivemos a tabela que aparece a seguir:

| Programa | Iter Serv | Iter Est | Estacao | Tempo |
|---|---|---|---|---|
| Prog1 | 10000 | 1 | Est2 | 14 s |
| Prog1 | 10000 | 1 | Est3 | 14 s |
| Prog1 | 1 | 10000 | Est2 | 82 s |
| Prog1 | 1 | 10000 | Est3 | 102 s |
| Prog2 | 1 | 10000 | Est2 | 43 s |
| Prog2 | 1 | 10000 | Est3 | 63 s |

Vamos começar avaliando o aspecto de overhead do sistema. Com relação à complexidade lógica da tarefa Prog1 em pouco difere de Prog2. As tarefas adicionais de declaração de variáveis e do Loop de controle devem ser realizadas por necessidade do interpretador. No entanto, comparando os resultados da linha 3 com 5 (ou 4 com 6) vemos que o custo em termos de tempo no servidor é de quase 50 % do tempo de processamento. Felizmente esse tempo parece estar sendo consumido pela declaração de variáveis. Podemos observar isso a partir da linha 1: o tempo total é de 14 s, sendo o LOOP de controle uma fração desse valor.

A capacidade pode variar enormemente, de acordo com o que programamos. Por exemplo, podemos efetuar 10000 somas a cada 14 s ou a cada 102 s para Est3.

A responsividade também pode variar muito. Esse é um ponto crítico. Imaginemos que um usuário requisitante monitore o tempo de resposta para identificar uma situação de possível problema. Para uma requisição simples é mais fácil definirmos um limite superior para o tempo de resposta. No entanto, se a tarefa for complexa, a estimativa desse tempo pode não ser possível de ser realizada. Num sistema real a situação pode ser ainda pior, em virtude da disputa para o acesso aos recursos da estação servidora. Para uma requisição que demande muitos recursos ela poderia ser diferida uma série de vezes tornando mais complicada ainda a medida do tempo de resposta.

## 4.5) VISUALIZAÇÃO DO SISTEMA

A figura 7 evidencia a presença dos elementos da RFE em cada estação participante. Ela toma como referência a figura 3, quanto às características funcionais dos susbsistemas componentes. Assim fazemos as seguintes observações:

- O elemento avaliador está presente em todas as estações.

- O que diferencia um elemento servidor de um outro usuário é a posse de recursos específicos disponibilizados ao SOR. Isto é traduzido na presença de bibliotecas com as rotinas de acesso.

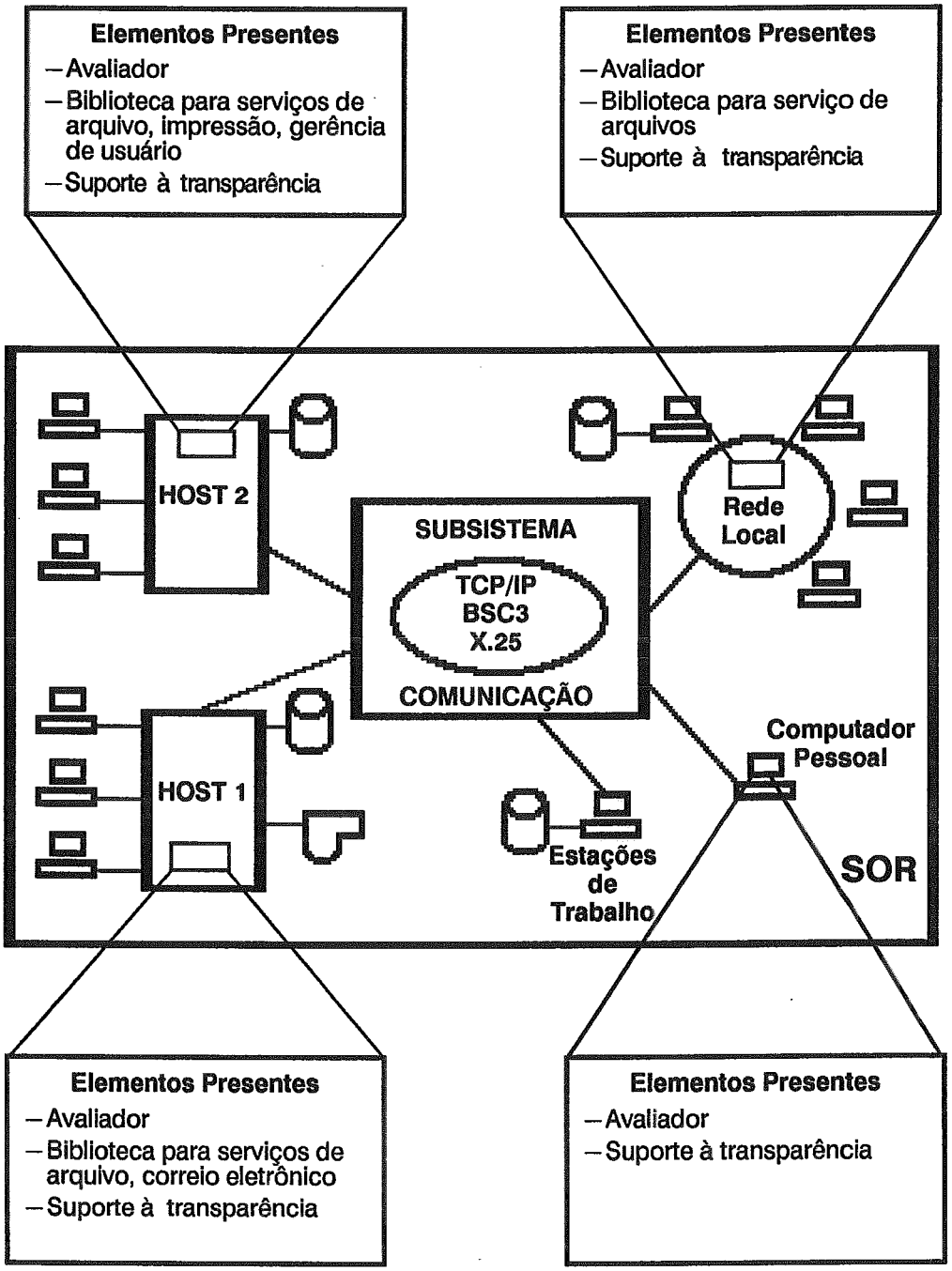


**Fig.7 Estruturação dos módulos de um SOR**

# CAPÍTULO V

# CONCLUSÕES

No início deste trabalho procuramos mostrar a heterogeneidade como um fato inevitável, na verdade mesmo desejável. Afinal, a rápida evolução dos sistemas de informação se viabiliza através do surgimento de novas técnicas em software de novos tipos de estações. Se esses novos sistemas não puderem ser integrados aos ambientes já existentes o processo de evolução ficará comprometido. Na medida em que se estuda o problema da integração de sistemas heterogêneos estamos a retirar os empecilhos para essa evolução. No contexto brasileiro em particular, a integração de sistemas heterogêneos constitui uma área importante onde se poderia exercer domínio tecnológico. Não só porque hoje não nos é possível investir na produção de equipamentos e sistemas, como também porque nos permitiria dominar uma área estratégica: a interconexão de sistemas, onde o custo é a aquisição de know-how pelo trabalho. Enfim seria uma área onde se poderia realizar efetivamente a engenharia de sistemas.

Para a construção desse sistema dois pontos chaves foram abordados nesse trabalho: o da administração do ambiente com ênfase na gerência do conjunto de usuários do sistema e no controle de acesso aos recursos disponíveis; os mecanismos para construção dos serviços.O problema de administração tem implicações técnicas em todas as partes do sistema. Logo estudar os problemas ligados à sua administração nos obriga a uma visão bastante ampla desse problema.

Com relação à definição de um patamar mínimo que uma estação tenha que atender para a participar do sistema, parece que nenhum obstáculo foi criado nos assuntos estudados. Do ponto de vista do controle de acesso, por exemplo, como um sistema simples não define logicamente o

conceito de usuário, a estação poderia se valer de uma outra, mais poderosa, que a acolhesse em seu domínio. Assim toda a complexidade da manutenção de chaves e elementos afins seria efetuada não por ela, mas pela estação hospedeira.

O mecanismo de RFE não parece impor complexidade relevante na estação usuária. Faria parte do patamar mínimo a presença do avaliador na estação. As partes envolvidas poderiam trocar informações quanto a suas capacidades no tratamento de situações de exceção, limitando-se a necessidade de recursos para a execução das tarefas. Outro fator que auxilia na minimização do patamar de entrada é a modularização no fornecimento e uso dos serviços. Cada um paga, em termos de complexidade, aquilo que usa ou oferece.

Após termos estudado esse assunto podemos tentar agora agora definir algumas características que novos sistemas operacionais poderiam trazer embutido. A necessidade de transparência no acesso aos recursos dispersos no sistema foi colocado como um ponto importante para viabilizar a integração de sistemas heterogêneos. No entanto, a identificação de objetos locais e remotos, ou melhor, de objetos pertencentes ao seu domínio é um problema ligado às regras paticulares de cada Sistema Operacional componente. Essas regras são tão próprias a cada sistema como o são as regras para a nomeação de seus objetos. Seria interessante que os sistemas operacionais que ainda venham a surgir, tragam interfaces definidas para que rotinas pudessem ser invocadas quando fosse identificado que um acesso é requerido sobre um objeto não presente em seu domínio.

Outra característica desejável seria a presença de primitivas que dessem suporte à construção dos serviços de controle do conjunto de usuários do sistema. Uma parte desse assunto talvez devesse mesmo ser alvo dos estudos de padronização como por exemplo a definição dos atributos que

permitissem a caracterização de um usuário de sistema para sistema.

Do ponto de vista da implementação dos serviços, seria interessante que os sistemas pudessem fornecer o suporte para que o executor de um serviço personificasse um usuário no sistema hospedeiro. O motivo é aquele já apresentado, o de permitir ao próprio Sistema Operacional efetuar o controle de acesso a seus objetos. Num sistema que admitisse a sua programação a partir um elemento remoto (RFE), essa funcionalidade seria chave. A integridade do sistema é garantida pelo próprio sistema. Nenhuma situação nova, com relação ao tópico de segurança, é criada. Trata-se apenas do sistema executando um programa construído pelo próprio usuário.

Como continuidade para esse trabalho, dada a abrangência do assunto, existem, além da elaboração dos tópicos acima mencionados, diversos pontos que tivemos que deixar a descoberto. Na administração do sistema enfocamos apenas alguns pontos da questão. A contabilização do uso dos recursos, por exemplo, não foi abordada. Outros tópicos já mencionados no terceiro capítulo também tem muito campo para evolução e sedimentação. Um maior detalhamento e trabalhos práticos sobre os pontos vistos também poderiam ser efetuados. Na verdade, todo esse estudo só poderá fazer sentido se as oportunidades que ele oferece puderem ser percebidas e aproveitadas num estudo amplo envolvendo todas as áreas a que está ligado.

# ANEXO I

## ALGORITMOS DO PROGRAMA SERVIDOR

Abaixo seguem as descrições dos tipos e dos algoritmos das rotinas principais. Como o programa foi construído em linguagem C, usando-se extensivamente tipos particulares, a linguagem usada para a descrição dos algoritmos foi nela baseada. Esses tipos - structure, union - foram usados por acomodarem bem a forma tipo Lisp uniforme da linguagem de comandos tratar os seus dados. Na descrição dos algoritmos fizemos algumas simplificações. Uma delas foi de, onde a lógica do programa não deixasse dúvidas, referenciar union em estruturas diretamente, sem indicar os elementos sintáticos intermediários, . Também nos valemos somente de uma única notação para os mesmos elementos - estrutura.elemento, ao invés de estrutura->elemento.

Definição das principais constantes

- Indicadores de tipos
  - VETOR
  - LISTA
  - INTEIRO
  - FUNÇÃO
  - REAL

- Indicadores de classe
  - VARIÀVEL_LOCAL
  - VARIÀVEL_DE_SESSÃO
  - VARIÀVEL_DE_AMBIENTE
  - PARÂMETRO_DE_RETORNO
  - VALOR_DE_ENTRADA
  - REFERÊNCIA

- Códigos de erro

INV_PAR

INV_PAR_CNT

INV_CLASE

INV_USER_ID

INV_SES_ID

SEM_MEMORIA

TIPOS_CONFLITANTES

Descrição dos tipos

```
byte unsigned char ;
word unsigned int ;
tipo_s struct {byte      classe ; /* Indica a Classe */
               byte      tipo ; /* Indica o tipo */
               word      tamanho ; /* Indica o tamanho */
               union     {long     valor ; /* Inteiros */
                          float    rval ; /* reais */
                          word     ind ;
                          struct   {word     offset ;
                                    word     seg ;
                                   } vet ; /* vetores */
                          struct   {word     offset ;
                                    word     seg ;
                                   } ptr ; /* referência */
                          struct   {word     func_id ;
                                    word     blb_id ;
                                   } func ; /* função */
                          struct   {word     tam_seg ;
                                    word     offset ;
                                   } lista ; /* listas */
```

Algoritmos do elemento servidor.

Funçõoes de apoio.

```
- atomo (s)
     tipo_s * s ;
  {
     /* Se s for átomo retorna TRUE
       senão retorna FALSO */
  }


- ret_func (ind_bib, ind_fnc)
     int ind_bib ;
     int ind_fnc ;
  {
     if (/* Existe a Biblioteca */)
       {
          selec[ind_bib] (ind_func) ;
          /* retorna ponteiro para rotina */
       }
  }

- peg_pos (s)
     tipo_s * s ;
  {
     /* Se s for REFERÊNCIA retorna elemento */
  }

- conexao ()
  {
     /* aloca uma estrutura de ambiente */
     /* preenche as informações do usuário */
     /* prepara mensagem de resposta */
     /* envia resposta */
  }
```

```
- disconexão ()
  {
     /* libera a estrutura de ambiente */
     /* prepara mensagem de resposta */
     /* envia resposta */
  }
```

Função de Avaliação

```
- tipo_s avaliador (s)
     tipo_s * s ;
  {
     tipo_s * s ;

     s = peg_pos (s) ;
     if (atomo (s))
       return *s ;
     if (s.tipo == LISTA)
       {
          s = s + s.offset ;
          if (s.tipo == FUNCAO)
            {
               rotina =ret_func(s.bib_id,s.func_id) ;
               s = (*rotina) (s) ;
               return s ;
            }
            else return avaliador (s) ;
       }
       else set_erro () ;
  }
```

Módulo principal do servidor.

```
- servidor ()
  {
     while (TRUE)
       {
          /* Espera chegar mensagens */

          switch (msg.func)
            {
               case CONEXAO:
                 conexao () ;

               case DISCONEXAO:
                 disconecta () ;

               case AVALIAÇÃO:
                 {
                    /* Pega dados da conexao */
                    sresult = avaliador(&msg.s) ;
                    /* prepara resposta */
                    /* envia resposta */
                 }
            }
       }
  }
```

Algoritmos da Biblioteca Padrão de rotinas.

Funções de apoio

```
- aloc_mem (tamanho)
     word tamanho ;
  {
     /* aloca o tamanho de memória pedido */
  }

- lib_mem (endereço,tamanho)
     word enedereço ;
     word tamanho ;
  {
     /* libera a área de memória definida */
  }

- pega_descritor ()
  {
     /*retira um descritor do pool para uma variável*/
  }
```

Funções da Biblioteca

```
- tipo_s set (s)
      tipo_s * s ;
  {
      tipo_s * s1 ;
      tipo_s sresut ;

      n = s.tamanho - 1 ;
      s += s.offset + 1 ;
      if (n != 3)
        {
           s1 = s + 1 ;
           sresult = avaliador (s1) ;
           if (s.tipo != sresult.tipo)
             ret_err (TIPOS_CONFLITANTES) ;
           if (s.tipo == vetor)
             /* atribui vetor */
             else
                s.valor = sresult.valor ;
        }
  }
```

```
- tipo_s if_func (s)
     tipo_s * s ;
  {
     tipo_s sresult ;
     byte n ;

     n = s.tamanho - 1 ;
     s += s.offset + 1 ;
     if (n != 3)
       ret_err (INV_PAR_CNT) ;
     sresult = avallador (s) ;
     if (sresult.valor != 0)
       s ++ ;
     else
       s += 2 ;
     sresult = avallador (s) ;
     return sresult ;
  }
```

```
- tipo_s while_func (s)
     tipo_s * s ;
  {
     tipo_s sresult ;
     byte n ;

     n = s.tamanho - 1 ;
     if (n != 2 )
       ret_err (INV_PAR_CNT) ;
     else
       {
          s += s.offset + 1 ;
          sresult = avaliador (s) ;
          while (sresult.valor != 0)
            {
               sresult = avaliador (s + 1) ;
               sresult = avaliador (s) ;
            }
          return sresult ;
       }
  }
```

```
- tipo_s dcl_var (s)
    tipo_s * s ;
  {
    tipo_s * s ;
    tipo_s sresult ;
    byte i,n ;

    if (s.tamanho != 2 )
      ret_err (INV_PAR_CNT) ;
    s += s.offset + 1 ;
    if (s.tipo != VETOR)
      n = s.tamanho ;
      else n = 1 ;
    for (i = 1; i<n; i++)
      {
        s1 = pega_descritor () ;
        (*s1) = (*s) ;
        if (s1.tipo == VETOR)
          .offset = aloc_mem (s1.tamanho) ;
      }
    return (*s1) ;
  }
```

```
- tipo_s mais (s)
     tipo_s * s ;

  {
     tipo_s sresult, stemp ;
     byte n,i ;

     n = s.tamanho - 1 ;
     if (n < 2)
       ret_err (INV_PAR_CNT) ;
     s += s.offset + 1 ;
     sresult = aval(s) ;
     if (sresult.tipo != INTEIRO)
       ret_err (INV_PAR) ;
     for (i=1; i<n; i++)
       {
          stemp = avallador (s[i]) ;
          if (stemp.tipo != INTEIRO)
            ret_err (INV_PAR)
          else sresult.valor += stemp.valor ;
       }
     return sresult ;
  }
```

```
- tipo_s menos (s)
     tipo_s * s ;

  {
     tipo_s sresult, stemp ;
     byte n,i ;

     n = s.tamanho - 1 ;
     if (n < 2)
       ret_err (INV_PAR_CNT) ;
     s += s.offset + 1 ;
     sresult = aval(s) ;
     if (sresult.tipo != INTEIRO)
       ret_err (INV_PAR) ;
     for (i=1; i<n; i++)
       {
          stemp = avaliador (s[i]) ;
          if (stemp.tipo != INTEIRO)
            ret_err (INV_PAR)
          else sresult.valor -= stemp.valor ;
       }
     return sresult ;
  }
```

```
- tipo_s vezes (s)
     tipo_s * s ;

  {
     tipo_s sresult, stemp ;
     byte n,i ;

     n = s.tamanho - 1 ;
     if (n < 2)
       ret_err (INV_PAR_CNT) ;
     s += s.offset + 1 ;
     sresult = aval(s) ;
     if (sresult.tipo != INTEIRO)
       ret_err (INV_PAR) ;
     for (i=1; i<n; i++)
       {
          stemp = avaliador (s[i]) ;
          if (stemp.tipo != INTEIRO)
            ret_err (INV_PAR)
          else sresult.valor *= stemp.valor ;
       }
     return sresult ;
  }
```

```
- tipo_s divide (s)
    tipo_s * s ;

  {
    tipo_s sresult, stemp ;
    byte n,i ;

    n = s.tamanho - 1 ;
    if (n < 2)
      ret_err (INV_PAR_CNT) ;
    s += s.offset + 1 ;
    sresult = aval(s) ;
    if (sresult.tipo != INTEIRO)
      ret_err (INV_PAR) ;
    for (i=1; i<n; i++)
      {
        stemp = avaliador (s[i]) ;
        if (stemp.tipo != INTEIRO)
          ret_err (INV_PAR)
        else sresult.valor /= stemp.valor ;
      }
    return sresult ;
  }
```

```
- tipo_s prog (s)
     tipo_s * s ;
  {
     tipo_s sresult ;
     byte n,i ;

     n = s.tamanho - 1 ;
     s += s.offset + 1 ;
     for (i=0 ; i<n; i++)
       if (s[i].tipo != LISTA)
          ret_err (INV_PAR) ;
     for (i=0 ; i<n; i++)
       sresult = avaliador (s[i]) ;
     return sresult ;
  }
```

## REFERÊNCIAS BIBLIOGRÁFICAS

[1] TANENBAUM, A.S. e RENESSE, R. V., "Distributed Operating Systems", COMPUTING SURVEYS, Dec 1985.

[2] SVOBODOVA,L., Worshop Summary-Operating Systems In Computer Netwroks, OPERATING SYSTEMS REVIEW, April 1985.

[3] MULLENDER, S.J. e TANENBAUM, A.S., "Protection and Resource Control in Distributed Operating Systems", COMPUTER NETWORK, 8-1984.

[4] CHERITON, D.R. e ZWAENEPOEL,W., "Distributed Process Groups in the V Kernel", ACM TRANSACTIONS ON COMPUTER SYSTEMS, May 1985.b

[5] FORSDICK, H.C. et alll, "Operating Systems for Computer Networks", COMPUTER, pp 48-57, Jan 1978.

[6] PANZERI, F.e RANDELL,B., "Interfacing UNIX to Data Communications Networks", IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, Oct 1985.

[7] WALSH, D. et al, "Overview of the Sun Network File System", PROCEEDINGS OF THE USENIX WINTER CONFERENCE, 1985.

[8] BUTTERFELD, D.A. e POPEK,G.P., "Network Tasking in the Locus Distributed Unix System", PROCEEDINGS OF THE USENIX SUMMER CONFERENCE, 1984.

[9] TODA,I., "DCNA Higher Level Protocols", TRANSACTIONS ON COMMUNICATIONS, vol. COM-28, N 4, pp 575-584, 1980.

[10] TSAY, D.e MING T. L., "Mike: A Network Operating System for the Distributed Double-Loop Computer Network", IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, vol. SE-9, N 2, pp 143 154, 1983.

[11] FALCONE,J.R., "A Programmable Interface Language", ACM TRANSACTIONS ON COMPUTER SYSTEMS, vol. 5, N 4, pp 330- 351, 1987.

[12] "Information Processing Systems - Open Systems Interconnection - Basic Reference Model, Ref. No. ISO 7498-1984.

[13] LININGTON,P.F., "The Virtual Filestore Concept", COMPUTER NETWORKS 8-1984.

[14] NOTKIN, D. et alii, "Interconnecting Heterogeneous Computer Systems", COMMUNICATION OF THE ACM, vol. 31, N 3, pp 258-273, 1988.

[15] NEEDHAM, R. e HERBERT,A., "Report on the Third European SIGOPS workshop 'Autonomy or Interdependence in Distributed Systems'", OPERATING SYSTEMS REVIEW, April 1989.

[16] SALTZER, J.H. e and SCHROEDER, M.D., "The Protection of Information in Computer Systems", PROCEEDINGS OF THE IEEE, vol. 63, N 9, pp 1278-1308, 1975.

[17] NEEDHAM,R.M., "Adding Capability Access to Conventional File Servers".

[18] BORROW, D.G. e CLARK,D.W. "Compact Encodings of List Structure", ACM TRANSACTIONS ON PROGRAMMING LANGUAGES AND SYSTEMS, October 1979 Vol.1 No 2.

[19] PATRICK HENRY WINSTON, P.H. e HORN, B.K.P., "Lisp", ADDISON-WESLEY PUBLISHING COMPANY, 1984.

[20] LUDERER, G.W.R. et alii, "A Distributed Unix System Based on a Virtual Circuit Switch", ACM 1981 <== Verificar

[21] QUARTERMAN, J.S. et alii, "4.2BSD and 4.3BSD as Examples of the UNIX system", COMPUTING SURVEYS, Dec. 1985.

[22] BIRRELL, A.D., "Secure Communications Using Remote Procedure Calls", ACM TRANSACTIONS ON COMPUTER SYSTEMS, Feb 1985.

[23] WITTIE, L.D.e TILBORG, A.M.V., "MICROS, a Distributed Operating System for MICRONET, a Reconglgurable Network Computer", IEEE TRANSACTIONS ON COMPUTER, Dec 1980.

[24] ROWE, L.A. e BIRMAN, K.P., "A Local Netwrok Based on the UNIX Operating System", IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, March 1982.

[25] MONTEIRO, M.C.B., "Um Servidor de Arquivos para uma Rede Local", TESE COPPE M.Sc 1984.

[26] GONG, L., "On Security in Capability-based Systems", OPERATING SYSTEMS REVIEW, April 1989.

[27] SCHERR, A.L., "Structures for Networks of Systems", IBM SYSTEMS JOURNAL, vol. 26 N 1, pp 4-12, 1987.

[28] BIRREL, A.D. e NEEDHAM, R.M., "A Universal File Server", IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, vol. SE-6, N. 5, pp 450-453, 1980.

[29] STURGIS, H. et al, "Issues in the Design and Use of a Distributed File System".

[30] HOGAN, C.B. "Protection Imperfect: The Security of some Computing Envbironments", OPERATING SYSTEMS REVIEW, V22 N3 July 1988.

[31] MOFFET, J.D.e SLOMAN, M.S., "The Source of Authority for Commercial Access Control", COMPUTER, Fev 1988.

[32] PRESS, J., "Software Based Encryption for Local Area Networks", COMPUTER NETWORKS AND ISDN SYSTEMS, N 17, pp 187 192, 1989.