UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

PROTÓTIPO DE SOFTWARE DE COMUNICAÇÃO UTILIZANDO COMUNICAÇÃO EM GRUPO SPREAD

JOAREZ SCHMITT

BLUMENAU
2004

2004/2-24

JOAREZ SCHMITT

PROTÓTIPO DE SOFTWARE DE COMUNICAÇÃO UTILIZANDO COMUNICAÇÃO EM GRUPO SPREAD

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Paulo Fernando da Silva - Orientador

BLUMENAU
2004

2004/1-24

PROTÓTIPO DE SOFTWARE DE COMUNICAÇÃO

UTILIZANDO COMUNICAÇÃO EM GRUPO SPREAD

Por

JOAREZ SCHMITT

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Paulo Fernando da Silva, MSc. – Orientador, FURB
Membro:	Prof. Sérgio Stringari, MSc. - FURB
Membro:	Prof. Francisco Adell Péricas, MSc. - FURB

Blumenau, 17 de novembro de 2004

Dedico este trabalho à Kanokwan Nontapot pela grande amizade e sábios conselhos; e a Rudolf Heß que sempre foi um exemplo de lealdade, persistência e determinação aos seus ideais, mesmo quando estes aparentavam impossíveis ou lhe causariam grandes danos.

Conhecer não é suficiente, nós devemos aplicar. Desejar não é suficiente, nós devemos fazer.

Johann Wolfgang von Goethe

AGRADECIMENTOS

Ao meu orientador, Paulo Fernando da Silva, por me orientar de forma oportuna no desenvolvimento deste trabalho.

À Graig Foley, pela grande ajuda na utilização do mecanismo Spread durante a implementação do protótipo.

Aos professores que contribuíram de forma indireta através de seus ensinamentos ao longo do curso.

E a todos os amigos e colegas que contribuíram para a realização deste trabalho ou que me ajudaram durante a jornada de estudos possibilitando que eu alcançasse meus objetivos.

RESUMO

Este trabalho apresenta a especificação e implementação de um protótipo para comunicação através da troca de mensagens, transferência de arquivos e salas de *chat*. É apresentado o conceito referente à comunicação em grupo e o mecanismo Spread. No protótipo são empregados os paradigmas cliente/servidor, *peer-to-peer* (P2P) e comunicação em grupo. O protótipo encontra-se dividido nos módulos servidor e cliente. São demonstradas as etapas de desenvolvimento e as funcionalidades do protótipo.

Palavras chaves: Comunicação; Comunicação em Grupo; Mensagens; *Peer-to-Peer*; Spread.

ABSTRACT

This work presents a specification and implementation of one prototype for a communication system trough messages interchange, files transferences and chat rooms. It explains the concepts of group communication and the Spread mechanism. In this prototype are used paradigms client/server, peer-to-peer (P2P) and group communication. The prototype is divided in the modules server and client. The development steps and the functions of this are explained in the current material.

Key-Words: Communication; Group Communication; Messages; Peer-to-Peer; Spread.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

P2P – *Peer-to-Peer*

IP – Internet *Protocol*

IRC – Internet *Relay Chat*

MUT – *MultiUser Talk*

BBS – *Bulletin Board System*

RPC – *Remote Procedure Call*

FIFO – *First In, First Out*

DSM – *Distributed Shared Memory*

EVS – *Extended Virtual Synchrony*

LTS – *Lamport Time Stamp*

ARU – *All-Received-Upto*

RF – Requisito Funcional

RNF – Requisito Não Funcional

LISTA DE SÍMBOLOS

@ - arroba

% - por cento

ß - *eszet*

SUMÁRIO

1 INTRODUÇÃO

Com a popularidade da Internet, cada vez mais faz-se uso de programas de comunicação que permitem duas ou mais pessoas conversarem entre si, seja por mensagens de texto, voz ou vídeo, fazendo desta forma com que os telefones tenham um uso menor para a sua finalidade primordial que é a transmissão de voz.

Os programas de comunicação começaram a se popularizar com o aparecimento do *Internet Relay Chat* (IRC) juntamente com a Internet, durante a década de 90. Anteriormente ao IRC existiam outros programas, como por exemplo, o *MultiUser Talk* (MUT), usado em *Bulletin Board System* (BBS), mas estes sem muito destaque devido a vários problemas de funcionamento e a restrição de uso na Internet.

Com um maior conhecimento sobre a tecnologia *Peer-to-Peer* (P2P), em 1996 foi desenvolvido um programa de comunicação com um conceito totalmente diferente dos usados até então. O programa ICQ (abreviação fonética para *I Seek You*), desenvolvido pela empresa israelense Mirabilis (ICQ INSTANT MESSENGER, 2004), utilizou um servidor central no qual todos os clientes conectavam-se recebendo uma lista de endereços *Internet Protocol* (IP) referentes aos clientes que tinham interesse em estabelecer conexão. Desta forma, as mensagens não relevantes ao servidor são encaminhadas diretamente de um cliente ao outro, amenizando drasticamente o tráfego de informações no servidor.

Atualmente vários estudantes e professores encontram-se freqüentemente conectados à Internet, embora não estejam interconectados entre si, tornando assim qualquer possibilidade de troca de idéias e arquivos inexistente, ou lenta e rudimentar, quando existente. De forma geral, as pessoas recorrem ao sistema de *e-mail* quando desejam transmitir algum documento ou alguma informação, devido à completa difusão deste método mesmo entre os mais leigos. *E-mails* são realmente eficientes quando a necessidade resume-se a informar determinado fato, mas acabam se tornando totalmente inconvenientes quando se deseja obter um retorno imediato do destinatário, como em uma discussão.

Os sistemas de comunicação P2P atuais resolveriam o problema, porém não seriam interessantes para a utilização em uma instituição, visto que se tratam de programas com códigos-fonte fechados e proprietários, impedindo assim qualquer implementação, correção ou adaptação específica para a instituição. O fato de estes softwares existentes serem

administrados e mantidos por empresas privadas, gera também inseguranças e incertezas, já que estes podem deixar de existir e de funcionar a qualquer momento, tornarem-se pagos, ou até mesmo fazerem mal uso das informações enviando-as a destinatários desconhecidos.

Tendo em vista as circunstâncias até aqui apresentadas, propõem-se com o protótipo desenvolvido neste trabalho uma solução que possibilite a comunicação entre duas ou mais pessoas, transferências de arquivos e que possa servir de base para futuras implementações de novos recursos relevantes à Universidade Regional de Blumenau. Para se alcançar tais objetivos de forma eficiente, empregam-se tecnologias recentes como o *Peer-to-Peer* (P2P) e a comunicação em grupo.

1.1 OBJETIVOS DO TRABALHO

Tem-se como objetivo deste trabalho o desenvolvimento de um protótipo que permita a comunicação entre duas ou mais pessoas conectadas à Internet através de troca de mensagens e arquivos.

Os objetivos específicos do trabalho são:

a) disponibilizar troca de mensagens entre usuários;
b) disponibilizar transferência de arquivos entre usuários;
c) disponibilizar conversas de *chat* em tempo real entre dois ou mais usuários conectados ao sistema;
d) disponibilizar cadastro para novos usuários pelo próprio sistema;
e) disponibilizar busca por outros usuários.

1.2 ESTRUTURA DO TRABALHO

A seguir é apresentado um resumo do conteúdo abordado em cada capítulo deste trabalho.

O primeiro capítulo apresenta uma breve introdução a qual possibilita o leitor se interar do conteúdo que este trabalho se refere, juntamente com os objetivos, justificativas e importância do mesmo.

O segundo capítulo contextualiza as possibilidades de comunicação encontradas atualmente no mercado.

O terceiro capítulo apresenta um estudo das tecnologias empregadas no protótipo aqui apresentado.

O quarto capítulo descreve o protótipo desenvolvido neste trabalho, suas características, especificações, operacionabilidade, principais telas.

O quinto capítulo apresenta as conclusões e sugestões para que possíveis interessados possam dar continuidade a fim de adicionar novos recursos, melhorar sua performance, torná-lo mais seguro ou ainda incorporar qualquer outro benefício relacionado ao protótipo.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo encontra-se uma descrição detalhada sobre as principais tecnologias e conceitos que serviram como base para a elaboração do protótipo desenvolvido.

Inicialmente é abordado brevemente sobre a área na qual este projeto tem sua base, sistemas distribuídos, em seguida é feita uma explicação mais aprofundada sobre a sub-área de mensagens em grupo e para finalizar é visto o mecanismo Spread mais detalhadamente, o qual foi empregado neste protótipo.

2.1 SISTEMAS DISTRIBUIDOS E COMUNICAÇÃO EM GRUPO

De acordo com Coulouris (2001), redes de computador estão em todos os lugares. A Internet é uma delas, assim como as demais redes menores, que interligadas formam a Internet. Redes de telefonia celular, redes corporativas, redes fabris, redes em campus, redes domésticas, redes embarcadas, todas estas, sejam separadamente ou em combinação, agregam uma característica essencial, compartilhar recursos tanto de hardware, como de software, através de uma rede, seja com a intenção de aumentar o poder computacional ou por segurança.

Segundo Coulouris (2001), sistema distribuído pode ser definido como componentes de software ou hardware que, localizados em uma rede de comunicação de computadores, coordenam as ações internas da rede somente através da passagem de mensagens.

Em sistemas distribuídos não é levado em consideração como ou a que distância computadores encontram-se conectados em uma rede, podendo estarem por exemplo separados por continentes ou por uma simples sala em um mesmo prédio. A definição de sistemas distribuídos segue as significantes conseqüências:

a) concorrência: em uma rede de computadores, programas são executados simultaneamente. Enquanto uma pessoa pode estar trabalhando em um computador, outra no mesmo instante de tempo pode estar trabalhando em um outro computador, ambas compartilhando recursos como por exemplo páginas de Internet;
b) sem ‘clock’ global: quando programas necessitam cooperar eles coordenam suas ações por troca de mensagens, não havendo um horário padrão para sincronismo, sendo todo o processo baseado no tempo em que cada ação do programa ocorre;

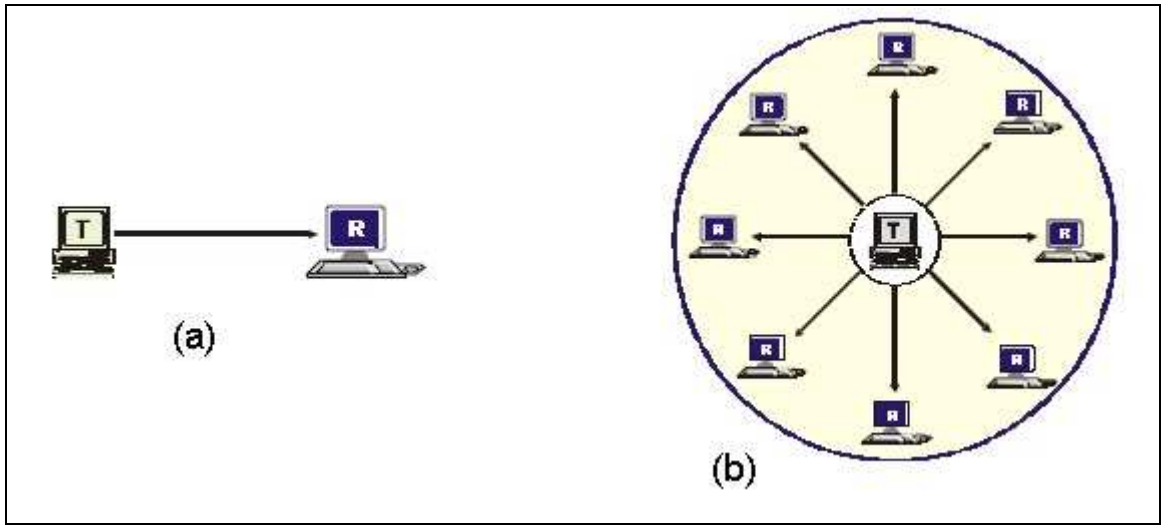
c) independência de falhas: ao contrário de redes convencionais, caso ocorra uma falha em uma máquina ou processo, esta falha é isolada dos outros membros, não resultando em uma paralisação de todo o sistema. Em alguns casos outros membros do sistema podem assumir a tarefa que estava sendo resolvida quando ocorreu a falha.

Conforme Goulart (2002), a comunicação entre os computadores interligados, pode ocorrer de diferentes formas, usando diversos protocolos tanto em ambiente de rede local como ambiente de redes de longa distância. Um caso particular de comunicação ocorre quando uma mesma mensagem deve ser enviada para diversos computadores na rede, entretanto o ambiente físico de rede não permite o *broadcast*, onde um único comando de envio manda a mensagem para todos os endereços da rede, ou ainda quando o *broadcast* é disponibilizado pela rede, a confiabilidade deste método não é garantida, podendo ocorrer alguma perda sem que o emissor da mensagem seja notificado da mesma. Com o intuito de atender a necessidade de comunicação de um para muitos, mantendo confiabilidade, escalabilidade e com formato transparente para quem desenvolve a aplicação, foi criado um novo paradigma chamado comunicação em grupo.

É fundamental, em sistemas distribuídos, a comunicação entre os diversos computadores que compõem este ambiente. O *Remote Procedure Call* (RPC) se aplica em especial para a comunicação que envolve dois processos. Em algumas situações é desejável que um processo possa se comunicar com diversos outros processos. Com RPC não se pode ter um único transmissor que envia de uma única vez para múltiplos receptores. Este mecanismo de comunicação, que trata múltiplas conexões, é chamado de comunicação em grupo.

Segundo Guerraoui e Schiper (1996), a comunicação em grupo é uma tecnologia utilizada para facilitar e prover alguns serviços que até então sem o uso deste mecanismo eram implementados pelo programador, como o gerenciamento de entrada e saída de um objeto ao grupo (*membership*) e primitivas de comunicação *multicast* e *broadcast*.

A comunicação em grupo nos permite ultrapassar a barreira da comunicação um para um, conhecida como comunicação ponto-a-ponto. Esta subárea de sistemas distribuídos responsabiliza-se para que determinada mensagem enviada a um grupo por um processo, seja entregue a todos os demais processos que compartilhem este grupo ao qual a mensagem foi destinada. Tanto o remetente da mensagem quanto os que exercem a função de destinatários, são considerados membros do grupo que compartilham. A figura 1 ilustra um comparativo entre a tradicional comunicação ponto-a-ponto e um-para-muitos onde se aplica a comunicação em grupo.



Fonte: Goulart (2002, p. 65)

Figura 1 – (a) Comunicação ponto-a-ponto (b) Comunicação um-para-muitos

De acordo com Tanembaum (1995), as dificuldades no desenvolvimento de sistemas de tempo-real, principalmente pela necessidade de hardware especializado, tornam o projeto destes sistemas muitas vezes inviável e com um alto custo. Para tentar diminuir tais dificuldades uma possibilidade é a utilização de sistemas distribuídos que minimizam em muito os problemas referentes a parte de comunicação e o tempo gasto, permitindo que o desenvolvedor possa voltar sua atenção à essência do projeto.

Segundo Figueiredo (1999), este novo paradigma conhecido como comunicação em grupo vem a cada dia sendo mais aceito. Este tipo de comunicação difere da comunicação ponto-a-ponto convencional, pois as informações passam a serem disseminadas para um grupo de participantes no sistema.

O mecanismo de comunicação em grupo é totalmente flexível e aberto quanto a mudanças, não há qualquer necessidade de uma pré-definição do funcionamento dos grupos aceitos, desprezando informações como por exemplo: membros existentes em cada grupo; a forma como estão conectados; em qual IP se encontram; e quais grupos devem ser criados.

O funcionamento do mecanismo ocorre em tempo de execução, desta forma novos grupos são criados ou excluídos conforme a presença ou não de membros ligados ao grupo. Os membros do serviço possuem completa liberdade também, podendo entrar ou sair dos grupos com total facilidade, além disto, não existe qualquer limitação quanto ao número de grupos a qual um membro pode pertencer simultaneamente. Para que haja esta ligação com o servidor e a comunicação seja uniforme para cada membro, todos os membros, bem como o gerenciador do serviço que interliga os membros, devem possuir um mecanismo de comunicação em grupo em comum sendo executado em paralelo. O gerenciador dos serviços de comunicação em grupo pode ser executado em um computador específico para esta finalidade, bem como pode ser executado em um dos membros do grupo, que irá utiliza o mesmo serviço, atuando como servidor e cliente ao mesmo tempo.

Conforme Goulart (2002), quando mecanismos de comunicação em grupo os processos tratam com coleções de outros processos de forma abstrata. Se um processo envia uma mensagem para um grupo de servidores, este processo não precisa se preocupar em quantos são os servidores nem onde eles estão localizados. Caso na chamada seguinte o número de servidores deste grupo e suas localizações tenham mudado, esta alteração permanece transparente para o processo que envia a mensagem para o grupo, sendo todo o tratamento realizado pelo mecanismo utilizado.

2.1.1 MÉTODOS DE IMPLEMENTAÇAO DE COMUNICAÇAO EM GRUPO

A implementação do mecanismo de comunicação em grupo pode ser dividida em basicamente três modos: *multicast*; *broadcast*; e *unicast*. Nem sempre tem-se a opção destes três modos, isto ocorre devido ao hardware no qual são executados não apresentar algumas características como um endereço IP de difusão pública. Uma breve descrição destes modos, é apresentada a seguir, pode-se encontrar mais detalhes do uso destas técnicas no item 2.1.3 referente ao gerenciamento na comunicação em grupos.

Conforme Oliveira (2002), *multicasting* é a técnica que possibilita o envio de uma mensagem para um grupo de usuários da rede.

De acordo com Martins e Moreira (2001), o uso da técnica de *multicasting* permite uma economia de largura de banda utilizada pelas aplicações, bem como economia de recursos em servidores multimídia.

Segundo Figueiredo (1999), o uso de aplicações multimídia com o protocolo IPv6 juntamente com a técnica de *multicasting* tanto em LANs como em WANs pode ser uma solução interessante, provendo um sistema altamente escalável e eficiente.

Em redes em que se pode configurar um endereço de *multicasting*, é possível utilizar-se deste endereço para difundir as mensagens, onde todas as máquinas que se encontram escutando este endereço irão receber a mensagem enviada, que é tratada pelo cliente do mecanismo de comunicação em grupo.

Conforme Goulart (2002), a implementação de aplicativos com mecanismos de comunicação em grupo utilizando *multicasting* torna-se muito mais simplificada, bastando somente abrir um endereço de *multicasting* para cada grupo.

Outra técnica que pode ser empregada é o *broadcasting*. Segundo Oliveira (2002), é chamado de *broadcasting* a técnica que permite o envio de uma mensagem para todos os usuários da rede.

O *broadcasting* pode ser usado para implementar comunicação em grupo, embora não seja tão eficiente, pois todos na rede vão receber as mensagens e cada processo terá que avaliar se aquela mensagem recebida é para o grupo ao qual ele pertence. Se não for, vai descartar a mensagem, mas neste caso o processo já gastou tempo e recursos analisando esta mensagem. Ainda assim, existe a vantagem de que um único pacote enviado atingir a todos os equipamentos da rede.

Por fim, tem-se a técnica de *unicasting*. Segundo Oliveira (2002), a técnica de *unicasting* permite o envio de uma mensagem de um *host* para um outro *host* específico.

Figueiredo (1999), descreve que o *unicasting* é o modo mais simples de transmissão de dados para múltiplos receptores. Esta técnica consiste em emitir uma cópia de dados para cada

destino, individualmente, e quanto maior o número de receptores, maior é a largura de banda utilizada.

Conforme Goulart (2002), caso a rede não suportar *multicasting* nem *broadcasting*, ainda é possível se implementar um sistema onde o transmissor vai enviar um pacote separado para cada um dos receptores do grupo. Para um grupo com N membros é necessário o envio de N pacotes ao invés de um único pacote como no *broadcasting* ou *multicasting*.

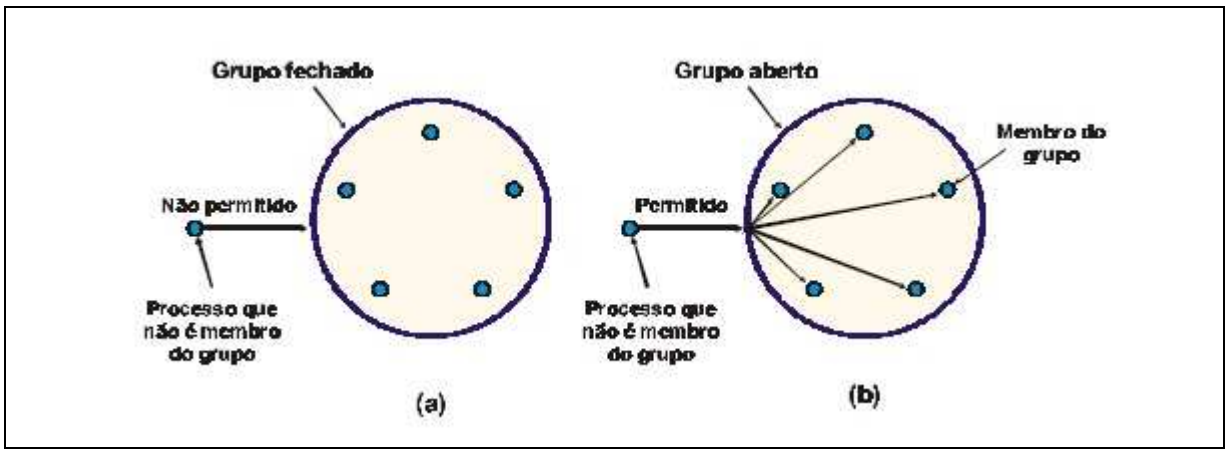
2.1.2 ORGANIZAÇÃO DOS GRUPOS

Os sistemas de comunicação em grupo são classificados em dois tópicos distintos referentes a sua organização. Os grupos podem ser abertos ou fechados, dependendo de suas restrições referentes ao uso ou não do canal de comunicação comum ao grupo para membros não pertencentes ao grupo. Podem também ser hierarquizados ou simétricos, cuja característica está relacionada à estrutura interna do grupo.

No caso de um sistema atuando em modo de grupo fechado, o envio de uma mensagem a todo o grupo de forma única é permitido somente aos membros pertencentes ao grupo em questão. Membros externos ao grupo não poderão utilizar este recurso, embora possam perfeitamente enviar as mensagens uma-a-uma a cada membro do grupo, o que acarretaria em maior consumo de banda de rede, portanto degradando a performance da rede, já que não estariam utilizando os benefícios da comunicação em grupo.

Já em sistemas de grupo aberto não há qualquer restrição quanto a isto, é possível o uso do serviço de mensagens em massa, tanto para membros internos quanto externos ao grupo. A figura 2 apresentação estes dois tipos de organização dos grupos.

A escolha quanto ao tipo de gerência, aberta ou fechada, é diretamente ligada à finalidade do sistema que utiliza o mecanismo. Quando se deseja utilizar o mecanismo em um sistema com processos que trabalhem de forma paralela, buscando uma solução em comum onde são totalmente irrelevantes mensagens exteriores ao grupo, o modo de grupo fechado aplica-se melhor, como exemplo típico pode-se citar o balanceamento de cargas aos quais os nodos trocam informações somente entre eles. Por outro lado, quando se necessita que um membro externo ao grupo envie para todos os membros do grupo a mesma mensagem, então se aplica o modo de grupo aberto, o que ocorre, por exemplo, em servidores replicados para processamento distribuído.

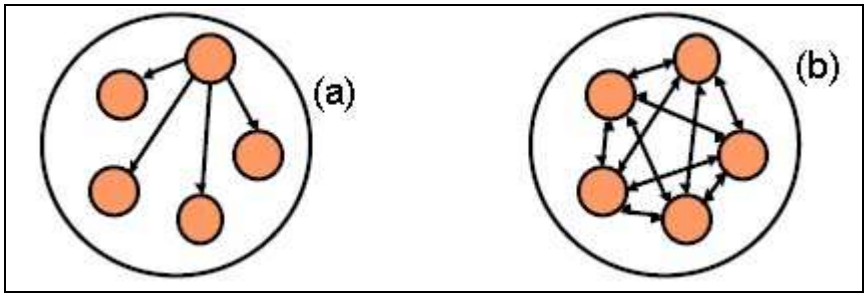


Fonte: Goulart (2002, p. 67)

Figura 2 – (a) Grupo fechado (b) Grupo aberto

Outra classificação divide os grupos em: grupos com processos hierarquizados, ou grupos com processos simétricos (também conhecido como grupo de semelhantes).

Grupos simétricos apresentam uma estrutura interna a qual todos os processos comportam-se de forma homogênea, sendo as decisões tomadas de forma coletiva, o que não ocorre em grupos de processos hierarquizados, onde existe um processo principal que exerce a tarefa de gerente e os demais processos ficam subordinados a este "gerente". No modo hierarquizado, todas as requisições são passadas primeiramente ao nodo gerente e cabe a este tomar a decisão sobre como proceder com a requisição recebida. Na seqüência, a figura 3 demonstra um comparativo entre o modelo hierarquizado e o simétrico.



Fonte: PUCRS (2002, p. 18)

Figura 3 – (a) Processos hierarquizado (b) Processos simétrico

De acordo com Goulart (2002), cada uma destas organizações tem suas próprias vantagens e desvantagens. Na comunicação em grupos de semelhantes não existe um ponto único de falha. Se um dos processos falhar, o grupo continua, apenas se torna menor com um desempenho inferior ao normal. A desvantagem é o *overhead* gerado pelas tomadas de decisões serem mais complexas. Para alguma decisão, tem que envolver todos do grupo,

gerando uma demora. No grupo hierárquico tem-se o oposto, sendo as decisões tomadas pelo coordenador. O coordenador especifica qual processo do grupo vai executar qual atividade, sendo que quando um processo termina sua tarefa, este avisa ao coordenador que está pronto para outra atividade. A perda do coordenador em um grupo hierárquico causa uma parada nas atividades do grupo.

2.1.3 GERENCIAMENTO NA COMUNICAÇÃO EM GRUPOS

Quando se utiliza o mecanismo de comunicação em grupo, existe a necessidade de um controlador de operações. Este controlador administra a criação e destruição dinâmica de grupos, bem como a inclusão e exclusão de membros a estes grupos. A implementação deste controlador pode ocorrer de duas formas distintas: centralizada ou distribuída.

Na implementação com o controle centralizado, todos os controles passam a serem manipulados por um servidor de grupos.

Segundo Goulart (2002), este método pode ser facilmente implementado, é de fácil entendimento e eficiente, embora possua a desvantagem de compartilhar a maior das deficiências dos sistemas centralizados, baixa confiabilidade por ter um único ponto de falha.

Em caso de um problema no servidor de grupos, todo o mecanismo fica comprometido, perdendo-se provavelmente a maior parte das definições dos grupos e interrompendo os processos em execução.

A escalabilidade para sistemas centralizados é baixa, limitando o crescimento dos grupos e membros.

Em contraste com o modo centralizado, tem-se a forma distribuída, a qual resolve os problemas encontrados no modo centralizado, mas diminuindo a performance devido ao aumento do *overhead* criado para manter-se atualizadas todas as replicações dos dados.

Em um grupo aberto, um processo de fora do grupo pode enviar uma mensagem para todos do grupo anunciando a sua presença. Em um grupo fechado, ao menos com relação à solicitação de participar do grupo, também será enviada uma mensagem para todos. Este grupo fechado está restrito a aceitar uma mensagem de algum processo fora do grupo, quando esta mensagem é de solicitação para participar do grupo. Uma mensagem direta sem antes

estar participando do mesmo grupo fechado, não seria aceita. Para um processo deixar de fazer parte de um grupo, basta enviar uma mensagem de adeus a todos do grupo.

Em caso de ocorrer uma falha em um dos membros de um grupo, este será incapaz de responder e avisar quanto ao seu desligamento do grupo como no procedimento normal, todos os demais membros deste grupo devem, notificarem-se por conta própria da perda daquele membro, descartando-o do grupo. Se ocorrer a perda de muitos membros simultaneamente, como, por exemplo, queda de parte da rede, incapacitando algum grupo, necessita-se de algum protocolo para reorganizar o grupo com os membros ainda em funcionamento.

Outros pontos relevantes quanto aos controles são o sincronismo das mensagens e sua relação com a entrada ou saída de um membro no grupo. As mensagens devem manter a mesma ordem de recebimento de quando foram enviadas e um membro só recebe mensagens a partir do momento em que integrar o grupo, não recebendo mensagens antes, e nem após seu desligamento do grupo.

Todo processo que for integrado ao grupo no momento t deverá receber a partir deste momento t, todas as mensagens enviadas. Da mesma forma quando deixa o grupo no momento z, todas as mensagens enviadas após z já não são mais endereçadas a este processo que deixou o grupo. Uma forma de resolver esta situação é transformar a operação de inclusão e exclusão de membros no grupo em uma mensagem especial de inclusão ou retirada que será transmitida de forma síncrona na seqüência das demais mensagens para o grupo, assim garantindo a ordem cronológica das mensagens.

Cada grupo deve possuir um identificador único para endereçamento. A forma de identificação pode ser implementada de três formas diferentes: endereçamento de rede; lista de endereçamento; e por predicado. A primeira forma, por endereçamento de rede, pode ser subdividida em três modos: *multicasting*, *broadcasting* e *unicasting*. O *unicasting* é independente do tipo de hardware de rede, sendo que os modos *multicasting* e *broadcasting* possuem dependência das configurações da rede para que possam ser empregados.

Quando a rede empregada dispõe de um endereço de *multicasting*, o endereço do grupo pode ser associado a este endereço de *multicasting*, desta forma toda mensagem enviada ao endereço de *multicasting* será recebida pelos computadores que compartilharem deste *multicasting*, fazendo com que todos os membros do grupo recebam a mensagem.

Se a rede suportar *multicasting*, o endereço do grupo pode ser associado com o endereço de *multicasting*, assim cada mensagem enviada ao grupo pode ser direcionada especificamente para aquele grupo identificado pelo endereço de *multicasting* e só vai ser recebida pelos computadores que fazem parte daquele endereço de *multicasting*. Caso na rede não exista o mecanismo de *multicasting*, entretanto exista o mecanismo *broadcasting*, uma alternativa é utilizar este endereço de *broadcasting* da mesma forma que o caso anterior, o diferencial é o envio da mensagem para todos os computadores ligados ao *broadcasting*, o que resultará em um aumento do processamento e do tempo para que a mensagem seja recebida, já que todas as máquinas terão que analisar se algum de seus processos faz parte ou não do grupo em foco, caso não seja relevante a nenhum dos processos encontrados no computador, a mensagem é descartada. Como último recurso tem-se o endereçamento por *unicasting*, onde o *kernel* da máquina que envia a mensagem deve possuir uma lista de todos os computadores que possuam processos relacionados ao grupo em uso. Através desta lista de endereçamento dos processos, é realizada uma conexão ponto-a-ponto a todos os membros do grupo para o envio da mensagem.

Um ponto importante é que mesmo nos três casos o processo que envia manda apenas uma mensagem para o endereço do grupo e a mensagem vai para todos os membros do grupo. A forma de como será transmitida a mensagem é problema do sistema operacional. O processo que envia não tem que se preocupar com o tamanho do grupo ou se a comunicação vai ser por *multicasting*, *broadcasting* ou *unicasting*.

O segundo método consiste em o processo que enviar a mensagem possuir uma lista com o endereçamento exato dos outros membros do grupo, como exemplo, o endereço IP das outras máquinas. Neste caso quando se envia uma mensagem é passado como parâmetro um ponteiro para a lista com o endereço dos destinos.

Este método tem o inconveniente de forçar o processo membro do grupo de saber precisamente quem faz parte do grupo, desta forma perdendo a abstração quanto à mudança de participantes do grupo, já que o processo do usuário final precisará atualizar toda lista de membros do grupo de forma explicita, o que não ocorre no primeiro método citado anteriormente, onde o *kernel* se responsabiliza em manter esta transparência para o processo do usuário final.

Como terceiro e último método tem-se endereçamento com predicado. Assim como no uso do broadcasting, a mensagem é enviada a todas as máquinas, entretanto junto com a mensagem é enviada uma expressão booleana a qual será analisada pelo destinatário e caso resulte em verdade a mensagem é aceita, caso contrario descartada. Para compor esta expressão booleana do predicado pode-se utilizar, por exemplo, desde variáveis locais, número da máquina receptora ou outros fatores.

Seria ideal se fosse possível ter um conjunto de primitivas comuns, tanto para comunicação ponto-a-ponto como para comunicação em grupo. Quando se usa RPC tem-se o envio e a recepção de mensagens usando duas simples primitivas tais como "*send*" e "*receive*", respectivamente. O que não ocorre na comunicação em grupo, onde uma mensagem enviada pode gerar diversas respostas.

O tipo de chamada pode ser classificado quando a três peculiaridades, elas são: quanto ao uso de buffer (*buffered* ou *unbuffered*); quanto à disponibilidade para múltiplas mensagens (bloqueada ou desbloqueada); e quando a confiabilidade (confiável ou não). Estas classificações são válidas tanto para conexões ponto-a-ponto, como para conexões de grupos.

Quando se utiliza o modo *unbuffered*, as mensagens são recebidas se o recebedor estiver preparado para receber, caso contrário a mensagem é automaticamente descartada pelo *kernel*. No modo *buffered*, ao contrário, tem se um *buffer* para alocar as mensagens recebidas para quando o recebedor estiver em algum processo e não puder atender a mensagem no momento da ocorrência. No modo bloqueado, o processo aguarda por uma mensagem para iniciar a atividade, oposto do não bloqueado. Caso necessite-se de confiabilidade quanto ao recebimento das mensagens, pode utilizar-se em diversos níveis. Para um nível em que não se espera qualquer aviso de recebimento de uma mensagem por parte do recebedor, utiliza-se o "*0-reliable*", quando basta a confirmação de apenas um recebedor, usa-se "*1-reliable*", para certificar-se que todos os recebedores obtiveram a mensagem tem-se o modo "*all-reliable*", e ainda pode-se personalizar a quantidade no modo "*m-out-of-n*", onde M representa a quantidade de confirmações mínimas que espera-se obter, e N a quantidade total de recebedores ($1<m<n$).

Um exemplo do uso deste último tipo de confiabilidade é o controle de consistência de informações replicadas por maioria, na qual $m=(n/2)$.

De um modo geral a escolha de qual tipo de chamada será utilizada é definida pelos projetistas do sistema. A parametrização das chamadas é fixa, não permitindo qualquer personalização nos parâmetros existentes.

Devido as dificuldades em manter a mesma semântica para formas de processamento tão diferentes, como comunicação ponto-a-ponto e para grupos, alguns projetistas incluíram novas primitivas como "*group_send*" e "*group_receive*", caso necessite-se uma resposta a uma chamada podemos usar "*getreply*" e após o envio pode-se utilizar "*getreplay*" repetidamente para coletar todas as respostas associadas.

2.1.4 PROPRIEDADES DA COMUNICAÇÃO EM GRUPOS

Assim como a maior parte dos bancos de dados, o mecanismo de comunicação em grupo também dispõe da propriedade da atomicidade, neste caso específico também chamado de *broadcast* atômico. Esta propriedade encarrega-se de que uma mensagem enviada a um grupo chega a todos os membros deste grupo ou nenhum, impedindo que determinada informação seja processada por somente parte dos nodos de um grupo. Quando utilizada esta propriedade, caso todos os membros recebam uma mensagem com exceção de um, mesmo que este um represente uma fração irrelevante, todo o processo é abortado e as mensagens que chegaram aos recebedores são descartadas. Para o projetista a quantidade dos membros é abstrata, sendo todo o processo representado por um único destinatário, a qual receberá ou não a mensagem.

A atomicidade é importante, por facilitar a programação de um sistema distribuído. Quando qualquer processo enviar uma mensagem para um grupo ele não deve preocupar-se se algum outro membro do grupo não recebeu esta mensagem. Com esta propriedade o processo que envia a mensagem para um grupo irá receber uma mensagem de erro se um ou mais integrantes do grupo teve problema no recebimento sendo que os demais vão ignorar esta mensagem.

Uma segunda propriedade corresponde ao tipo de ordenamento das mensagens. Para a comunicação em grupo deve-se ter o conceito de entrega de mensagem, além dos tradicionais enviar e receber. Isto se deve ao processo de entrega passar por tratamentos e controles para manter uma seqüência coerente das mensagens desde o emissor até o destinatário.

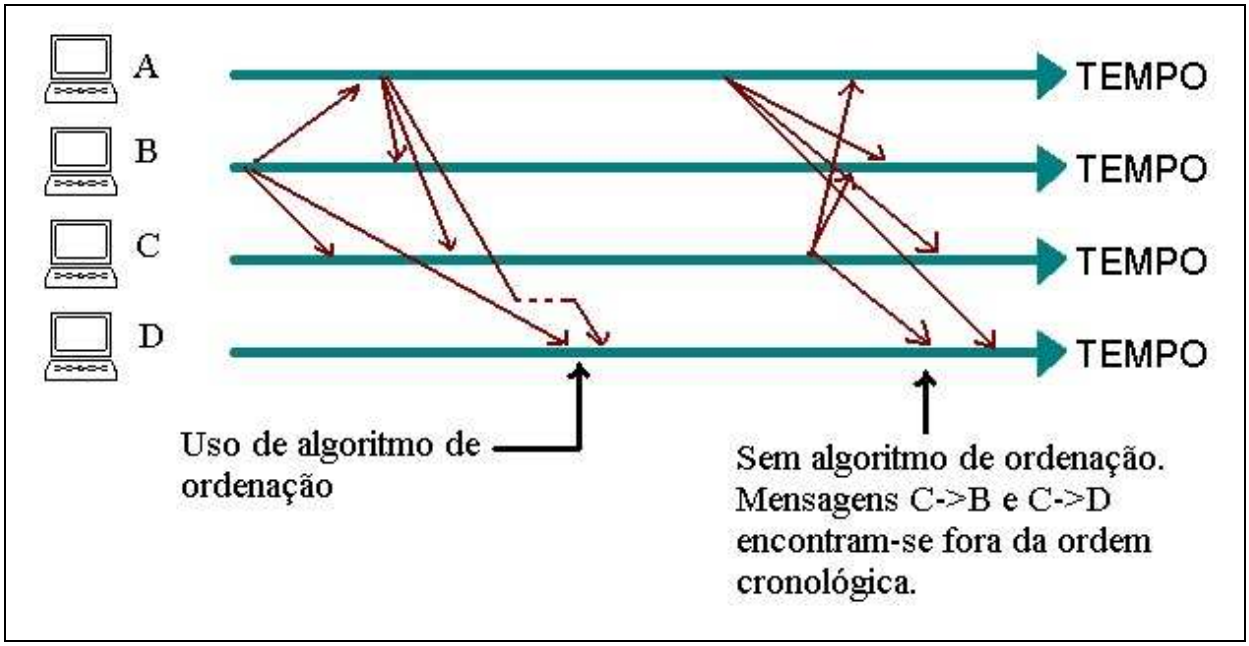


Figura 4 – Ordenamento causal

A importância desta propriedade de ordenação é ilustrada na figura 4. Neste exemplo, tem-se uma conversa entre quatro usuários, estes pertencentes ao mesmo grupo, sendo cada usuário um membro do grupo. Na parte superior da figura, observa-se um caso onde se emprega o uso de algum algoritmo de ordenação, a qual permite que as mensagens cheguem de forma consistente mantendo a seqüência, assim garantindo a integridade da conversa como um todo, e não somente das mensagens individualmente. Na segunda parte da figura tem-se o possível problema que pode ocorrer caso não utilize-se ordenação. Como as características físicas entre cada usuário diferem (desde distância, velocidade de processamento, meios de transmissão e etc) e o servidor de mensagens trabalhar de forma assíncrona, é perfeitamente compreensível que uma mensagem enviada pelo usuário A chegue no usuário B após a mensagem do usuário C, mesmo se a mensagem do usuário A ter sido encaminhada para envio com um tempo significativamente anterior ao usuário C. Neste caso as mensagens não seguiram uma ordem cronológica, deixando a conversa completamente confusa.

Um mecanismo de comunicação em grupo necessita de uma semântica bem definida em relação à ordem em que as mensagens serão entregues, sendo a melhor garantia fazer com que as mensagens sejam entregues na mesma seqüência em que foram enviadas.

Os mecanismos de ordenação mais difundidos e implementados são o sem ordem, ordenamento FIFO (*First In, First Out*), ordenamento causal e ordenamento total. Abaixo, tem-se uma breve explicação do funcionamento dos mesmos:

a) sem ordem: neste tipo não há qualquer controle quanto à ordenação. Possui o menor *overhead* em comparação a outros tipos, por dispensar várias informações. Como não possui um controle que mantenha a seqüência, sua utilização limita-se a aplicações em que somente requerem transmitir mensagens, não possuindo ligação com outras mensagens anteriores ou posteriores, para formar uma informação íntegra, dispensando uma seqüência cronológica;

b) ordenamento FIFO: mantém a mesma seqüência de mensagens enviadas por um membro, para os demais membros do grupo. Por exemplo, caso um participante A envia as seguintes mensagens respectivamente: "um", "dois" e "três", os outros membros do grupo receberam a mesma seqüência, inicialmente a mensagem "um", e por último a mensagem "três". Jamais ocorrerá de um membro receber as mensagens em outra ordem, ou tão pouco receber uma mensagem sem as que antecedem a mesma;

c) ordenamento causal: assim como na ordenação FIFO, a ordenação causal garante a mesma ordem de recebimento, conforme a ordem em que foram enviadas. O que diferenciam elas é o fato de o causal preocupar-se com a seqüência do grupo como um todo, ao contrário do modo FIFO que garante a seqüência das mensagens individualizando cada membro. Exemplo, caso o membro A envie a mensagem "um", depois o membro B envia a mensagem "dois" e o membro A envia novamente uma outra mensagem "três" após a mensagem do membro B. Nesta situação, um participante C receberia respectivamente, "um", "dois" e "três", e de forma nenhuma outra seqüência, mesmo em caso da mensagem "três" tenha sido recebida anterior a "dois". No caso FIFO a única garantia seria quanto a chegada da mensagem "três" posterior a "um", não necessariamente respectivos;

d) ordenamento total: o ordenamento total certifica-se de que todos os membros do grupo recebam as mensagens na mesma ordem, entretanto não a qualquer ligação entre a seqüência de envio e a seqüência de recebimento. Supondo que a seguinte seqüência de mensagens sejam enviadas: "um", "dois", "três". Caso o usuário A receba anteriormente aos outros usuários, sendo respectivamente "três", "dois" e "um", isto significa que esta mesma seqüência será mantida para os outros

membros. Este método de ordenamento é o melhor dos quatro aqui apresentados, além de tornar a programação fácil da aplicação que usa o método de ordenação, entretanto é o mais difícil de ser implementado.

Para um entendimento mais fácil, em seguida tem-se uma série de figuras que ilustram o funcionamento destes quatro modos de ordenação.

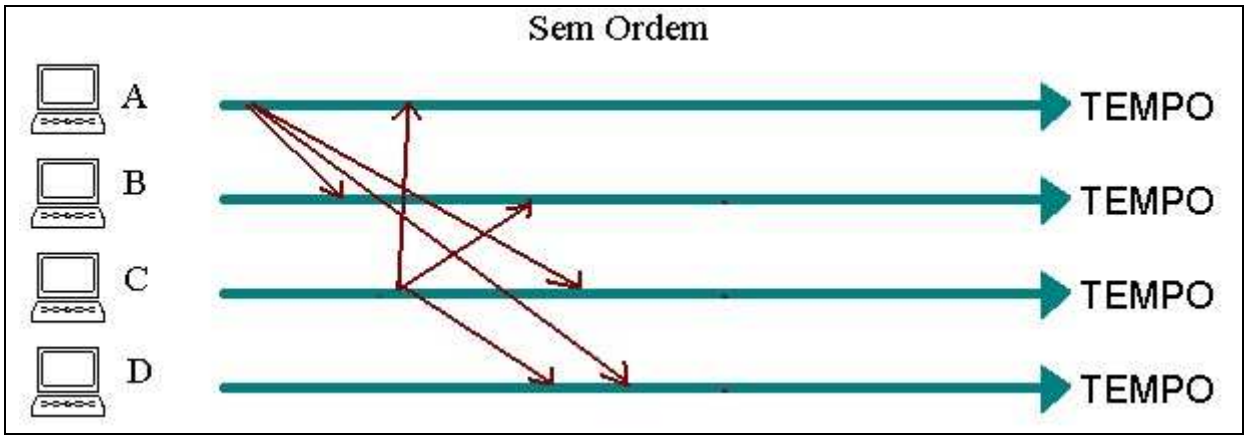


Figura 5 – Sem ordem

A figura 5 descreve o mecanismo de ordenação conhecido como sem ordem, pode-se observar que não existe qualquer regra para a ordem do recebimento das mensagens.

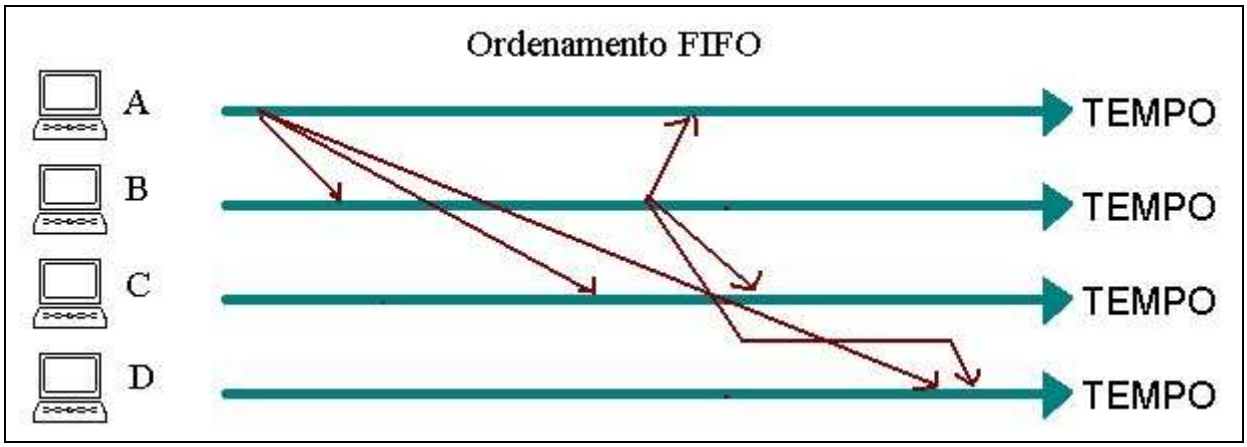


Figura 6 – Ordenamento FIFO

Na figura 6 pode-se observar o ordenamento do tipo FIFO. Nesta figura a mensagem enviada pelo usuário B ao usuário D sofre um atraso como o intuito de que a mensagem do usuário A para o usuário D chegue anteriormente, desta forma respeitando a ordem.

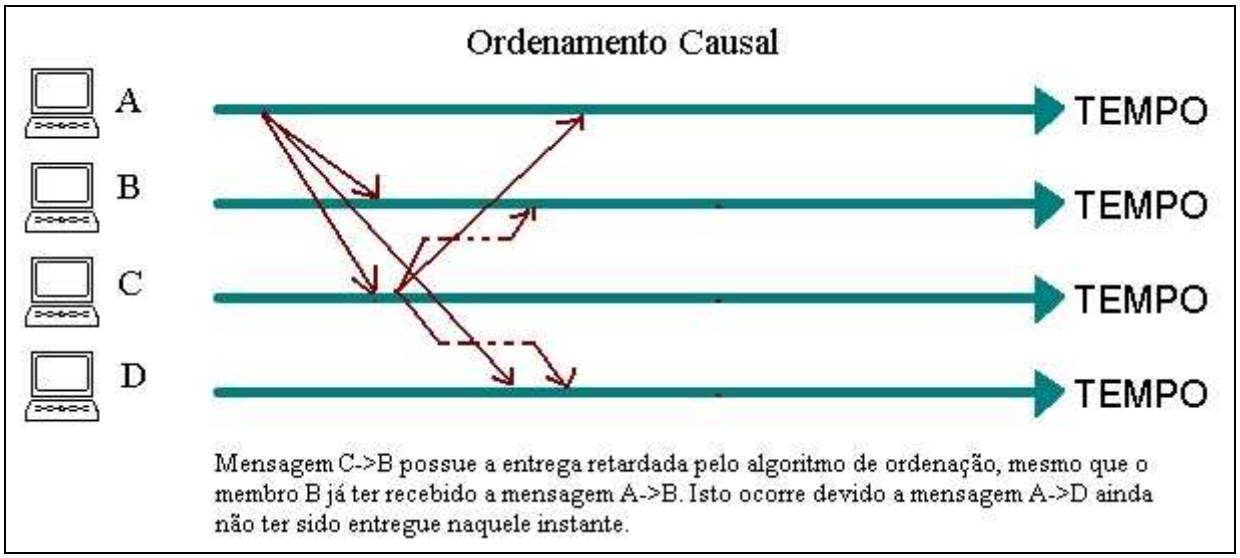


Figura 7 – Ordenamento causal

O ordenamento do tipo causal pode ser observado na figura 7, onde são atrasadas as mensagens do usuário C destinadas aos usuários B e D.

Representado na figura 8, o ordenamento total apresenta suas características ao atrasar as mensagens do usuário A para D e do usuário D para A e C.

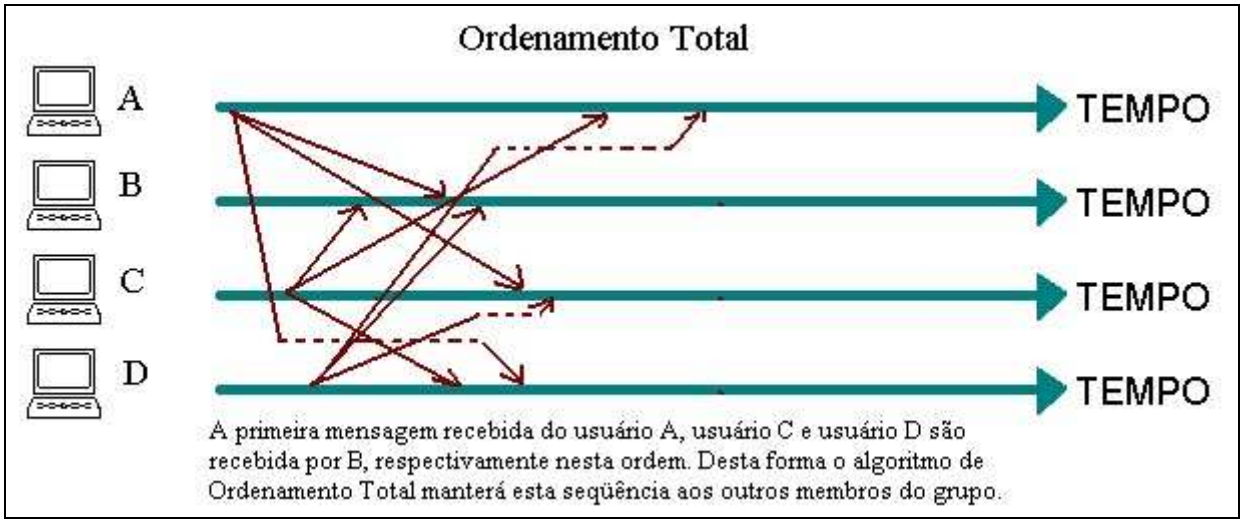
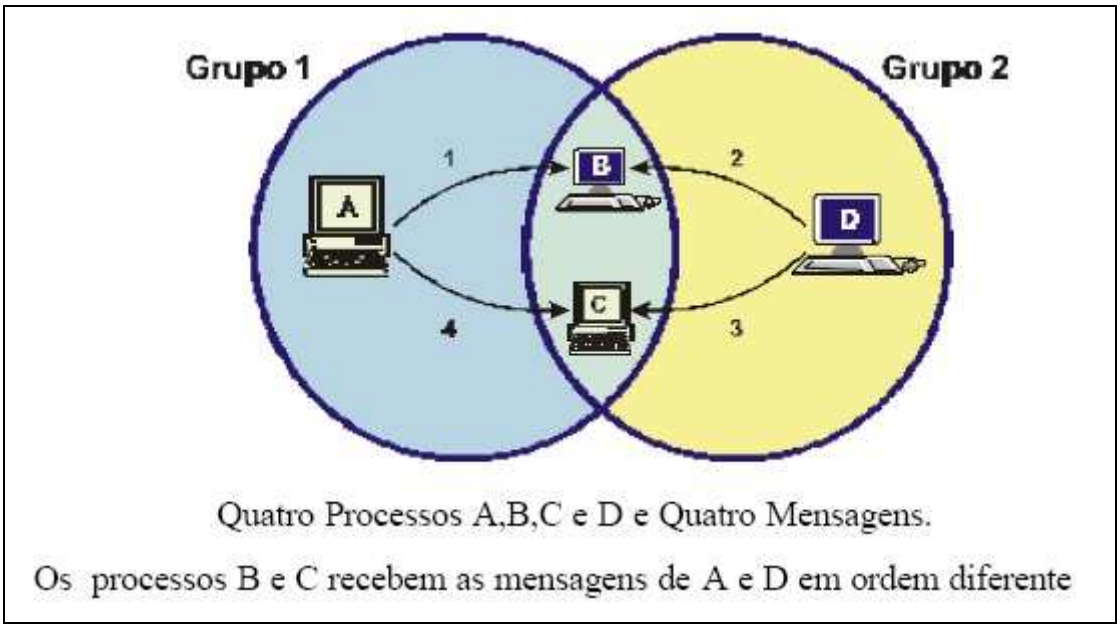


Figura 8 – Ordenamento total

Outra propriedade muito utilizada corresponde à filiação de um processo a dois ou mais grupos simultaneamente, esta propriedade nomeia-se sobreposição. A sobreposição pode levar a algumas inconsistências, isto ocorre por não haver uma coordenação entre os diversos grupos. Os grupos podem conter ordenações internamente, mas não há um controle geral que trate estas diferenças entre cada grupo.

Alguns sistemas que suportam ordenação no tempo global bem definidas para grupos sobrepostos, entretanto outros não, sendo estes últimos certamente atingidos por este empecilho.

Na figura 9 tem-se uma demonstração da sobreposição. Neste exemplo, tem-se dois grupos distintos, grupo I e grupo II, e quatro participantes de grupos, A, B, C e D. Integram o grupo I os membros A, B e C, no grupo II estão os membros B, C e D. Como observa-se, os usuários B e C encontram-se tanto no grupo I como no grupo II, assim ocasionando a sobreposição.



Fonte: Goulart (2002, p. 75)

Figura 9 – Sobreposição de grupos

Supondo que os membros A e D enviem simultaneamente mensagens aos seus grupos, para o grupo I a mensagem chegue antes ao usuário B e posteriormente ao C, enquanto que no grupo II a mensagem é recebida por C e D, respectivamente. Neste caso o usuário C receberá as mensagens de forma contraria ao usuário B, gerando assim uma desordem na seqüência.

Uma última propriedade que deve ser citada é a escalabilidade, esta propriedade refere-se à quantidade de membros suportados pelo mecanismo.

Muitos algoritmos funcionam bem enquanto os grupos possuem poucos elementos e não houver muitos grupos, entretanto podem ter problemas se um grupo tiver centenas ou milhares de componentes, ou se tivermos milhares de grupos e se o sistema for muito grande.

2.2 TRABALHOS CORRELATOS

Nesta seção será exposta uma visão superficial dos principais aplicativos existentes que apresentam alguma relação com o protótipo em foco neste trabalho.

As análises destas ferramentas estão divididas em duas seções, uma referente a aplicativos que realizam de forma similar o objetivo deste trabalho proposto, a comunicação, e uma segunda seção relacionada a aplicativos que empregam o mesmo mecanismo de comunicação em grupo utilizado no protótipo.

2.2.1 APLICATIVOS DE MENSAGEM INSTANTÂNEAS

Os aplicativos mais populares para troca de mensagens instantâneas são o MSN Messenger, o ICQ e o Yahoo Messenger. Adiante encontra-se um resumo das principais características destes aplicativos.

2.2.1.1 ICQ

O aplicativo ICQ (abreviatura fonética para "*I Seek You*") foi provavelmente o primeiro software do tipo. Se não o primeiro a ser desenvolvido, certamente o primeiro a tornar-se popular atingindo números gigantescos de usuários.

Conforme ICQ (2004), em novembro de 1996, apenas quatro meses após o estabelecimento da empresa israelense Mirabilis, a primeira versão do ICQ encontrava-se disponível na Internet. Em um curto período de tempo a quantidade de usuários cresceu em proporções monstruosas. Em junho de 1998 a Mirabilis foi adquirida pela norte-americana America Online.

Os principais recursos disponíveis no ICQ são:

a) envio de mensagens a outros usuários;

b) envio de arquivos a outros usuários por conexão P2P;

c) *chat* em tempo real entre dois ou mais usuários;

d) busca de usuários.

O ICQ oferece recursos extras que podem ser baixados na forma de *plug-ins*, como uso de voz e *webcam*, além de permitir uma série de configurações como: lista de usuários ignorados, lista de usuários sem visibilidade de *status*, lista de usuários com visibilidade sempre. A conexão no ICQ usa o paradigma cliente-servidor, entretanto, caso haja uma possibilidade de conexão P2P, esta arquitetura é utilizada entre dois usuários após o devido estabelecimento da localização de ambos através do servidor central do sistema.

Figura 10 – Lista de contatos do ICQ

A versão do ICQ aqui analisado corresponde ao ICQ Pro 2003b Build #3916. A figura 10 apresenta a interface da lista de contatos do ICQ.

2.2.1.2 MSN MESSENGER

O MSN Messenger (MICROSOFT CORPORATION, 2004) é desenvolvido pela Microsoft desde 1997 e encontra-se integrado as versões mais recentes do sistema operacional Windows. Em parte devido ao fato de encontrar-se já integrado ao Windows e também por

apresentar uma interface bem atraente, cheio de imagens, cores, provavelmente seja o software de mensagens instantâneas mais utilizado atualmente.

Os principais recursos disponíveis pelo MSN Messenger são:

a) envio de mensagens a outros usuários;
b) envio de arquivos a outros usuários por conexão;
c) *chat* em tempo real entre dois ou mais usuários;
d) uso de voz;
e) uso de *webcam*.

Figura 11 – Lista de contatos do MSN Messenger

A figura 11 ilustra a lista de contatos do MSN Messenger. Foi utilizada a versão 6.2 (6.2.0137) nesta análise.

2.2.1.3 YAHOO MESSENGER

O Yahoo Messenger (YAHOO, 2004) não encontra-se no mesmo nível de popularidade dos seus concorrentes da América Online e Microsoft. A versão do software aqui analisada é a 6,0,0,7050. Os principais recursos deste software são:

a) envio de mensagens a outros usuários;
b) envio de arquivos a outros usuários por conexão;
c) *chat* em tempo real entre dois ou mais usuários;
d) uso de voz;
e) uso de *webcam*;
f) radio.

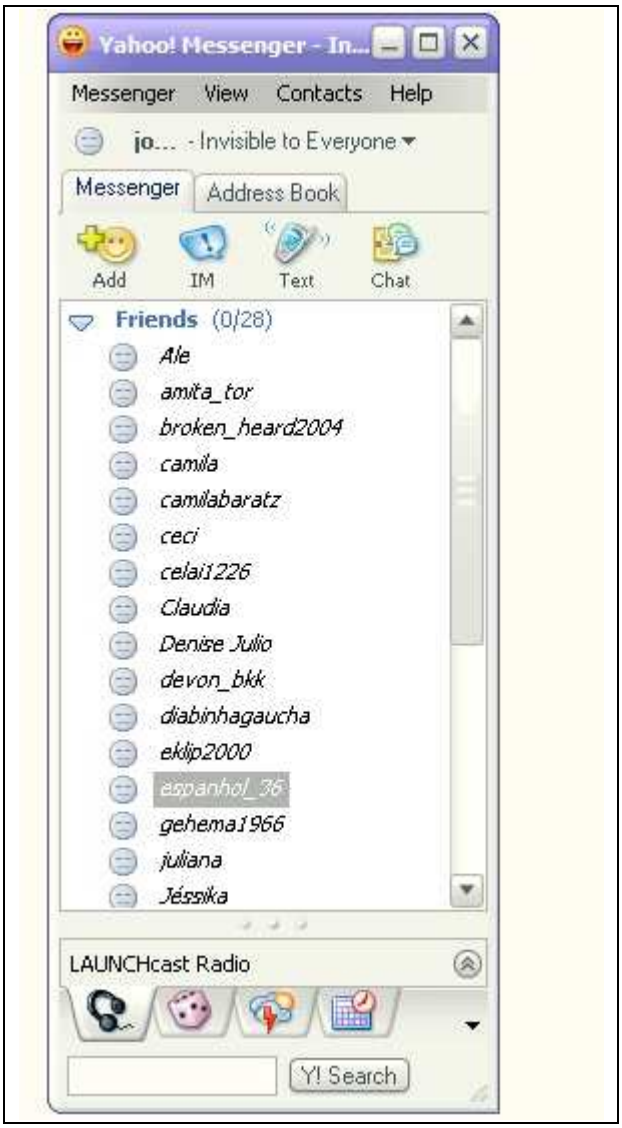


Figura 12 – Interface principal do Yahoo Messenger

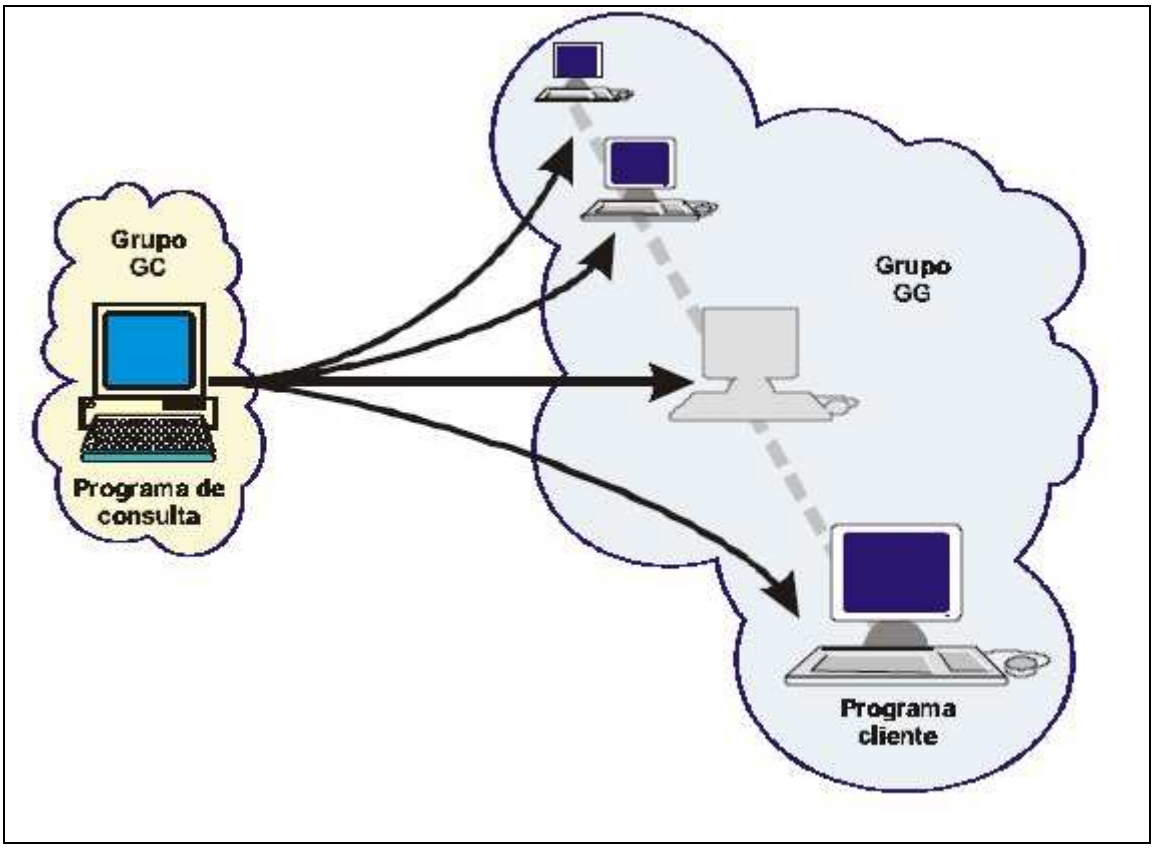
A figura 12 apresenta a interface principal do Yahoo Messenger.

2.2.2 APLICATIVOS QUE UTILIZAM SPREAD

Apesar dos mecanismos de comunicação em grupo ainda serem desconhecidas por grande parte das pessoas ligadas a informática, alguns projetos vêem sendo desenvolvidos aplicando estas ferramentas. Nesta seção apresenta-se alguns destes projetos que utilizam as facilidades de um mecanismo de comunicação em grupo através do Spread.

2.2.2.1 COMUNICADOR

Um pequeno aplicativo nomeado de Comunicador foi desenvolvido por Alex Kuhnen com o objetivo de gerenciar recursos de ambientes distribuídos onde um módulo de consulta interage com N clientes membros de um grupo, recebendo informações de gerência de recursos que estão disponíveis nestes clientes.



Fonte: Goulart (2002, p. 135)

Figura 13 – Módulo mestre (consulta) e escravos (clientes)

A segunda função básica do aplicativo Comunicador é o gerenciamento em tempo real interagindo com diversos computadores clientes. Assim, esta aplicação é composta de dois módulos distintos, um na forma de mestre que gerenciará vários escravos. Nesta aplicação o módulo escravo é tratado como um conjunto de escravos que fazem parte do mesmo grupo. Assim cada computador que estiver sendo gerenciado vai executar uma cópia deste módulo cliente, fazendo parte do grupo. O módulo mestre, de consulta, enviará mensagens aos módulos clientes que identificarão o tipo de consulta e fornecerão a resposta adequada. A figura 13 demonstra a situação recém descrita.

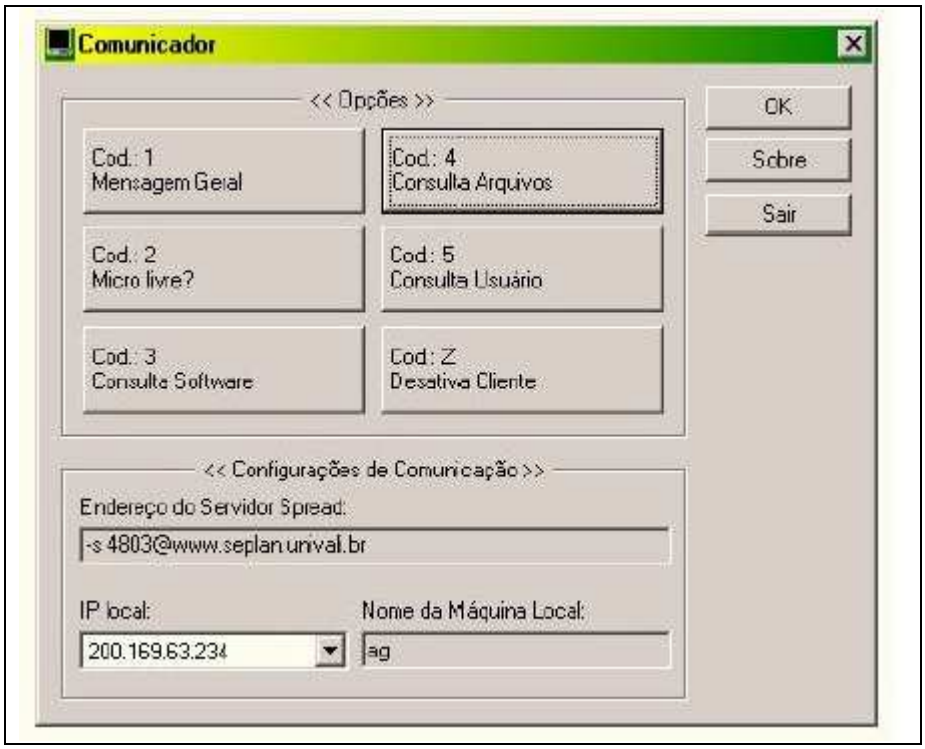
O módulo cliente foi desenvolvido para plataforma Windows e atua como um *daemon*. Assim que é iniciado, conecta-se através do Spread a um grupo GG utilizando como identificador único o nome o computador. Este módulo permanece sempre filiado ao grupo GG aguardando mensagens, é somente finalizado por uma requisição de término por parte do módulo mestre. Este módulo tem as seguintes funções:

a) exibição de mensagem em janela tipo *pup-up*;
b) informar se um computador esta sem usuários ativos;
c) verificar arquivos com extensão XYZ respondendo a quantidade de arquivos e o total de memória em k bytes ocupados por estes arquivos;
d) verifica se determinado usuário esta usando o computador, caso positivo, responde passando a identificação da máquina correspondente;
e) finaliza um módulo cliente.

O módulo de consulta, assim como o módulo cliente, também foi desenvolvido para operar na plataforma Windows. Este módulo tem como objetivo interagir com os módulos clientes que integram o grupo GG. Na figura 14 tem-se a interface principal deste módulo.

As funções desempenhadas por este módulo de consulta são:

a) envio de mensagem em broadcast;
b) consulta de computador não “logado”;
c) consulta quais clientes estão executando determinado software;
d) consulta se as máquinas com o módulo cliente possuem arquivos do tipo XYZ;
e) busca se determinado usuário esta “logado” em alguma máquina;
f) desliga um módulo cliente.



Fonte: Goulart (2002, p. 137)

Figura 14 – Módulo de consulta

Pode-se observar na figura 14 o módulo de consulta sendo executado no computador “ag” localizado no endereço IP “200.169.63.234”, este módulo encontra-se conectado ao servidor Spread executado no endereço “www.seplan.univali.br” através da porta 4803.

2.2.2.2 SISTEMA DISTRIBUIDO DE ARMAZENAMENTO OASIS+

De acordo com Fleisch (2004), Oasis+ é um sistema distribuído de armazenamento confiável e para uma pequena escala de *clusters*. Um dos principais benefícios do sistema é permitir o nível de confiabilidade ser configurado de acordo com as necessidades da aplicação. Esta propriedade permite que aplicativos que requerem um alto grau de tolerância a falhas operem sem sobrecarregar aquelas aplicações em que deseja-se sacrificar a tolerância a falhas em nome da velocidade.

A extensa quantidade de configurações providas pelo Oasis+ é diretamente resultado da implementação do protocolo de coerência BR, um protocolo de alta performance também desenvolvido pelo grupo Mirage. Por permitir o nível de confiabilidade ser configurado, Oasis+ pode satisfazer os requerimentos de uma abrangente variedade de aplicativos. Mais

detalhes sobre este projeto podem ser encontrados em "*http://www.cs.ucr.edu/~brett/pubs.html*".

2.2.2.3 WACKAMOLE

Segundo Johns Hopkins University (2003), Wackamole é uma aplicação que ajuda a tornar um *cluster* altamente disponível. Wackamole manipula um ramo de IP que deve ser disponibilizado para o mundo em tempo integral. É assegurado que dentro do *cluster* uma máquina simples está ouvindo cada endereço IP virtual que Wackamole gerencia. Caso por alguma circunstância uma das máquinas do *cluster* em particular tornar-se inoperante, quase imediatamente será assegurado que uma outra máquina adquira estes IPs públicos. Em nenhum momento mais que uma máquina estará ouvindo um IP virtual.Wackamole também funciona buscando alcançar uma distribuição balanceada do número de IP na máquina do *cluster* que é gerenciada.

Este aplicativo trabalha como uma árvore em um *cluster* para torná-lo altamente disponível. É usada a notificação entre os membros através do mecanismo de comunicação em grupo Spread para gerar um estado consistente que é consentido entre as instâncias Wackamole conectadas. Este conhecimento é usado para assegurar que todos os IP servidos pelo *cluster* sejam cobertos por exatamente uma instância do Wackamole.

Wackamole é desenvolvido pelo centro de redes e sistemas distribuídos na Johns Hopkins University e encontra-se mais detalhado no *site* "*http://www.backhand.org/wackamole*".

2.3 O MECANISMO DE COMUNICAÇÃO EM GRUPO SPREAD

Nesta seção serão apresentados os principais conceitos e funcionalidades da principal tecnologia utilizada no desenvolvimento do protótipo.

De acordo com THE SPREAD GROUP COMMUNICATION TOOLKIT (2004), o Spread é uma ferramenta que provê alta performance para serviços de mensagens, sendo flexível a problemas, tanto a redes externas, como em *Intranet*. Os serviços prestados pelo mecanismo Spread vão desde mensagens confiáveis, passando por ordenação de mensagens com garantia de entrega, até em casos de falhas em computadores ou particionamento de redes. O Spread é designado para encapsular todos os cuidados necessários sobre a

comunicação em redes assíncronas, permitindo assim que o projetista foque em outros componentes de suas aplicações.

Este mecanismo foi criado por Yair Amir, Michal Miskin-Amir e Jonathan Stanton, atualmente é desenvolvido por *Spread Concepts LLC e Center for Networking and Distributed Systems* (CNDS) na universidade *Johns Hopkins University*.

Segundo Stanton (2002), quando se deseja desenvolver alguma aplicação distribuída, devem-se escolher vários tipos de arquiteturas. Estas escolhas incluem, como será suportada a comunicação entre cada aplicação, quais as funções de cada processo, o quanto dependente cada máquina é das outras para operações na aplicação. Parte do que dificulta ser confiável, com alta performance, útil para aplicações distribuídas, é o número destas escolhas fundamentais e a complexidade entre cada escolha. O modelo de comunicação em grupo é um esqueleto que provê juntamente, uma ferramenta física para desenvolver e um modelo que limita este número de escolhas que devem ser encontradas. Este modelo simplifica as tarefas de construção de confiabilidade, corrige aplicações distribuídas enquanto ainda fornecem ao usuário um conjunto poderoso de abstrações sob quais muitas aplicações distribuídas diferentes podem ser desenvolvidas.

Certamente não é qualquer aplicação que pode ser desenvolvida usando um modelo de comunicação em grupo, mesmo caso pudesse, as características negativas do modelo podem fazer a comunicação em grupo ser uma péssima alternativa. O que a comunicação em grupo propõe fazer uma grande parte de aplicações distribuídas serem mais fáceis de se desenvolver e mais poderosas. O Spread não é diferente de qualquer outro alto nível de abstração. Por exemplo, ninguém poderia, para cada aplicação de rede, iniciar desenvolvendo desde o nível da criação de IPs, encapsulamentos, *checksums* de pacotes, multiplexadores, segurança, ordenação e fluxo de controle, mas sim entende-se que apesar de isto ser o mais otimizado para a aplicação (e isto é utilizado por algumas aplicações especializadas), em quase todos os casos deseja-se usar uma API de alto nível como *sockets* e um difundido conjunto de protocolos de rede como o TCP/IP.

Pode-se, observar algumas desvantagens para comunicação em grupo de alta performance quando usada em redes de longa distância, em relação a redes locais. Algumas destas desvantagens são:

a) grande variação em diferentes partes da rede quanto às características (baixas taxas, quantidade de *buffer*) e performance (latência e largura de banda);
b) maiores quantidades de perdas de pacotes e tempo de latência;
c) difícil implementar de forma eficiente, a ordenação e a confiabilidade sobre os mecanismos de *multicast*. Principalmente porque o mecanismo disponível de *multicast* por melhor esforço para redes de grande distância vem com limitações significantes.

Levando-se em consideração estas dificuldades, observa-se que houve um bom crescimento no desenvolvimento da comunicação em grupo quanto a redes locais, onde os problemas anteriormente citados são largamente amenizados, entretanto o mesmo não ocorreu quanto às redes de longa distância. Atualmente estes problemas vem sendo desviados através do uso do IP *multicast* melhor esforço, construindo serviços confiáveis e uma ordenação com semântica mais fraca do que a aplicada nos mecanismos de comunicação em grupo.

O Spread tem o ponto de vista da comunidade de comunicação em grupo. Implementa técnicas e facilidades de projeto que não são diferentes das técnicas usadas para prover confiabilidade sobre *IP-Multicast* e leva disseminação e controle de fluxo para redes de banda larga. Assim um sistema de comunicação em grupo tem extensivo e detalhado conhecimento do sistema, os protocolos usados podem ser mais precisos e produzirem melhor performance do que muitos protocolos de redes genéricos.

Todas estas vantagens apresentadas quanto ao Spread possuem um custo, incidindo na escalabilidade. O Spread faz mais trabalho por nó e não pode crescer demasiado em número de usuários. Todavia, pode ser disponibilizado em WANs e o número de grupos pode aumentar sem problemas, tendo em vista que não são mantidas tabelas de estados nos roteadores referentes a cada grupo.

O Spread possui três características principais que, comprometem-se a minimizar as dificuldades anteriormente citadas quanto a redes de grande abrangência, elas são:

a) permite uso de diferentes protocolos de baixo nível para prover disseminação confiável de mensagem. Cada protocolo conte diferentes parâmetros, podendo assim, serem aplicados em diferentes partes da rede, de acordo com as peculiaridades de cada área, contudo tornando o processo mais otimizado. São integrados ao Spread dois protocolos de baixo nível, um chamado de Ring

destinado a redes locais, e outro chamado de Hop para uma melhor adaptação em redes de longa distância;

b) uso da arquitetura cliente/servidor (*client-daemon*). Esta arquitetura possui como principal vantagem um custo mínimo para alterações de membros do grupo. Caso um membro associe-se, ou deixe de fazer parte de um grupo, esta alteração é realizada com apenas uma mensagem. Para um processo se conectar ou desconectar do servidor o custo é super baixo, sem a necessidade de trocas de tabelas entre os roteadores de uma rede WAN, havendo esta necessidade de troca somente entre as partes que fazem parte da rede local;

c) o mecanismo de disseminação e segurança local é separado do protocolo global de ordenação e estabilidade. Assim, permitindo que mensagens possam trafegar na rede de forma direta sem o uso de controles de perde e ordenação. Esta separação também permite que mensagens sejam enviadas a somente alguns componentes da rede, sem causar qualquer distúrbio ao funcionamento da comunicação em grupo.

Outras características positivas do Spread incluem:

a) eficiência e confiabilidade do serviço de entrega;

b) confiança para *multicast* de qualquer número de remetentes para grande quantidade de destinatários;

c) serviço de grupo com boa escalabilidade, permitindo milhares de grupos ativos simultaneamente;

d) serviço de comunidade na qual informa a cada componente da aplicação sobre outro componente esta em funcionamento, habilitado serviço de recuperação em caso de falhas;

e) serviços de mensagens para todo o grupo, onde todos os destinatários recebem mensagens enviadas a todo o grupo exatamente na mesma ordem, o que facilita muito para aplicações que necessitam manter uma ordem para razões de segurança ou usabilidade;

f) ordenamento tipo FIFO, que possui grande utilidade para transferências de *streaming* multimídia, onde a seqüência de um vídeo ou som é indispensável;

g) facilidades de controle sobre mecanismo de comunicação e o *layout* da rede virtual;

h) suporta canais prioritários, que mantém garantias de ordenação requeridas e acelera o envio da mensagem;

i) suporta semântica de grupo aberto;
j) multiplataforma, tendo versões que podem ser aplicadas aos ambientes de programação C#, PHP, Python, Lua, Squeak Smalltalk, Ruby, Java e Objective Caml. Por Java ser suportado em quase todos os sistemas operacionais, o Spread possui uma enorme portabilidade.

Segundo Stanton (2002), existe uma grande variedade de aplicativos distribuídos que empregam a comunicação em grupo, portanto o Spread, com a finalidade de ampliar o grau de abstração destes sistemas. Na seqüência encontram-se alguns exemplos destas aplicações:

a) serviços e máquinas de monitoramento: uma quantidade de máquinas exportando seus status para grupos de monitores interessados. Ocorrendo qualquer falha, os monitores recebem uma notificação;
b) ferramentas colaboradoras: muitos diferentes grupos de participantes que desejam compartilhar dados, vídeos e conferências de áudio;
c) DSM (*Distributed Shared Memory*): enviando páginas de memória para máquinas onde é necessário usar *multicast* confiáveis;
d) serviços altamente confiáveis (como sistema de tráfego aéreo, troca de estoques, rastreamento militar e sistemas de controle de combate): Serviços que envolvem comunicação de informações acumuladas em numerosas máquinas e pessoas, e possuem alto requerimento tanto para disponibilidade quando para tolerâncias à falhas;
e) replicação de bancos de dados: um número de instâncias de um banco de dados existe em diversas localidades. Todos estes bancos de dados devem ser mantidos sincronizados ao ponto que um cliente possa pesquisar ou atualizar qualquer um deles e o resultado será o mesmo, passando ao cliente a idéia de que existe somente um banco de dados.

2.3.1 ARQUITETURA DO SPREAD

De acordo com Amir e Stanton (1998), o Spread funciona com base na arquitetura *daemon-client*. Os *daemons* atuam como servidores centrais do mecanismo, possibilitando a formação de uma rede básica de disseminação de mensagens além dos controles de ordenação dos grupos e manutenção dos membros existentes. Já a parte *client* reside em cada aplicação que utilizará os recursos do mecanismo Spread. O módulo *client* que estará incorporado ao

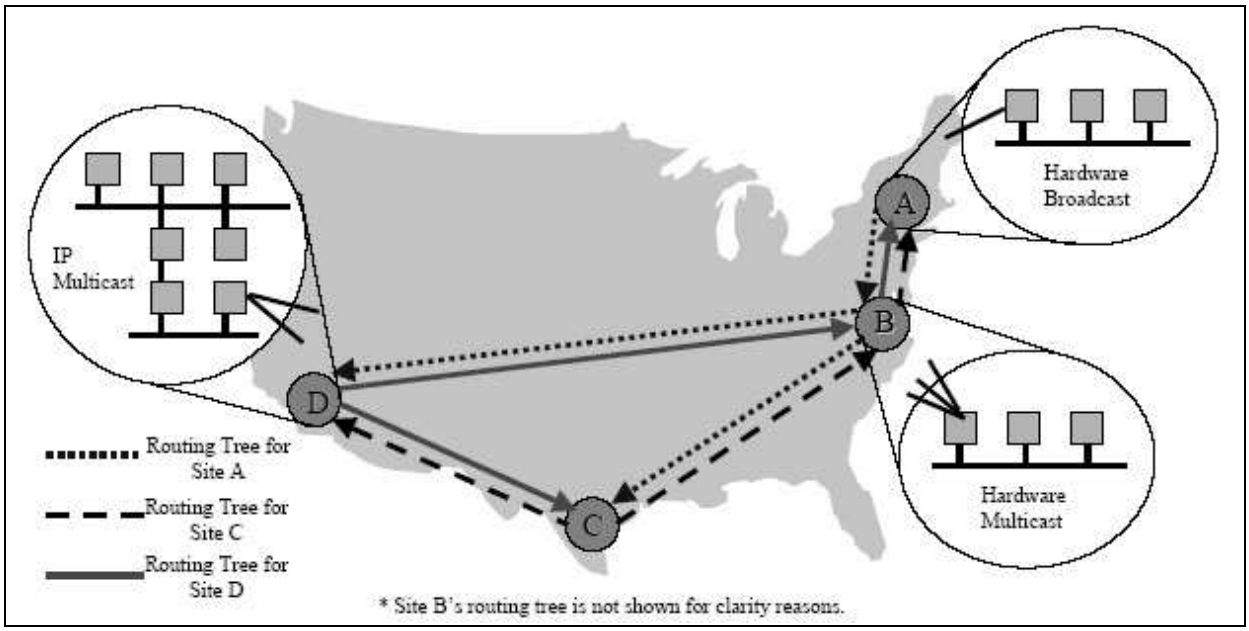
aplicativo terá a função de criar uma ponte de comunicação entre o aplicativo e o mecanismo, para isto, uma conexão entre o *client* e o módulo *deamon* mais próximo na rede será realizada. Vale ressaltar que um aplicativo pode conter integrado tanto o módulo *client*, como o *daemon*, assim podendo disponibilizar os serviços do Spread, bem como, utilizar-se destes recursos.

Segundo Goulart (2002), o uso da arquitetura *daemon-client* encontrada no Spread é muito importante em uma rede de grande distância por que os *daemons* minimizam o número de trocas no controle dos membros que o sistema tem que fazer através dos pontos de interligação da rede de grande distância, devido a conter os serviços de ordenação e controle dos membros do grupo atendidos de forma centralizada pelo *deamon*, e não de forma individualizada por cliente. Além disso, fornece uma infra-estrutura básica e estável na qual se pode construir um eficiente controle de roteamento e ordenação para redes de grande distância.

Há um certo inconveniente neste tipo de arquitetura *daemon-client*, pois pode ocorrer interferência entre dois diferentes clientes que usam a mesma configuração do *daemon*. Uma solução para amenizado esta dificuldade é executar múltiplas configurações dos *daemon*, cada uma servindo diferentes aplicações, ou ainda também, pelo modelo de disseminação usado pelo Spread, que somente manda mensagens de dados para aqueles *daemons* que necessitam das mensagens, minimizando o custo de atividades extras nos *daemons* que a aplicação não está usando, infelizmente com custos adicionais quanto à comunicação interprocesso e chaveamento de contexto.

Quanto a configuração o Spread mostra-se bastante flexível, contendo abundante quantidade de configurações, o que permite uma melhor otimização do mecanismo de acordo com as necessidades encontradas em cada caso. Pode-se utilizar apenas um módulo *daemon* para gerenciar todo o sistema, ou ainda individualmente cada computador, permitindo assim a execução de aplicações de comunicação em grupo internamente.

Contudo, a melhor performance é alcançada utilizando-se um módulo *daemon* em cada computador para casos em que a rede comporte-se estável. Entretanto, para redes com falhas, um layout com alguns poucos *daemons* talvez seja mais indicado, devido a um custo maior de recuperação de quando utilizado *daemons* em todos os computadores.



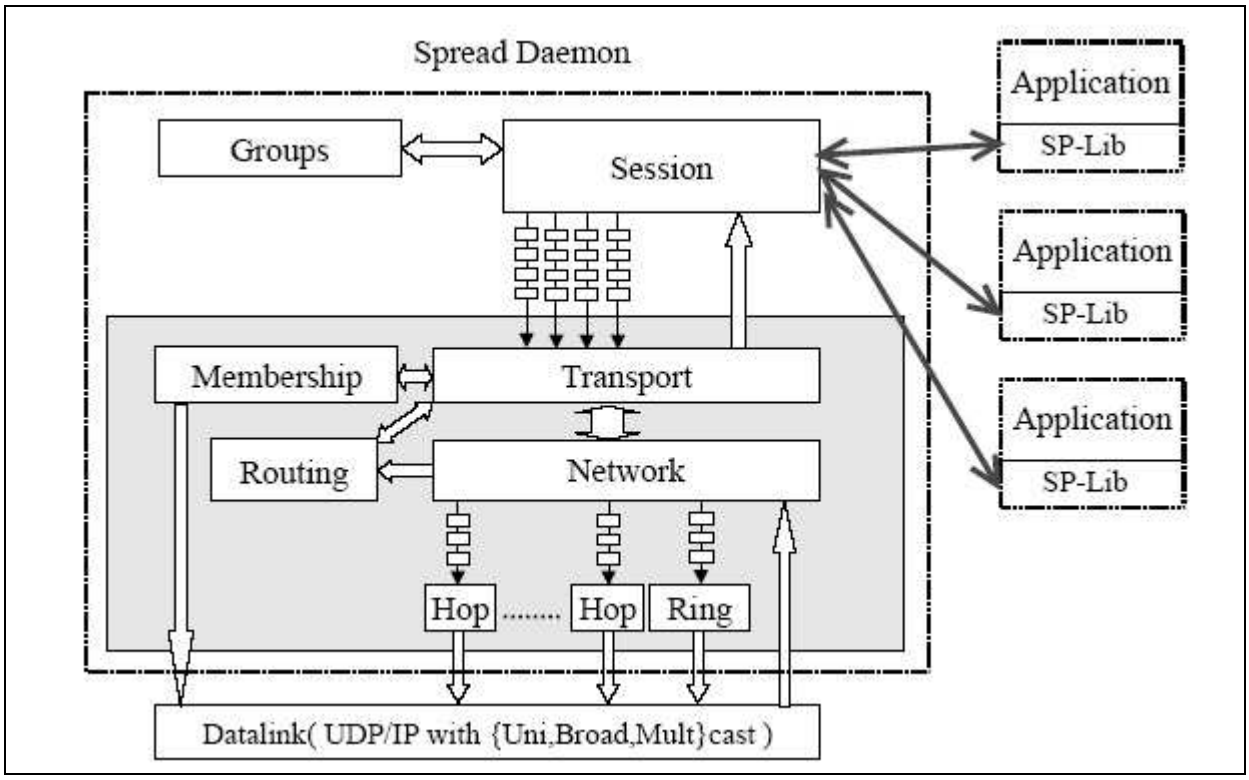
Fonte: Amir e Stanton (1998, p. 4)

Figura 15 – Rede de grande abrangência configurada com Spread.

Na figura 15 tem-se um exemplo de rede. Pode-se, observar alguns sites dispersos geograficamente com diferentes custos para cada link entre eles. Como site pode-se entender como uma coleção de computadores com potencial que atingir outras máquinas por uma mensagem, por exemplo, *broadcast* por hardware, *multicast* por hardware, ou *IP-Multicast*. Cada site pode ter uma infinidade de máquinas, tendo em vista que não ocasiona em qualquer distúrbio para a escalabilidade do mecanismo Spread, sendo que cada site representará um único membro, permanecendo as demais máquinas locais para cada site, contudo não totalizando o número de máquinas envolvidas. Cada *daemon* participante em uma configuração Spread tem conhecimento do potencial completo de cada membro quando iniciado, entretanto estas informações podem sofrer alterações dinamicamente em tempo de execução, desta forma, mantendo o mecanismo trabalhando com base numa análise recente, portanto consistente com a realidade. Cada site tem um *daemon* que atua como representante do site, participando da disseminação na rede de longa distância. Este representante é determinado baseado nos membros participantes no momento e não é uma configuração estática de hardware.

Na figura 16 é apresenta a arquitetura do Spread. O aplicativo se relaciona com a biblioteca *SP_lib* (ou uma classe, como é o caso do Spread para Java), a qual oferece uma completa interface ao cliente. A conexão entre a biblioteca *SP_lib* e o módulo *daemon*, trata-

se de uma conexão segura ponto-a-ponto, independente de IPC ou através da rede. A seção e os módulos gerenciam as conexões dos usuários, gerenciam a participação de processos no grupo, e traduzem as mudanças de membros do grupo no *daemon* em mudanças de membros do grupo no processo grupo. A parte sombreada na figura 16 ilustra os protocolos internos encontrados no módulo *daemon*.



Fonte: Amir e Stanton (1998, p. 5)

Figura 16 – Arquitetura Spread

Alguns pontos relevantes quanto a arquitetura são:

a) existem várias vias de transporte entre a sessão e os módulos de transporte, um para cada sessão, assim permitindo recursos de prioridade;
b) o módulo de roteamento calcula as árvores de roteamento baseada nos dados do modulo de rede. O módulo de transporte consulta o módulo de roteamento para determinar os *links* no qual cada mensagem deve ser enviada;
c) podem existir várias instâncias do módulo Hop, cada uma delas representando uma margem que é usada por uma ou mais árvores de roteamento;
d) no máximo um módulo Ring provê disseminação e confiabilidade no site local, caso mais de um *daemon* estiver ativo neste site;
e) de acordo com as mudanças dos membros do grupo, instâncias de Hop e Ring podem ser criadas ou destruídas.

De acordo com Amir e Stanton (1998), o Spread suporta o modelo de EVS (*Extended Virtual Synchrony*) de membros participantes do grupo. O EVS pode manusear partições da rede e reagrupar bem como agregar ou desagregar, assim provendo diversos tipos de confiabilidade (confiável, não confiável), ordenação (sem ordem, FIFO, causal, ajustado) e serviços de estabilidade (seguro) para mensagens da aplicação.

Segundo Goulart (2002), todos os ordenamentos globais e estabilidade (causal, acordada e segura) são providos através de todos os grupos, caso duas mensagens sejam enviadas por diferentes clientes para diferentes grupos, qualquer um que faça parte de ambos os grupos recebera as duas mensagens na ordem garantida, mesmo pensando que elas serão recebidas em diferentes grupos. O ordenamento FIFO por sua vez, é provido com cada conexão em um *daemon*, atuando como uma fonte FIFO para propósitos de ordenamento. Assim como no ordenamento global de mensagens o ordenamento FIFO é preservado através do grupo. Em redes de longa distância a entrega confiável se torna útil por que em principio não terá uma penalidade de latência comparado com entrega não confiável por que pode ser entregue tão logo seja recebido. Com o mecanismo FIFO a entrega de uma mensagem somente será bloqueada caso alguma mensagem da mesma aplicação conectada enviada anteriormente a esta estiver faltando.

```
SP_connect( char *spread_name, char *private_name, int priority, int group_membership
            mailbox *mbox, char *private_group )

SP_disconnect( mailbox mbox )

SP_join( mailbox mbox, char *group )

SP_leave( mailbox mbox, char *group )

SP_multicast( mailbox mbox, service service_type,
              char *group,
              int16 mess_type, int mess_len, char *mess )

SP_receive( mailbox mbox, service *service_type, char sender[MAX_GROUP_NAME],
            int max_groups, int *num_groups, char groups[][MAX_GROUP_NAME],
            int16 *mess_type, int *endian_mismatch,
            int max_mess_len, char *mess )

SP_error( int error )
```

Fonte: Amir e Stanton (1998, p. 6)

Figura 17 – Interface de programação do Spread

Todo o mecanismo do Spread é abstraído para o programador através de uma interface de programação (API). Esta interface é completa, apesar de conter uma aparência simples, podendo ser aplicada tanto para redes locais, como para redes de longa distância, sem nem mesmo necessitar de qualquer mudança na aplicação, cabendo mudanças somente em relação

à otimização do mecanismo para a rede presente, mesmo assim não se tornando uma exigência para seu funcionamento, contudo mostra-se um modelo claro e bem definido de comunicação em grupo.

Pode-se observar um exemplo destas interfaces na figura 17. Um aplicativo básico pode utilizar-se do mecanismo de grupo necessitando de apenas cinco funções (*SP_connect*, *SP_join*, SP_leave, *SP_multicast*, *SP_receive*), todavia além destas funções básicas existem uma série de outros recursos disponibilizados pela API, os quais permitem o desenvolvimento de aplicações mais robustas e complexas. Algumas destas funções mais avançadas incluem envios e recebimentos espalhados (*scatter-gather*), envio multi grupo, *polling* em uma conexão ou comparação de identificação de grupos.

2.3.2 PROTOCOLOS

O núcleo do sistema Spread é composto pelos serviços de disseminação, confiabilidade, ordenamento e estabilidade das mensagens. Estes serviços estão dispostos em duas camadas distintas de protocolos, a camada de rede e a camada de transporte. Abaixo, tem-se uma explicação destas camadas, sendo a camada de rede subdividida em dois componentes:

a) camada de rede:
   - protocolos *link-level*, que provem confiabilidade e controle de fluxo de pacotes. O mecanismo Spread implementa dois protocolos possibilitando uma melhor otimização do sistema conforme o domínio. Para casos de conexão ponto-a-ponto utiliza-se o protocolo Hop, caso contrário o protocolo Ring, o qual mostra-se mais eficiente para domínios *multicast*,
   - roteamento, que constrói as saídas de Hops e Rings com base no membro corrente conectado no módulo *daemon*, bem como em seus conhecimentos sobre a rede a qual esta sendo trafegado. A rede construída implementa uma diferente árvore de disseminação de mensagens remanescentes em cada site;

b) camada de transporte - esta camada fornece entrega de mensagens, ordenamento, estabilidade e controle do fluxo global. Esta camada opera entre todos os módulos *daemon* existentes no sistema.

A construção destas árvores de roteamento em cada site é importante por várias razões, entre elas, por ficar evidente que uma árvore para cada site é mais eficiente que uma árvore

global compartilhada entre todos os sites. O Spread não foi desenvolvido para um gerenciamento monstruoso em termos de escalabilidade, entretanto comporta muito bem algumas dezenas de sites, podendo ainda cada site conter até algumas dezenas de módulos *daemons*.

Este protocolo possui uma grande vantagem em relação aos muitos outros existentes, isso se deve, ao overhead de construção destas árvores serem amenizados com o passar do tempo no qual os membros permanecerem em cada site, ao contrário de muitos outros protocolos de roteamento *multicast* os quais assumem que a árvore deve ser somente construída em caso de necessidade, desde que mudanças nos membros participantes do grupo sejam muito comuns e então a árvore tenha que ser refeita rapidamente.

Amir e Stanton (1998) alertam que para utilizar uma infra-estrutura de rede tão eficiente quanto possível, necessita-se enviar o máximo de pacotes completos, para isto, deve-se utilizar diferentes tamanhos de pacotes e ter a habilidade de empacotar múltiplas mensagens de usuários ou pacotes de controle em um único pacote de redes, todos os protocolos da camada de link atualmente tratam não pacotes, mas objetos abstratos que podem variar de tamanho desde 12 bytes até 700 bytes. Contudo, cada pacote enviado na rede é preenchido com a quantidade máxima de objetos cabíveis.

2.3.2.1 DISSEMINAÇÃO DE PACOTES E CONFIABILIDADE

O serviço mais básico encontrado em um protocolo de comunicação *multicast* é a disseminação de mensagens de dados para todos os computadores interessados em receber as mensagens. Uma mensagem de nível de aplicação pode variar seu tamanho entre 0kb e 128kb, devido a isto, o mecanismo Spread fragmenta as mensagens que ultrapassarem este tamanho delimitado, assim aumentando o volume de mensagens enviadas. Para mensagens inferiores a 128kb, o mecanismo encarrega-se de organizar de forma que possa agrupar outra mensagem, ou pelos menos um fragmento de outra mensagem ao mesmo pacote da rede básica, sem causar qualquer prejuízo significativo no tempo de latência.

Pelo menos até 1998 estava em estudo uma forma mais eficiente para fazer com que o Spread possa se decidir pela melhor árvore para conectar-se aos sites. Todavia, até que uma melhor solução pudesse ser encontrada, foi utilizado o algoritmo do menor caminho de

Dikstra para a construção de árvores para o membro dinâmico corrente, tendo-se como base o gráfico completo da estrutura da rede e pesos definidos estaticamente.

Uma vez que as árvores de roteamento estão construídas, ocorre o empacotamento das informações. Este empacotamento segue três princípios:

a) sem bloqueio: pacotes são enviados sem levar em consideração possíveis perdas de pacotes requisitados anteriormente;
b) retransmissão rápida: imediatamente os ascendentes do link onde ocorreu perda realizam a retransmissão dos pacotes perdidos;
c) corte: pacotes não são enviados adiante para *links* herdeiros, caso não seja encontrado algum membro interessado neste pacote na árvore em questão.

O protocolo Hop prove o funcionamento dos modos sem bloqueio e a retransmissão rápida, enquanto o modo corte é provido por um código de procura no roteamento, o qual filtra em todos os sites os *links* herdeiros para onde não há membros de grupos com o pacote destinado.

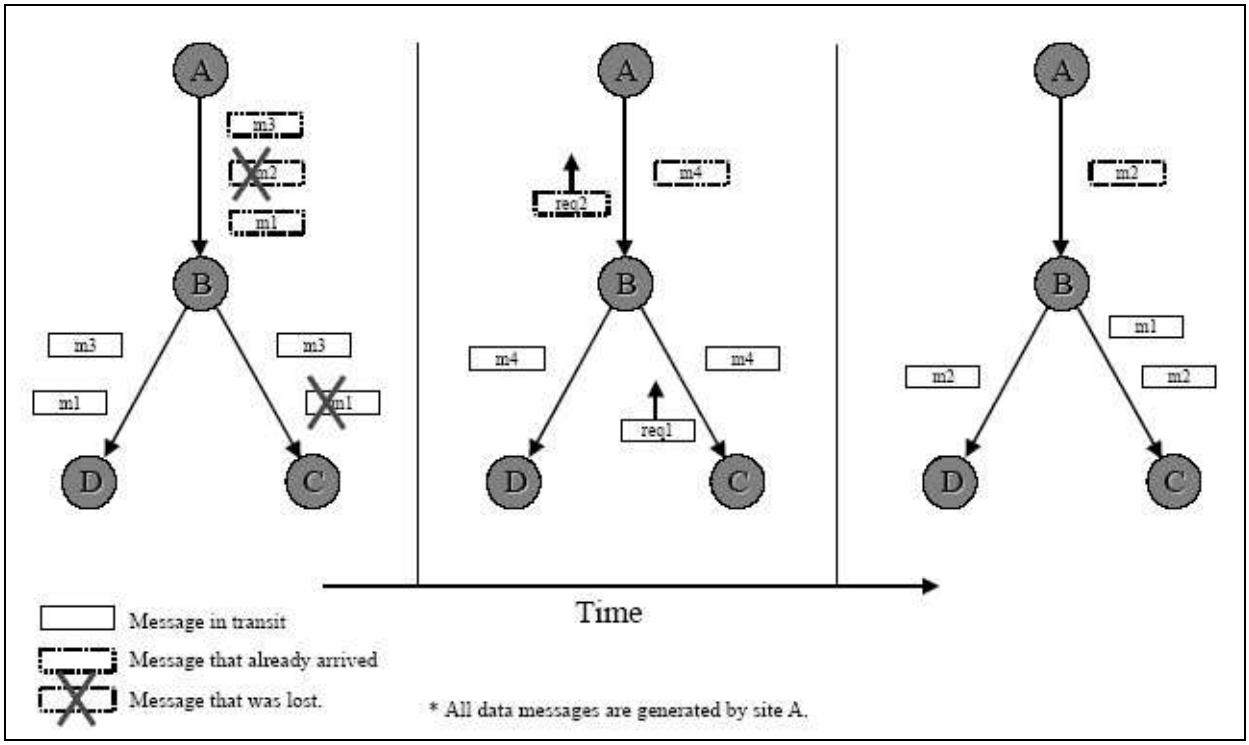
A identificação de qual site está interessado em um pacote é feita quando o pacote é criado no site de origem. É garantido que cada site interessado neste pacote o receba. Isto é mais comum durante o período que uma aplicação pede para se juntar a um grupo ou deixar o grupo, mas a operação ainda não foi completada. Para prover a garantia de ordenação e estabilidade, informações de controle e negociação são enviadas a todos os sites.

Conforme visto anteriormente, a disseminação de pacotes, confiabilidade e controle de fluxo são providos pelos protocolos Hop e Ring, sendo o protocolo Hop destinado à conexão ponto-a-ponto e o Ring para domínios *multicast* e *broadcast*. A seguir, é apresentada uma descrição mais detalhada de cada um destes protocolos.

O protocolo Hop opera com prioridade para o menor tempo de latência possível. Para isto, utiliza um serviço de datagramas não confiáveis da mesma maneira que o popular UDP/IP. Caso ocorra uma perda de pacote, esta dificuldade é tratada pelo próprio nodo (Hop) anterior, ao invés de outros protocolos, como por exemplo, o TCP/IP, onde este tratamento de falhas ocorre no destino do trajeto. Assim, a retransmissão do pacote ocorre imediatamente entre o Hop que obteve o problema e o Hop anterior, mesmo caso não exista uma ordem entre

eles. As vantagens podem ser ainda maiores para casos onde pode-se rodar os protocolos Spread em roteadores que conectam sites envolvidos em comunicação em grupo.

No protocolo Hop é utiliza a mensagem ACK para confirmar o recebimento correto de um pacote e remoção do buffer de envio ou NACK, caso ocorra algo inesperado e seja necessária uma retransmissão do pacote. O controle de fluxo é mantido por um controle de recipiente (*token/leaky bucket*) que limita o número de pacotes enviados simultaneamente e a taxa geral máxima. É utilizada uma janela deslizante para limitar o envio de pacotes que saem de um link, desta forma pode-se prevenir longos tempos de espera para busca de pacotes perdidos.



Fonte: Amir e Stanton (1998, p. 8)

Figura 18 – Protocolo Hop

A figura 18 ilustra o protocolo Hop, como se pode observar trata-se de um *link* bi-direcional, podendo tanto enviar como receber pacotes, de acordo com a figura, em um primeiro momento é perdido o pacote “m2” enviado de A para B, o mesmo problema ocorre no pacote “m1” enviado de B para C. Em um segundo momento C requisita à B o re-envio do pacote “m1” e B requisita à A o re-envio do pacote “m2”. Na última parte da figura tem-se o recebimento dos pacotes perdidos por B e C.

O Hop atua como remetente utilizando-se das seguintes variáveis:

a) *S_highest_linkseq*: maior valor de número de seqüência (*link_seq*) associado a um capote;
b) *S_other_end_aru*: link de seqüência aru (*all-received-up-to*) reportado pela outra ponta do link;
c) *S_cur_windown*: tamanho atual da janela deslizante usada para controle de fluxo.

O receptor possui as seguintes variáveis:

a) *R_highest_seq*: maior valor de número de seqüência no link corrente;
b) *R_link_aru*: valor de seqüência para pacotes recebidos;
c) *Retransmit list*: lista contendo valores de seqüência que foram perdidos e estão aguardando para serem re-transmitidos.

As duas variáveis a seguir são utilizadas tanto pelo transmissor como pelo receptor:

a) *Pkt_cnt_linkack*: quantidade de pacotes enviados ou recebidos desde que o ultimo ACK do link foi enviado;
b) *Max_pkt_btw_ack*: valor limitante referente a quantidade de pacotes que podem ser enviados ou recebidos antes de enviar um ACK.

Os pacotes enviados podem ser classificados em três itens conforme abaixo:

a) dados: são dados da mensagem enviada pelo usuário ou uma mensagem de controle interna do Spread criada por uma camada superior;
b) ACK: cópia do valor do *link_aru* corrente do receptor e o *highest_linkseq* do transmissor;
c) NACK: lista de valores de todas as seqüências de pacotes no qual ocorreu algum tipo de falha ou perda e, portanto precisam ser retransmitidas.

As mensagens de ACK e NACK são limitadas a envio somente ao *link* no qual originaram. Em uma operação normal o lado do *link* Hop irá enviar pacotes de dados e ocasionalmente ACKs vindos do lado do destinatário. Assim procederá enquanto o receptor está de acordo com o transmissor limitado ao tamanho da janela deslizante e não ocorrem perdas de pacotes.

Caso a mensagem recebida em um Hop contenha um número de seqüência maior que seu controle interno (*r_hihest_seq* + 1), então todas as seqüências entre o *r_highest_seq* e o

número da seqüência recebida são adicionados na lista de retransmissão, nestas situações é iniciado um timer, durante este período de tempo será aguardado os pacotes que não foram recebidos, caso tenha sido somente um atraso ou um pacote fora de ordem, o número da seqüência do pacote é retirado da lista de pendências, entretanto, caso este timer se esgote e ainda tenha pacotes na lista, então um NACK é enviado para o emissor requisitando uma retransmissão dos pacotes pendentes.

O pacote de NACK será reenviado com a lista corrente dos pacotes faltantes cada vez que expirar este contador de tempo até que todos os pacotes faltantes sejam recebidos. Cada vez que uma mensagem é novamente requisitada para retransmissão é incrementado um contador, se esta quantidade de vezes alcançar determinado valor, o receptor declara como morto o emissor e realiza alterações de membro de grupo.

Esta alteração de membro do grupo ocorre mesmo que o receptor tenha recebido outras mensagens do transmissor, sendo muito importante eliminar este problema de falha no recebimento, caso contrário o sistema seria forçado a ficar bloqueado eventualmente devido ao buffer ficar cheio.

Quando o transmissor recebe um NACK os pacotes requisitados são adiciona na fila de dados a serem enviados e o NACK é excluído. Os pacotes retransmitidos serão incluídos como dados enviados no calculo do controle de fluxo. Na figura 19 é apresentado parte de um fonte para o protocolo Hop, tem-se as funções de envio de dados (*Hop_Send_Data*), confirmação de recebimento (*Hop_Send_Ack*), notificação de perda de pacote (*Hop_Send_Nack*) e recebimento de pacotes, esta última classifica o conteúdo recebido entre ACK (confirmação de recebimento de pacote), NACK (notificação de pacote enviado perdido) e DATA_PKT (dados).

O protocolo Ring, por sua vez, tem sua aplicação em *layouts* onde se tem mais de um *daemon* em execução no mesmo site.

De acordo com Amir e Stanton (1998), um site é uma coleção de máquinas que tem condições de alcançar outras máquinas por uma única mensagem, ou seja, através de broadcast de hardware, *multicast* de hardware ou *IP-multicast*.

```
Hop_Send_Data(linkid)
     Update token bucket for hop
     while( token is available in bucket for hop )
          Each packet is assigned a unique link_seq value and stored in Link[linkid].open_pkts.
          Send(linkid, pkt)
     if( still_data_available for this hop )
            Event_Queue(Hop_Send_Data, linkid, SendTimeout)

Hop_Send_Ack(linkid)
     ack.link_aru := Link[linkid].r_link_aru
     ack.link_max_sent := Link[linkid].s_highest_linkseq
     Send(linkid, ack)
     Event_Queue(Hop_Send_Ack, linkid, LongHopAckTimeout)

Hop_Send_Nack(linkid)
     add all link_seq values found in retransmit list to nack.req_list[]
     Send(linkid, nack)
     Event_Queue(Hop_Send_Nack, linkid, HopNackTimeout)

Hop_Recv()
     recv(pkt)
     case(pkt.type)
     ACK:
          if (Link[linkid].s_other_end_aru < ack.link_aru)
             remove packets with linkseq < ack.link_aru from Link[linkid].open_pkts[]
             Link[linkid].s_other_end_aru := ack.link_aru
          if (Link[linkid].r_highest_seq < ack.link_max_sent)
             add sequence numbers from r_highest_seq to link_max_sent to retransmit list
             Event_Queue(Hop_Send_Nack, linkid, HopNackTimeout)
     NACK:
          for (i from 0 to nack.num_req-1 )
              req_linkseq := nack.req_list[i]
              pkt := Link[linkid].open_pkts[req_linkseq % MAXPKTS]
              queue_packets_to_send(linkid, pkt)
     DATA_PKT:
          Link[linkid].pkt_count++;
          if (Link[linkid].pkt_count == 1)
              Event_Queue(Hop_Send_Ack, linkid, ShortHopAckTimeout)
          if (Link[linkid].pkt_count > Max_Count_Between_Acks)
              Send_Link_Ack(linkid)
          if (msg_frag.link_seq == Link[linkid].r_highest_seq + 1)
              /* Right on time packet without drops */
              if (Link[linkid].r_link_aru == Link[linkid].r_highest_seq)
                  Link[linkid].r_link_aru++
                  Link[linkid].r_highest_seq++
          else if (msg_frag.link_seq <= Link[linkd].r_highest_seq)
               /* Duplicate or delayed packet */
               remove msg_frag.link_seq from retransmit list and update lowest_missing_linkseq
               if duplicate packet
                    return Null
               if (lowest_missing_linkseq == NONE)
                    Link[linkid].r_link_aru := Link[linkid].r_highest_seq
                    Event_Dequeue(Hop_Send_Nack, linkid)
               else if (msg_frag.link_seq < lowest_missing_linkseq)
                    Link[linkid].r_link_aru := lowest_missing_linkseq - 1
          else
               /* we missed some packets before this one */
               add seq numbers from r_highest_seq to msg_frag.link_seq to retransmit list
               Link[linkid].r_highest_seq := msg_frag.link_seq
               Event_Queue(Hop_Send_Nack, linkid, HopNackTimeout)
          return packet
     Otherwise:
          return packet

* Upper layers will call Event_queue( Hop_Send_Data, Linkid, 0) when data has been queued to send.
```

Fonte: Amir e Stanton (1998, p. 10)

Figura 19 – Protocolo Hop

Cada participante de grupos possui um identificador distinto e inalterável. Mesmo em quedas da rede ou outros problemas este identificador se manterá privado àquele membro. Este protocolo, Ring, é uma modificação do protocolo Ring implantado no projeto Totem

(MOSER, L. E. et al), onde foram utilizados para prover confiabilidade, fluxo de controle global e ordenamento global.

A utilização do protocolo Ring no mecanismo Spread tem como principal função manter um melhor grau de confiabilidade em nível de empacotamento e o controle de fluxo interno em um site local. Além disto, o protocolo Ring proporciona estabilidade na camada de mensagens entre membros que compartilhem o mesmo Ring. O ponto crucial é que o mesmo *token* é utilizado para várias finalidades. Em uma única circulação de um *token*, os campos do pacote e da mensagem são atualizados pelo algoritmo de cálculo do Ring. Desta forma o custo em latência e complexidade no protocolo para conseguir informações na camada de mensagens é desprezível. Em contraste com o projeto Totem e o Transis, no Spread o ordenamento global e controle de fluxo são encontrados em nível de transporte, com isto limitando o uso do Ring nas tarefas as quais demonstra-se mais efetivo: ordenação, confiabilidade e disseminação em área local.

Uma coleção inteira de *daemons* pode ser configurada como um único site conectado por roteadmento *IP-multicast*. Este método não apresenta as mesmas vantagens em relação à confiabilidade quando comparado a protocolo de rede WAN, além de poder desempenhar uma baixa performance.

Os campos contidos em um pacote token são:

a) type: regula as exceções durante mudanças de membros;
b) link_seq: número de seqüência mais elevado de um pacote confiável envido no anel;
c) link_aru: número da seqüência do último pacote ao qual todos os membros receberam de forma confiável. Usado para controlar quando um link pode descartar qualquer referência local ao pacote;
d) flow_control: contador de número de pacotes enviados ao anel durante a ultima rotação da ficha, incluindo retransmissões;
e) rtr list: lista com toda seqüência de valores que um token anterior aguarda por retransmição;
f) site_seq: número mais alto da seqüência de mensagens confiáveis com origem no anel. Esta seqüência é local para o site e combinada com o *site_id* que gera um identificador único para cada mensagem enviada no sistema;

g) site_lts: valor LTS mais elevado encontrado por qualquer membro do anel. Isto é utilizado para prover ordenamento consistente e casual para mensagens seguras e com concordância;
h) site_aru: é o número LTS o qual todos os membros do site tenham recebido todas as mensagens com número identificadoras inferior a este;
i) site_aru_modifier: identificador do site membro que modificou o valor do site_aru pela última vez.

Por ficha entende-se um pacote que trafega internamente em um protocolo, segundo Goulart (2002), depois de recebido uma ficha, o *daemon* trata qualquer retransmissão requisitada pelo dono anterior da ficha, em seguida processa as mensagens recebidas pelas aplicações clientes, envia os pacotes até o limite imposto pelo controle de fluxo, depois atualiza a ficha com novas informações e envia para o próximo *daemon* no anel. Após enviar a ficha o *daemon* tentará entregar qualquer mensagem que tenha para as aplicações clientes.

A cada mensagem processada no sistema são atribuídos um *site_seq* e *site_lts* ocasionando no incremento dos contadores. Para cada pacote confiável enviado um único *link_seq* é atribuído e o contador incrementado. Para atualizar os valores do *link_aru* e do *site_aru* na ficha, o *daemon* compara o valor local do aru com aquele da ficha, se o valor local é menor, então o valor da ficha é abaixado para o valor local e o campo *site_aru_modifier* recebe a identificação do *daemon*. Se o valor local é igual ou maior que o valor da ficha, então o *daemon* altera o valor na ficha somente se na variável *site_aru_modifier* estiver com a identificação deste *daemon*, ou o *site_aru_modifier* é zero, indicando que nenhum *daemon* tenha diminuído este valor durante a última passagem. O valor mais alto pode ser calculado por qualquer membro do anel, usando como base o menor recém calculado valor da ficha e o valor da ficha da rodada anterior na ficha.

O protocolo Ring prove o controle de fluxo, limitando o número de pacotes que cada membro pode enviar durante cada rotação da ficha. O número de pacotes que podem ser enviados em todo o anel para cada rodada, e o limite de quanto cada membro individual pode enviar a cada rodada são parâmetros de configuração de performance. O *daemon* simplesmente envia o mínimo de seu limite individual e o limite total menos o valor no campo *flow_control* que foi enviado na ultima vez.

```
Ring_handle_token(token)
     drop token if it is malformed, a duplicate, or the wrong size.
     if( Link[linkid]->highest_seq < token.link_seq ) Link[linkid].highest_seq := token.link_seq;
     answer any retransmit requests which are on the token
     update SiteLTS and SiteSeq values
     Assign site SiteLTS and SiteSeq values to new application messages
     Calculate flow control for this ring
     Send_Data(linkid)
     update tokens flow control fields
     add any link_seq values I am missing to the token req_list[]
     update Link[linkid].my_aru
     update token.link_aru, token.set_aru, token.link_seq, token.site_seq, token.site_lts,
            token.site_aru, token.site_aru_modifier
     send token to next daemon in the ring
     calculate Link[linkid].ring_aru based on token.link_aru and last_token.link_aru
     discard all packets with link_seq < ring_aru from Link[linkid].open_pkts[] array
     calculate site_aru (Highest_ARU[my site]) based on token.site_aru and last_token.site_aru
     copy token to last_token
     Deliver_Mess()
```

Fonte: Amir e Stanton (1998, p. 12)

Figura 20 – Protocolo Ring

A figura 20 apresenta o pseudocódigo para o protocolo Ring, conforme pode-se observar, inicialmente é analisado se o *token* é valido, em seguida são retransmitidas todas as requisições do *token* e somente então são feitos os demais processos de cálculos e atualizações das informações.

2.3.2.2 ENTREGA DE MENSAGENS, ORDENAÇÃO E ESTABILIDADE

De acordo com Goulart (2002), a camada de transporte provê as garantias requeridas para a entrega, ordenação de mensagens e a semântica de serviços estáveis. Para que ocorra a disseminação, controle de fluxo local e confiabilidade na transmissão dos pacotes a camada de transporte utiliza-se da camada de rede. Uma mensagem enviada pelo Spread percorre vários *links* até chegar ao seu destino. Prevendo casos de falhas o protocolo de transporte mantém em cada *daemon* uma cópia completa das mensagens enviadas ou recebidas até que se torne estabilizada através de todo o sistema, e desta forma possa prover a retransmissão de mensagens que foram perdidas durante as trocas entre os membros do grupo.

Ordenação e estabilidade são providas por um protocolo nível de transporte simétrico que roda em todos os *daemons* ativos. Usa uma seqüência única de valores identificando cada mensagem que origina do site, o identificador único assinalado a cada site, e um tempo de relógio lógico definido pela relação de *Lamport* (aconteceu antes), para prover um ordenamento total em todas as mensagens no sistema e para calcular quando a mensagem se tornou estabilizada. Estabilidade é definida como uma mensagem que um *daemon* sabe que

todos os outros *daemons* relevantes tenham recebido e então esta mensagem pode ser removida do sistema.

Para cada mensagem são designados os seguintes valores antes da disseminação:

a) uma identificação do site;
b) uma seqüência no site;
c) um tempo tipo *Lamport* (LTS – *Lamport Time Stamp*);
d) uma seqüência da sessão.

Uma mensagem pode ser completamente ordenada baseada estritamente nas informações contidas na mensagem. O uso de um tempo de relógio lógico e valor de seqüência têm um beneficio substancial de ser completamente descentralizado e fazer a recuperação de partições e intercalar da rede de maneira consistente com o uso do *Extended Virtual Synchrony*.

As mensagens do tipo não confiáveis possuem um comportamento diferente do adotado pelas mensagens confiáveis por não necessitarem dos mesmos recursos mais complexos existentes em mensagens confiáveis, onde existe a certificação da entrega. Já que as mensagens não confiáveis podem ser descartadas sem ocasionar prejuízos, mantém-se este controle a parte para que não haja interferência entre os dois tipos. A razão de que elas tenham um valor de seqüência, contudo é para ter certeza que o *daemon* não faça entrega de cópias duplicadas da mensagem para a aplicação e para ter uma forma de identificar uma específica mensagem e então o *daemon* possa recompor esta mensagem a partir de seus pacotes.

Caso sejam entregues somente alguns fragmentos de uma mensagem do tipo confiável, portanto não a mensagem integral, após um período de tempo específico estes fragmentos são descartados e a mensagem é perdida por completo. Mensagens não confiáveis são ainda enviadas somente quando o *daemon* tem a ficha no anel local por causa de assuntos de controle de fluxo. Mensagens não confiáveis são enviadas assim que as mensagens cheguem completas e não fazem parte do protocolo principal de envio descrito a seguir.

As mensagens do tipo confiável utilizam-se das propriedades de confiabilidade existentes na própria rede em que atuam. Cada link garante transporte confiável dentro de um tempo limite, independente de falhas no processador ou na rede física. Assim todas as

mensagens são garantidas contra falhas através de pacotes enviados entre os *daemons* utilizando-se dos protocolos Hops e Rings. Mensagens confiáveis são enviadas assim que todos os fragmentos são recebidos e a mensagem encontra-se inteira, já que não existe qualquer tratamento quanto à ordenação, o que evita qualquer atraso por parte de outras mensagens. Mensagens FIFO têm a mesma garantia de confiabilidade que mensagens confiáveis. Também tem-se garantia do envio após todas as mensagens da mesma sessão com menor valor de seqüência de sessão. Para ordenamento FIFO por sessão, o Spread ocasiona em um pequeno aumento no custo da memória por sessão. Em um sistema ativo com muitas sessões este custo é pequeno comparado com as áreas de *buffer* de mensagens requeridas e pode ainda ter considerável beneficio para latência da mensagem.

De acordo com Moser et al. (1994), mensagens do tipo *Agreed* são enviadas em ordem consistente com ambas ordenações FIFO e tradicional causal. Esta ordenação é consistente através dos grupos como um todo. O Spread minimiza os custos deste método, pois requer somente uma mensagem de cada site, não de cada *daemon*. O único potencial método mais rápido requer um seqüenciador o que introduz um recurso centralizado. Assim sendo, um seqüenciador não pode prover EVS em redes particionadas.

Mensagens do tipo seguras são enviadas em ordem consistente com ordenação combinada. A estabilidade é determinada de forma hierárquica. O campo *Site_aru* do anel local presente em cada site é utilizado para gerar um valor *All-Received-Upto* (ARU) para todo o site. Este valor representa o status de estabilidade do site local e é então propagado para todos os outros sites usando a camada de rede física. O valor mínimo do ARU de todos os sites determina a estabilidade global das mensagens. Mensagens seguras iguais ou abaixo deste valor podem ser enviadas e todas as mensagens enviadas iguais ou abaixo destas podem ser descartadas. Spread usa diversas técnicas para otimizar os cálculos necessários para a ordenação combinada, tais como tabelas *hash*, listas multi-dimensionais ligadas e *caching*.

3 DESENVOLVIMENTO DO TRABALHO

Neste capítulo é tratado sobre o desenvolvimento do protótipo proposto, abordando assuntos como ferramentas e técnicas utilizadas na implementação, bem como modelagens e especificações que facilitaram o entendimento de como o protótipo deveria ser desenvolvido. Posteriormente é contextualizado o funcionamento do sistema.

Este protótipo teve como principais objetivos a troca de mensagens entre usuários, transferências de arquivos de um usuário a outro e disponibilização de salas de *chat*. Para realização do mesmo foram utilizados conceitos de troca de dados aplicando os paradigmas cliente/servidor (utilizado para integrar o usuário ao sistema o qual conecta-se no servidor), P2P (utilizado para a comunicação usuário-usuário dispensando o fluxo de dados através do servidor quando as condições permitirem) e o paradigma de comunicação em grupo onde empregou-se o mecanismo Spread (utilizado para disponibilizar as salas de *chat*).

Na especificação e modelagem foram utilizadas as ferramentas Power Designer da SYBASE (2004) e o Rational Rose da RATIONAL (2004). Para a implementação empregou-se a linguagem Object-Pascal no ambiente de desenvolvimento Delphi 7 da BORLAND (2004). Empregou-se uma biblioteca para usar o mecanismo Spread, tendo em vista que não é disponibilizada uma versão nativa do Spread para a linguagem Object-Pascal. Pela facilidade de uso e por ser gratuito para os fins não comerciais, optou-se pelo MySQL 5.0.1-alpha como bando de dados.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

A seguir á apresentada uma relação dos requisitos indispensáveis para que seja alcançado o objetivo desejado com o protótipo de software desenvolvido.

Os requisitos principais do protótipo de software são:

a) enviar mensagens entre usuários (RF);
b) disponibilizar salas de *chat* para dois ou mais usuários (RF);
c) possibilitar transferências de arquivos entre dois usuários (RF);
d) permitir busca com filtros por usuários cadastrados no sistema (RF);
e) cadastrar usuário através do próprio modulo cliente do protótipo (RF);
f) executar em sistema operacional Microsoft Windows 98 ou superior (RNF);
g) possuir ambiente amigável e intuitivo para usuários leigos (RNF);

h) conectar usando *modems* de velocidade de 56kbps ou superior (RNF);
i) disponibilizar várias opções de status do usuário (RF);
j) notificar usuário da mudança de status de amigos da lista de contato (RF);
k) armazenar lista de contatos no servidor (RF);
l) executar em máquinas com 200Mhz de processador e 64mb RAM ou superior (RNF);
m) carregar lista de usuários ao conectar no servidor (RNF);
n) permitir adicionar e remover usuários da lista de contatos (RF);
o) armazenar no servidor mensagens enviadas para usuários *offline* (RF);
p) possibilitar mudanças no perfil de cadastro do usuário (RF);
q) permitir consulta a perfil de usuários do sistema (RF).

3.2 ESPECIFICAÇÃO

Para a especificação optou-se pela ferramenta Rational Rose da RATIONAL (2004), onde foram elaborados os diagramas de classes (empregando metodologia orientada a objetos), diagramas de caso de uso e diagrama de seqüência. Para a modelagem dos dados no módulo servidor utilizou-se o Power Designer da SYBASE (2004).

A seguir são apresentados os diagramas na seguinte seqüência: caso de uso do módulo cliente; caso de uso do módulo servidor; diagrama de classes do módulo cliente; diagrama de classes do módulo servidor; diagramas de seqüência do módulo cliente; modelo físico dos dados. O diagrama de seqüência do módulo servidor não é apresentado devido ao reduzido número de classes, sendo praticamente todas as funcionalidades executadas entre duas ou mais classes nomeadas de tCliente, que é utilizada na manipulação de cada cliente.

3.2.1 DIAGRAMAS DE CASO DE USO

De acordo com Furlan (1998), o diagrama de caso de uso descreve a visão externa do sistema e suas interações com o mundo externo, não se preocupando como deve ser implementado.

O caso de uso passa uma idéia das funcionalidades do sistema com alto nível de abstração, permitindo entendimento fácil e rápido do sistema mesmo por pessoas com pouco conhecimento de informática.

A figura 21 demonstra os principais casos de uso encontradas no módulo cliente.

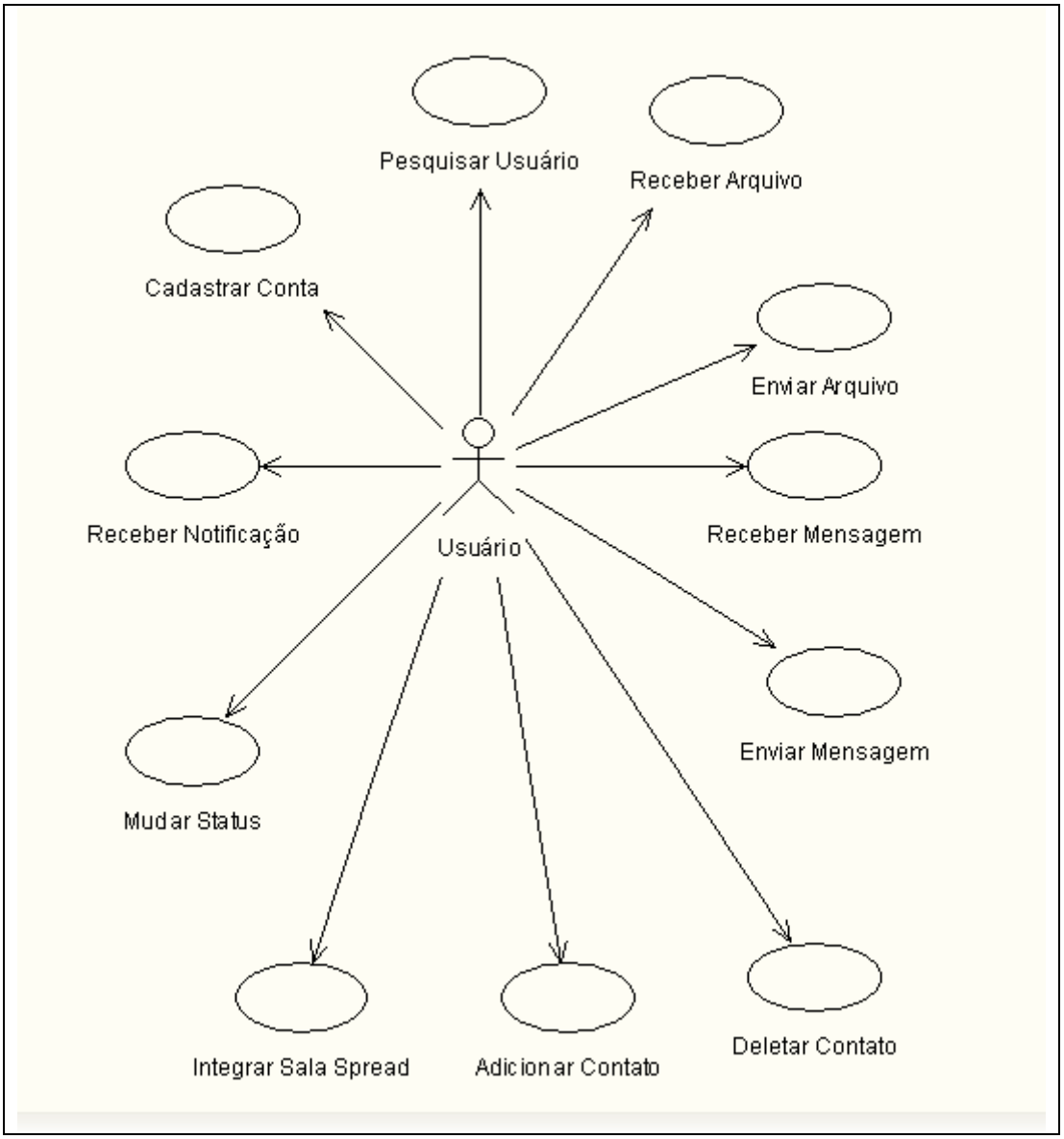


Figura 21 – Caso de uso (módulo cliente)

Na seqüência encontra-se uma relação que descreve de forma resumida cada funcionalidade do caso de uso do módulo cliente:

a) pesquisar usuário: o usuário informa alguns dados da pessoa que deseja entrar em contato, por exemplo nome, idade e cidade. Os dados informados são enviados ao servidor que busca no banco de dados contas que atendam a estas requisições retornando ao usuário uma seleção de clientes;

b) receber arquivo: o usuário recebe um arquivo de um dos seus membros presentes na lista de contatos. Este arquivo é enviado de forma P2P, não tendo qualquer relação com o servidor;
c) enviar Arquivo: o usuário convida outro usuário da sua lista a receber um arquivo, caso aceite, uma nova conexão entre os dois usuários é criada e o arquivo é enviado caso uma conexão do tipo P2P seja estabelecida;
d) receber mensagem: o usuário recebe uma mensagem. Esta mensagem pode ser P2P caso o remetente seja um dos contatos da lista e não tenha nenhum obstáculo que impeça esta conexão, como exemplo *firewalls*. Esta mensagem pode ser recebida também através do servidor, em casos de problemas em receber diretamente de outro usuário, mensagens enviadas enquanto *offline* ou mensagens do próprio servidor;
e) enviar mensagem: o usuário envia uma mensagem a algum outro usuário. Caso não seja possível enviar de forma P2P ou não se tenha o endereço IP do destinatário, esta mensagem é enviada através do servidor, que re-encaminhará a mensagem ao outro usuário. Caso o usuário esteja *offline*, a mensagem será armazenada no banco de dados;
f) deletar contato: requisita ao servidor a eliminação de um dos usuários da lista de contatos;
g) adicionar contato: requisita ao servidor a inclusão de um usuário na lista de contatos;
h) integrar sala Spread: conecta a um servidor Spread e integra uma sala de *chat*, podendo enviar e receber mensagens para todos os usuários que estiverem fazendo parte daquela sala;
i) mudar *status*: o usuário modifica seu *status* (*online*, *offline*, ocupado, ausente);
j) receber notificação: o usuário recebe notificação da mudança de *status* de algum outro usuário que faz parte da sua lista de contatos;
k) cadastrar usuário: o usuário cria uma nova conta de acesso ao sistema.

Na figura 22 é apresentado o diagrama de caso de uso do módulo servidor.

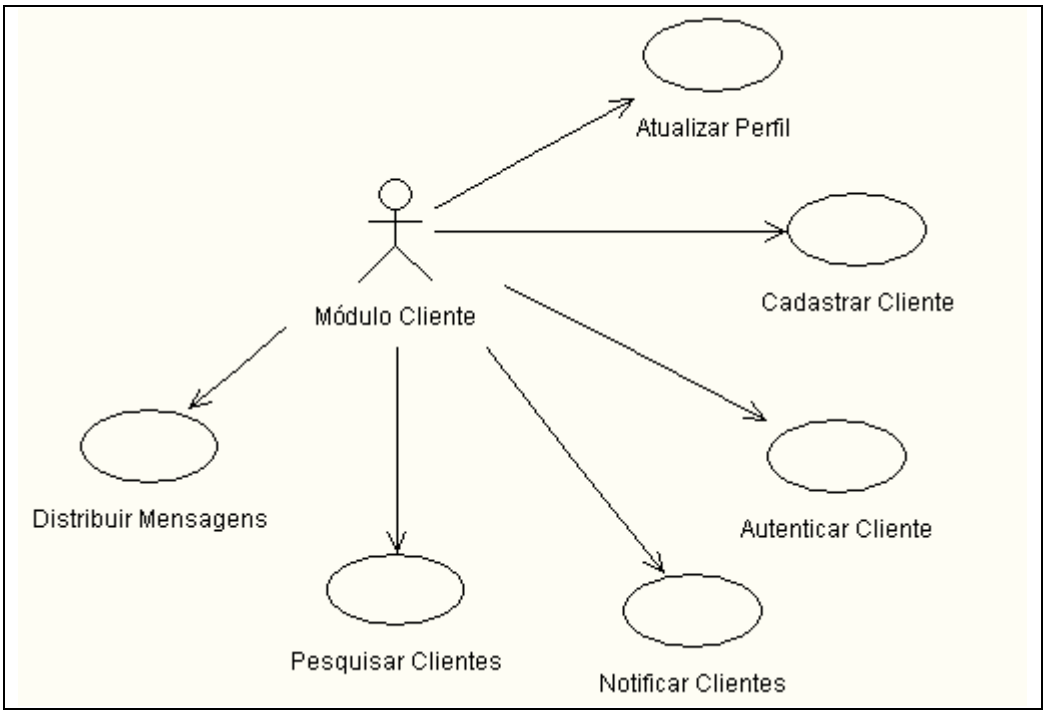


Figura 22 – Caso de uso (módulo servidor)

Na seqüência uma breve descrição das funcionalidades vistas na figura 22:

a) atualizar perfil: atualiza perfil de um cliente, podendo alterar dados do seu cadastro como por exemplo: nome, cidade e apelido. Estes dados do perfil são os mesmos consultados quando se realiza uma busca por usuários;
b) cadastrar cliente: cadastra um novo cliente no banco de dados;
c) autenticar cliente: verifica se a senha do usuário que esta tentando conectar está correta, em caso positivo habilitando-o ao funcionamento do sistema, ou em caso negativo desconectando-o;
d) notificar clientes: notifica clientes que tenham relação com algum cliente que modificou seu *status*. Por exemplo: caso o usuário A tenha na sua lista de contatos o usuário B e o usuário B passe de *online* para *offline*, uma notificação é enviada a A relatando a mudança;
e) pesquisar clientes: pesquisa por clientes que atendam a determinadas características devolvendo a relação resultante da busca ao usuário que a requisitou;
f) distribuir mensagens: re-encaminha mensagens entre usuários.

3.2.2 DIAGRAMA DE CLASSES

Segundo Furlan (1998), o diagrama de classes apresenta a estrutura lógica contendo uma coleção de elementos declarativos, como classes, tipos e seus respectivos conteúdos e relações.

Na figura 23 tem-se o diagrama de classes do módulo cliente, apresentando as classes e seus relacionamentos. Optou-se em excluir os métodos e propriedades deste diagrama devido ao número de métodos e classes, principalmente no módulo cliente, desta forma facilitando a visualização.

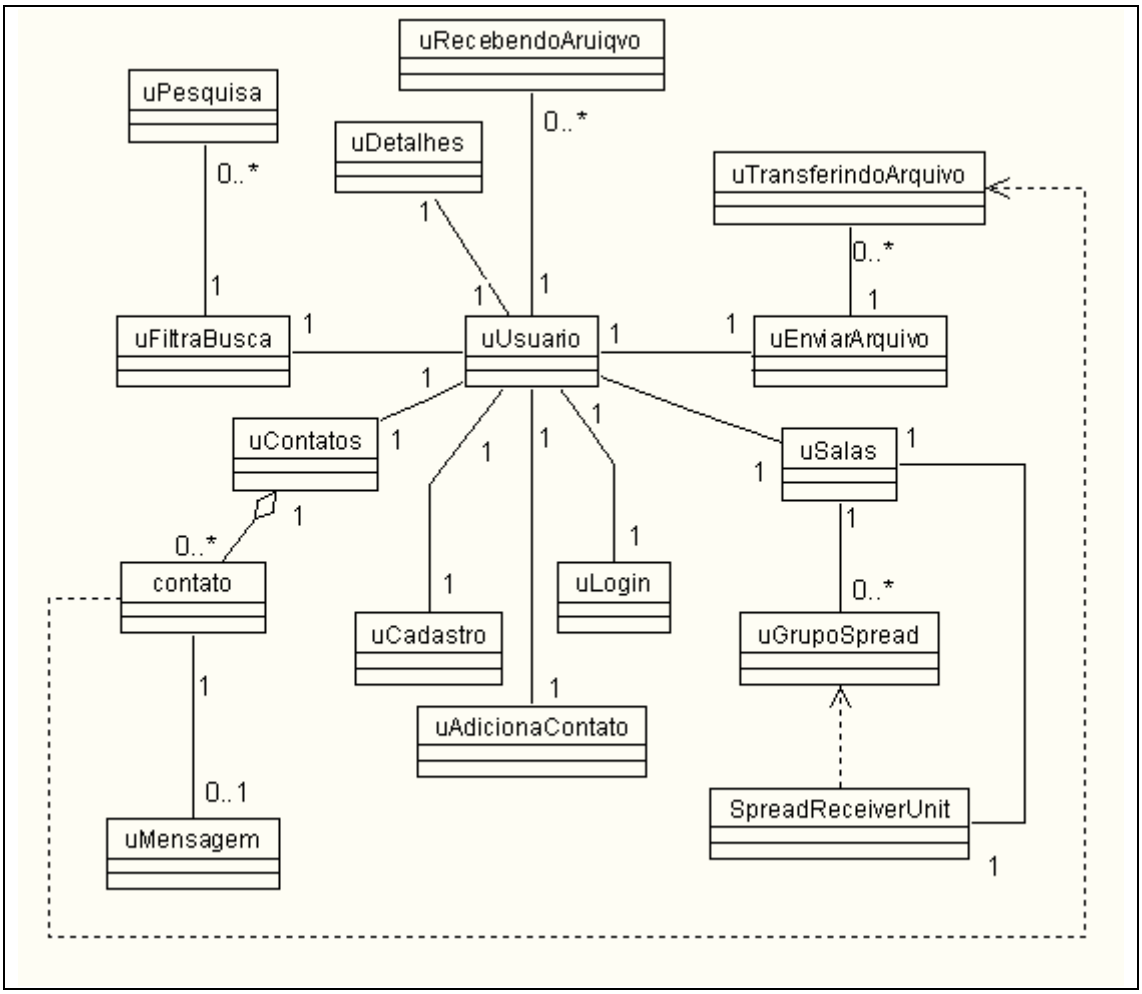


Figura 23 – Diagrama de classes (módulo cliente)

Na seqüência tem-se uma breve descrição da funcionalidade das classes empregadas no módulo cliente:

a) uRecebendoArquivo: classe utilizada para receber um arquivo de outro cliente. Esta classe sustenta a conexão que recebe os dados do arquivo informando ao usuário em que estágio encontra-se a transferência;
b) uUsuario: responsável pelas chamada da maioria das operações, é a classe que se comunica com o servidor e interpreta os dados recebidos;
c) uTransferindoArquivo: classe que manipula o envio de um arquivo a outro contato, informando ao usuário em que estado está a transferência;
d) uEnviarArquivo: classe que especifica qual arquivo será enviado, para quem e envia o convite ao destinatário;
e) uSalas: conecta e desconecta o usuário de um mecanismo Spread, além de criar salas de *chat* através do Spread;
f) uGrupoSpread: sala de *chat* que recebe e envia mensagens em um grupo do mecanismo Spread no qual esteja conectado, para cada grupo em que o usuário encontra-se ligado, um objeto uGrupoSpread é criado;
g) SpreadReceiverUnit: classe que faz o gerenciamento da conexão a um mecanismo Spread, recebendo e enviando dados através da interface da *unit* sp_func, a qual utiliza a biblioteca cliente Spread;
h) uLogin: classe responsável pela conexão do usuário ao sistema de mensagens, enviando os dados ao servidor, em caso autenticado, recebe a lista de contatos e habilita as funcionalidades do sistema;
i) uAdicionarContato: requisita ao servidor a adição de um novo contato a lista do usuário;
j) uCadastro: conecta e cadastra uma nova conta de usuário no servidor do sistema, após o cadastro devidamente realizado é informado o identificador da conta cadastrada;
k) uMensagem: classe que gerencia o recebimento e o envio de mensagens entre clientes, para cada cliente é usada um objeto uMensagem quando realizado o envio ou recebimento de mensagem;
l) contato: representa um contato da lista de contatos, contendo seu status, endereço IP, *socket* de conexão e componentes gráficos para visualização do usuário;
m) uContatos: classe que gerencia a lista de contatos;
n) uFiltraBusca: faz a requisição de usuários que atendam a determinadas características;

o) uDetalhes: classe com o perfil de determinado cliente para consulta, ou do próprio usuário, neste caso permitindo a atualização do mesmo;
p) uPesquisa: lista de usuários encontrados em determinada pesquisa.

Na figura 24 é apresentado o diagrama de classes do módulo servidor.

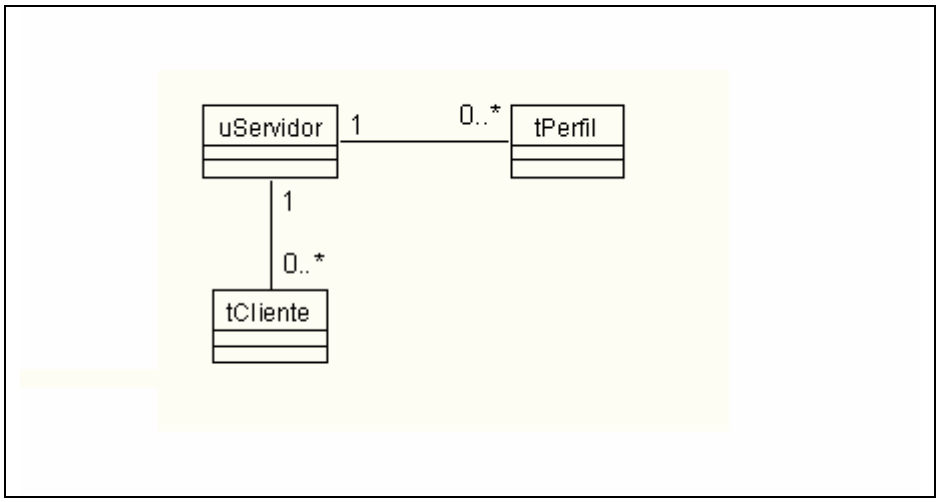


Figura 24 – Diagrama de classes (módulo servidor)

Na seqüência tem-se uma breve descrição das classes empregadas no módulo servidor:

a) uServidor: classe principal do servidor, faz a análise dos dados que são recebidos dos clientes, acessos ao banco de dados, cadastramento, autenticação e demais funcionalidades não referentes a clientes já instanciados;
b) tPerfil: representa unicamente o perfil de um cliente, utilizada com o objetivo único de facilitar a manipulação dos dados no servidor;
c) tCliente: classe instanciada para cada cliente, responsável pelas ações entre clientes no próprio servidor, como envio de mensagens, notificação da mudança de status aos contatos, mudanças na lista de contatos e demais funcionalidades relacionadas entre clientes. Para cada cliente autenticado é criado pelo menos um objeto de tCliente.

3.2.3 DIAGRAMAS DE SEQUÊNCIA

De acordo com Furlan (1998), o diagrama de seqüência é uma forma de representar um comportamento que inclui uma seqüência de troca de mensagens entre um conjunto de objetos dentro de um contexto para realizar um propósito específico, tal como a realização de um caso de uso.

Na figura 25 é representado o diagrama de seqüência da operação que ocorre quando o usuário adiciona ou remove um de seus contatos da lista.

Para adicionar um novo contato na lista do usuário é informado o identificador deste contato, então é requisitado ao servidor que faça a adição, depois de realizada a adição, o usuário recebe a notificação do status atual do novo usuário adicionado.

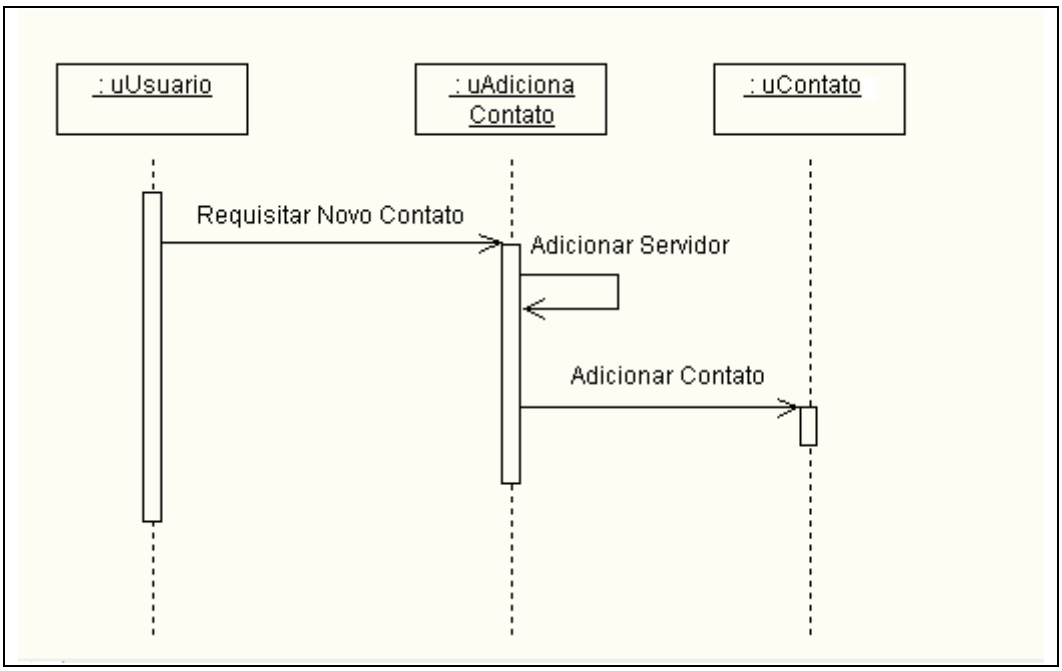


Figura 25 – Diagrama de seqüência (adicionar contatos)

O diagrama da figura 26 representa o modo de como arquivos são recebidos e enviados entre dois clientes.

Quando um convite para receber arquivo é recebido o protótipo analisa se o pedido vem de algum dos contatos encontrados na lista do usuário. Caso seja encontrado na lista e o usuário aceite o arquivo, é criado um objeto de uRecebendoArquivo que demonstra a transferência do arquivo. Para enviar um arquivo é requisitado qual dos clientes da lista receberá e qual arquivo. Caso o arquivo seja aceito pelo destinatário, é criado um objeto uTransferindoArquivo que se encarregará da transferência.

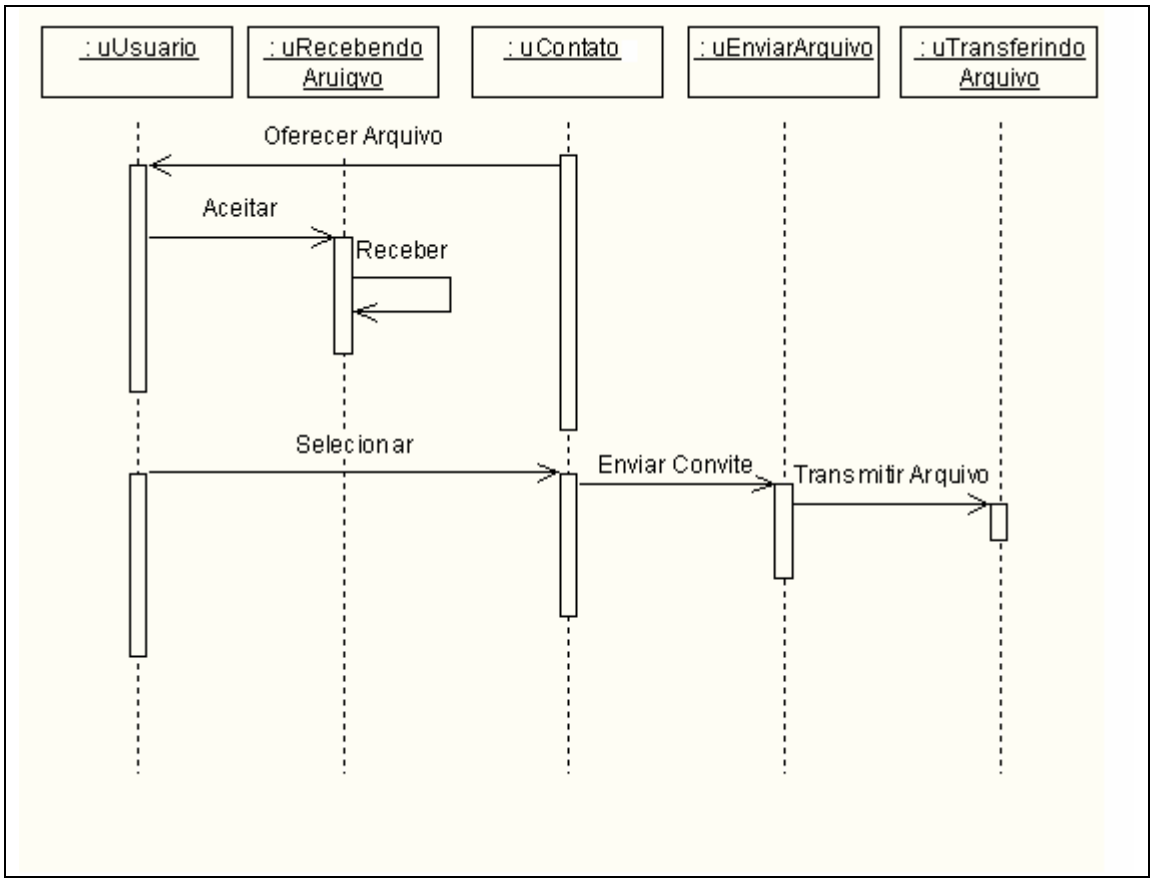


Figura 26 – Diagrama de seqüência (enviar e receber arquivos)

O diagrama de seqüência mostrado na figura 27 representa a pesquisa por outros usuários. Quando uma busca é requisitada são informadas as características obrigatórias dos usuários que se deseja encontrar, após ser enviado ao servidor é aguardado durante alguns segundos alguma resposta do servidor, caso este responda e encontre pelo menos um cliente que atenda as requisições, uma relação de clientes é apresentada ao usuário.

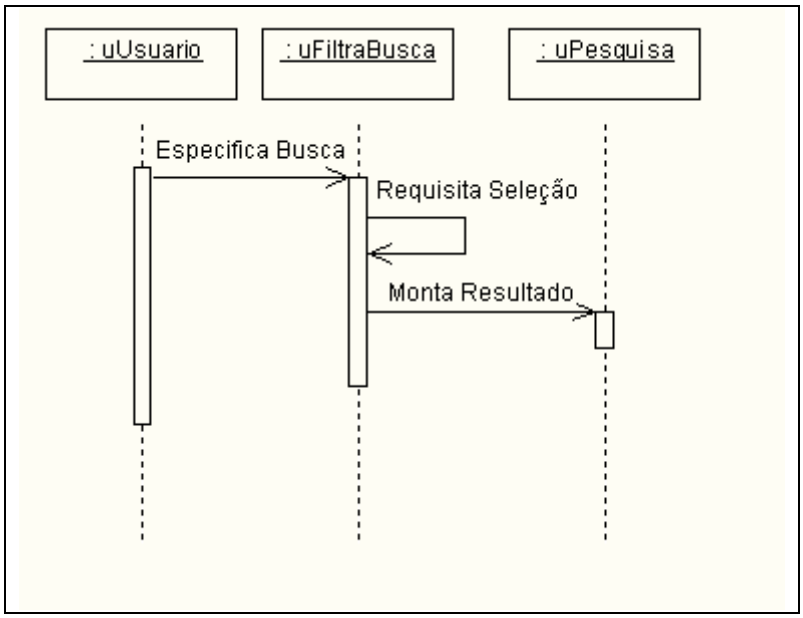


Figura 27 – Diagrama de seqüência (pesquisar usuário)

O diagrama a seguir (figura 28) mostra o diagrama de seqüência da operação de cadastro de usuário.

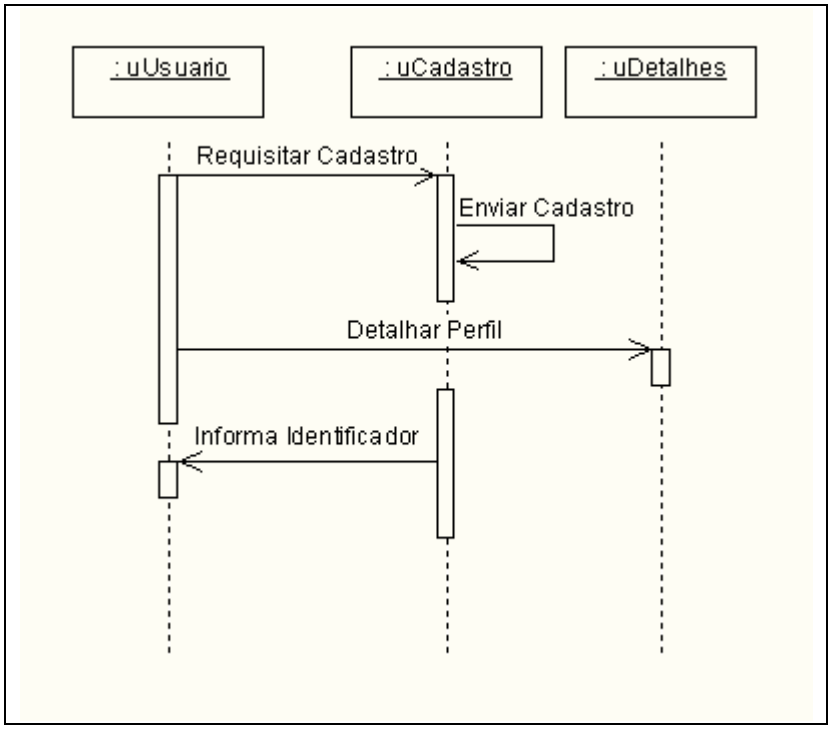


Figura 28 – Diagrama de seqüência (cadastrar usuário)

Ao requisitar o cadastro de uma nova conta, a classe uCadastro requisita as informações básicas para que ocorra um cadastro, caso o usuário queira definir melhor seu

perfil de usuário é chamado o objeto uDetalhes que permite informar mais detalhes, depois que a requisição é enviada o sistema aguarda que o servidor informe o identificador da conta cadastrada. Na figura 29 tem-se o diagrama de seqüência que representa a conexão do usuário ao servidor.

A classe uUsuario chama o objeto uLogin que requisita os dados para conectar-se ao servidor, quando os dados são enviados é acionado um temporizador que aguardará uma resposta do servidor ou cancelará a operação por *time-out*, caso a conexão seja bem sucedida a classe uUsuario recebe notificações sobre os contatos que integram a lista de contatos do usuário, para cada contato do qual se recebe notificação é instanciado um objeto de uContato.

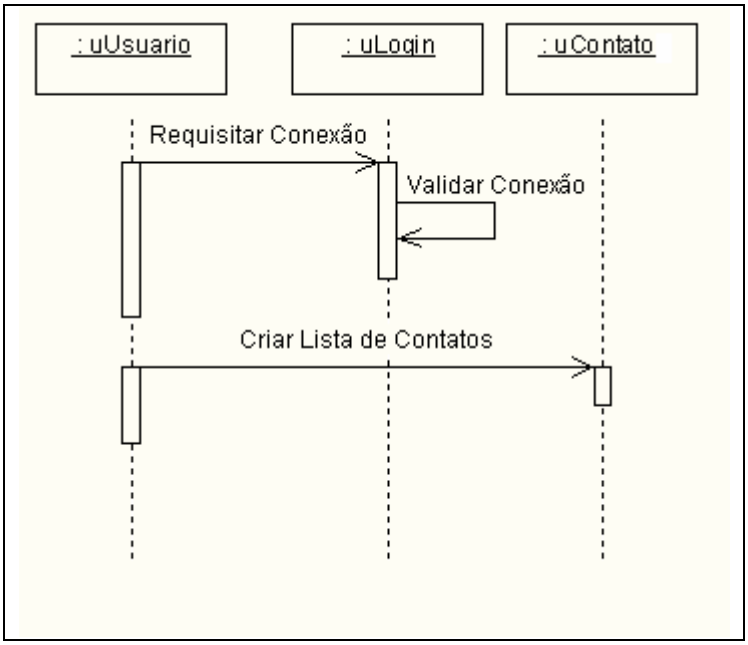


Figura 29 – Diagrama de seqüência (conectar usuário)

O diagrama de seqüência da figura 30 representa o envio e recebimento de mensagens que podem ser relacionados ao servidor ou a outro usuário.

No caso do envio de uma mensagem após selecionar qual o contato que receberá a mensagem é mostrada a janela de mensagem, no caso de ainda não estar instanciada para o contato selecionado, é criado um novo objeto de uMensagem ligada ao objeto uContato que representa o contato que receberá a mensagem. Após o usuário redigir a mensagem o protótipo tenta enviar de forma P2P diretamente ao outro cliente, caso não obtenha sucesso é enviada através do servidor.

Ao receber uma nova mensagem ela é passada ao objeto uMensagem do contato correspondente que a enviou e o usuário é alertado sobre seu recebimento.

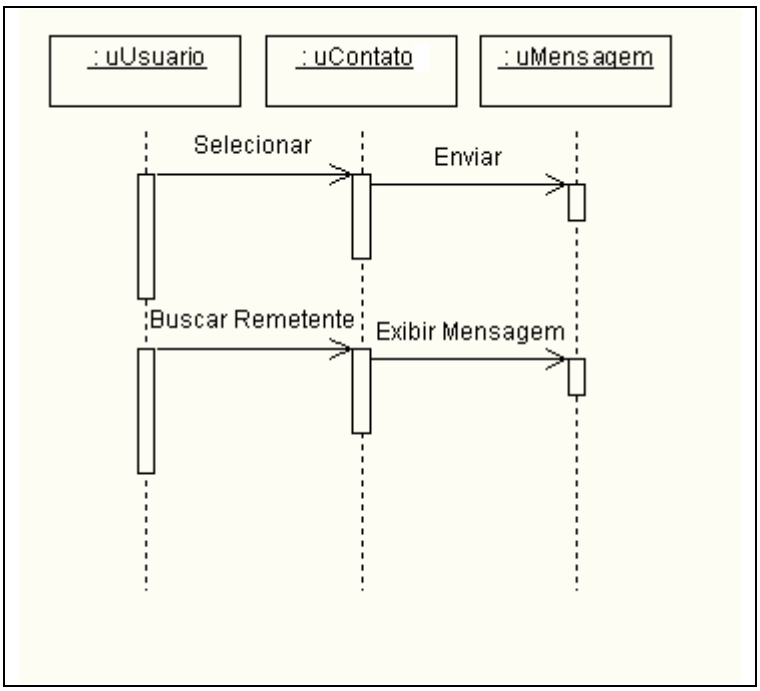


Figura 30 – Diagrama de seqüência (enviar receber mensagem)

Na figura 31 encontra-se o diagrama de seqüência da consulta ao perfil de usuários. Um perfil pode ser consultado a partir de um outro usuário conectado ao sistema ou seu próprio perfil, neste último caso e permitido alterá-lo.

Após ser selecionado de qual usuário deseja-se consultar o perfil, uma requisição de perfil é enviada ao módulo servidor e o cliente entra em estado de espera de perfil durante alguns segundos. Quando o módulo servidor recebe a requisição uma busca no bando de dados é realizada a partir da identificação do usuário que se deseja o perfil, caso a busca obtenha sucesso encontrando um usuário, estes dados são retornados ao módulo cliente que os requisitou. Caso o módulo cliente receba um perfil enquanto estiver no estado de espera de perfil, um objeto uDetalhes é montado e exibido ao usuário.

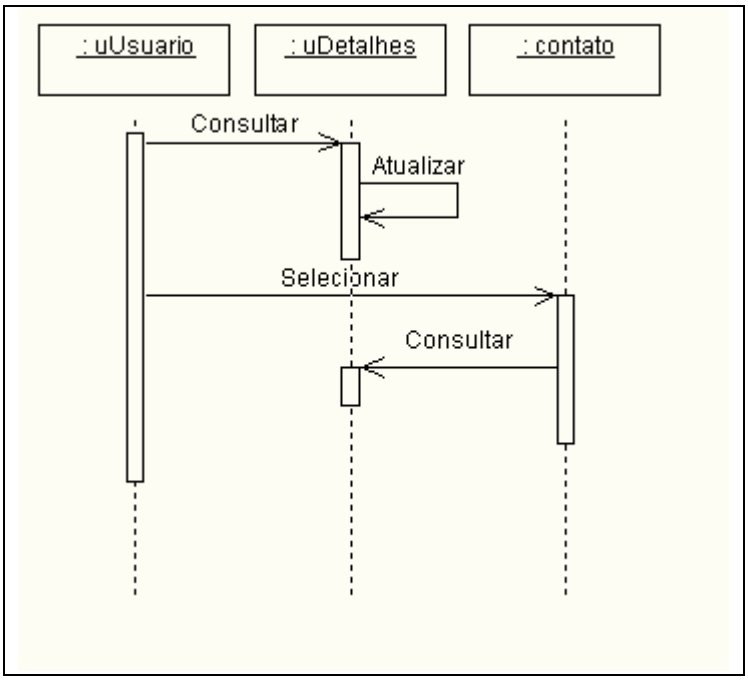


Figura 31 – Diagrama de seqüência (consultar perfil)

Na figura 32 são apresentadas as funcionalidade referente às salas de *chat* que utilizam o mecanismo Spread.

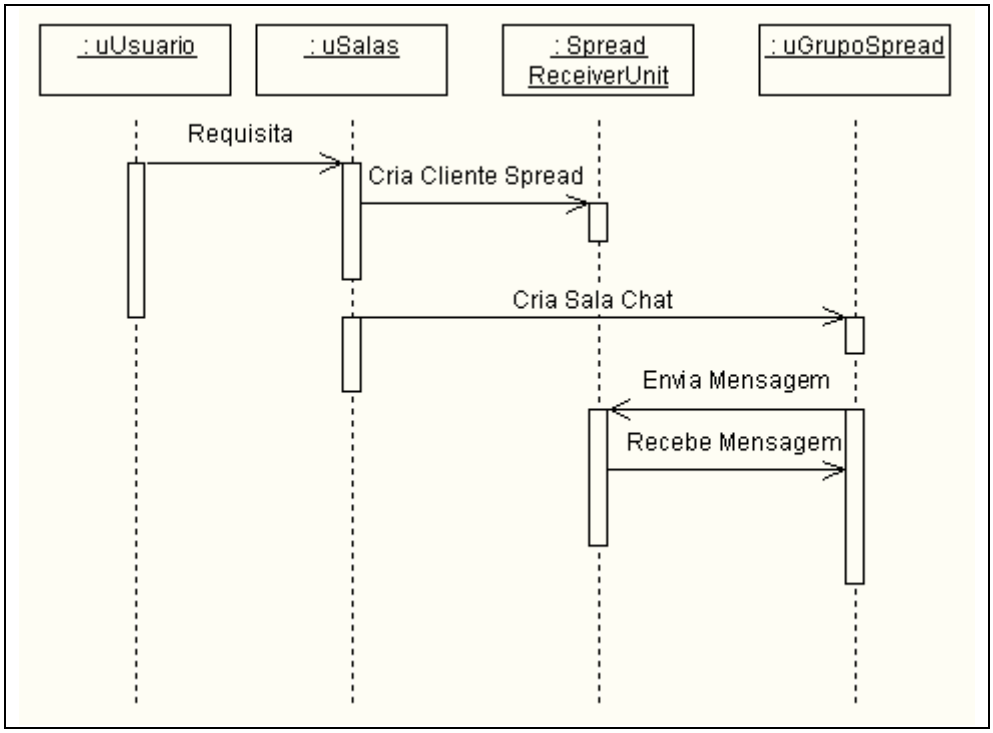


Figura 32 – Diagrama de seqüência (integrar sala de chat)

Na classe uSalas é informado o endereço do servidor Spread permitindo a conexão. Após estar conectado é possível integrar o número de salas que se desejar, para cada sala que o usuário ingressar um novo objeto uGrupoSpread é criado no módulo cliente.

Um objeto SpreadReceiverUnit será instanciado a partir do primeiro momento que o usuário tentar conectar a um mecanismo Spread, as mensagens são recebidas e enviadas por esta classe que trabalha juntamente à *unit* sp_func e acessa a biblioteca de cliente Spread. Ao receber uma mensagem pelo Spread, a classe SpreadReceiverUnit trata a mensagem e envia ao uGrupoSpread correspondente.

3.2.4 MODELAGEM DOS DADOS FÍSICOS

Na Figura 33 é apresentada a modelagem física dos dados encontrados no módulo servidor.

Esta estrutura de dados é acessada unicamente via servidor. Qualquer modificação ou consulta aos dados por parte do módulo cliente serão somente requisições interpretadas pelo módulo servidor, e em caso de serem viáveis e devidamente autorizadas, o módulo servidor encarrega-se de executá-las, apenas passando um resultado ao cliente que não possui condições de saber onde ou como os dados estão sendo armazenados e manipulados.

Conforme observa-se na figura 33, são mantidos os dados do usuário em uma tabela chamada “Usuarios”, nesta tabela são mantidos dois campos referentes a países, país de residência e de trabalho, estes dois campos possuem chaves as quais são relacionadas à tabela “Paises”. O campo “t_area” armazena a chave que, relacionada com a tabela “AreasTrabalho”, especifica a área de trabalho do usuário.

A tabela “Usuario_idoma” é usada para relacionar os usuários aos idiomas por eles conhecidos, para esta ligação são utilizados como chaves estrangeiras o código do usuário e o código do idioma, respectivamente encontrados nas tabelas “Usuarios” e “Idiomas”. Atualmente o módulo cliente permite somente até três línguas por usuário, entretanto pode-se expandir facilmente esta opção tendo-se em vista que no banco de dados não existe um limite.

Na tabela “MensagensOffline” são mantidos os dados de mensagens que foram enviadas enquanto um usuário encontrava-se *offline*, toda vez que um usuário conecta-se são

buscadas estas mensagens e caso alguma mensagem seja encontrada esta é eliminada do base dados e passada ao cliente.

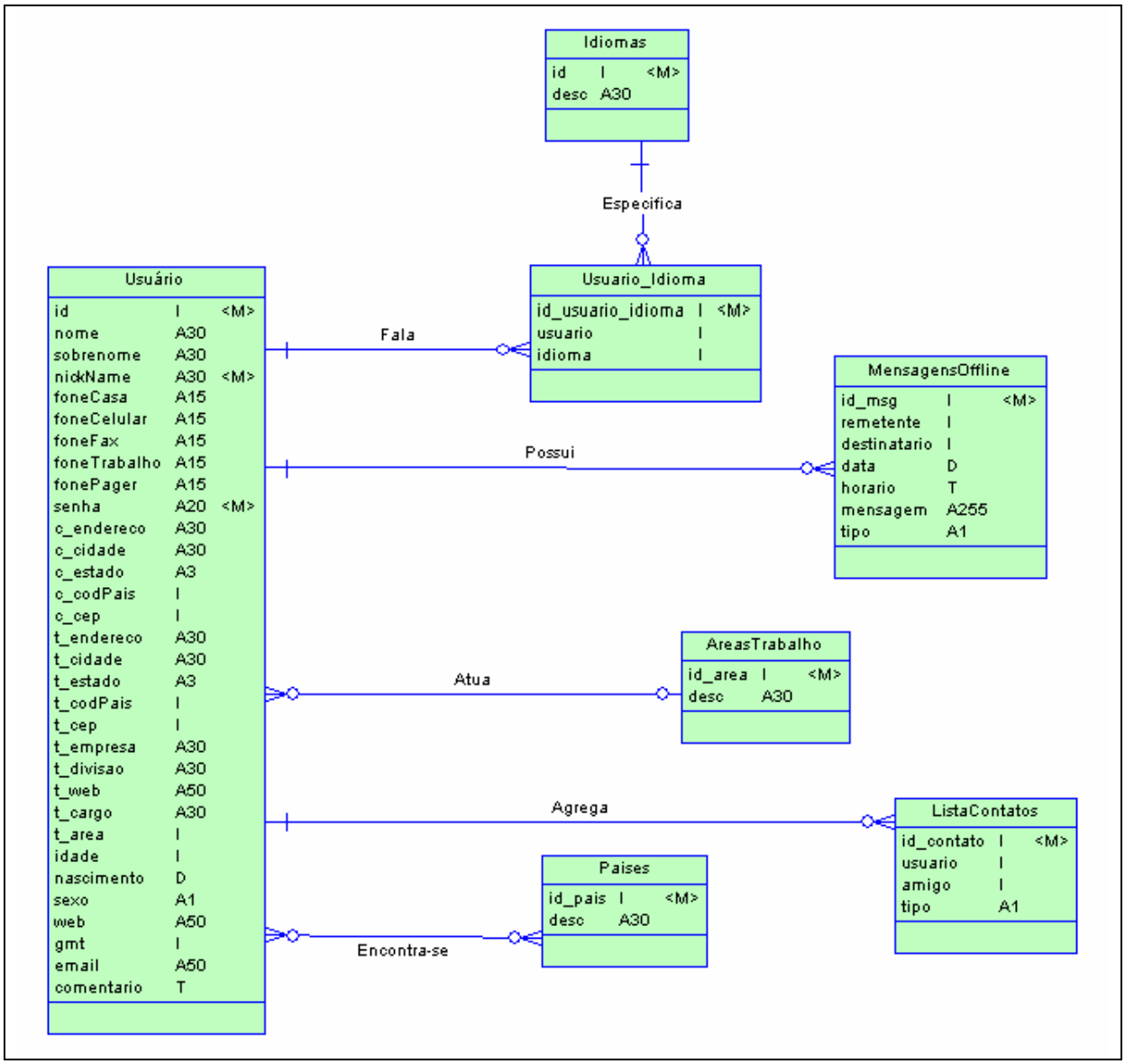


Figura 33 – Modelagem física dos dados

Por último, tem-se a tabela “ListaContatos”, que é carregada para cada usuário que se conecta ao sistema, desta forma podendo manter sua lista de contatos mesmo acessando o sistema através de outra máquina. O campo “tipo” é mantido nesta tabela com o objetivo de facilitar a adição de novos recursos como um tratamento diferenciado a cada contatos, como por exemplo apresentar sempre um *status* específico para determinado contato.

3.2.5 CONTEXTUALIZAÇÃO DO FUNCIONAMENTO DO PROTÓTIPO

A figura 34 contextualiza o funcionamento do protótipo. É representado o módulo servidor, o banco de dados e três módulos clientes.

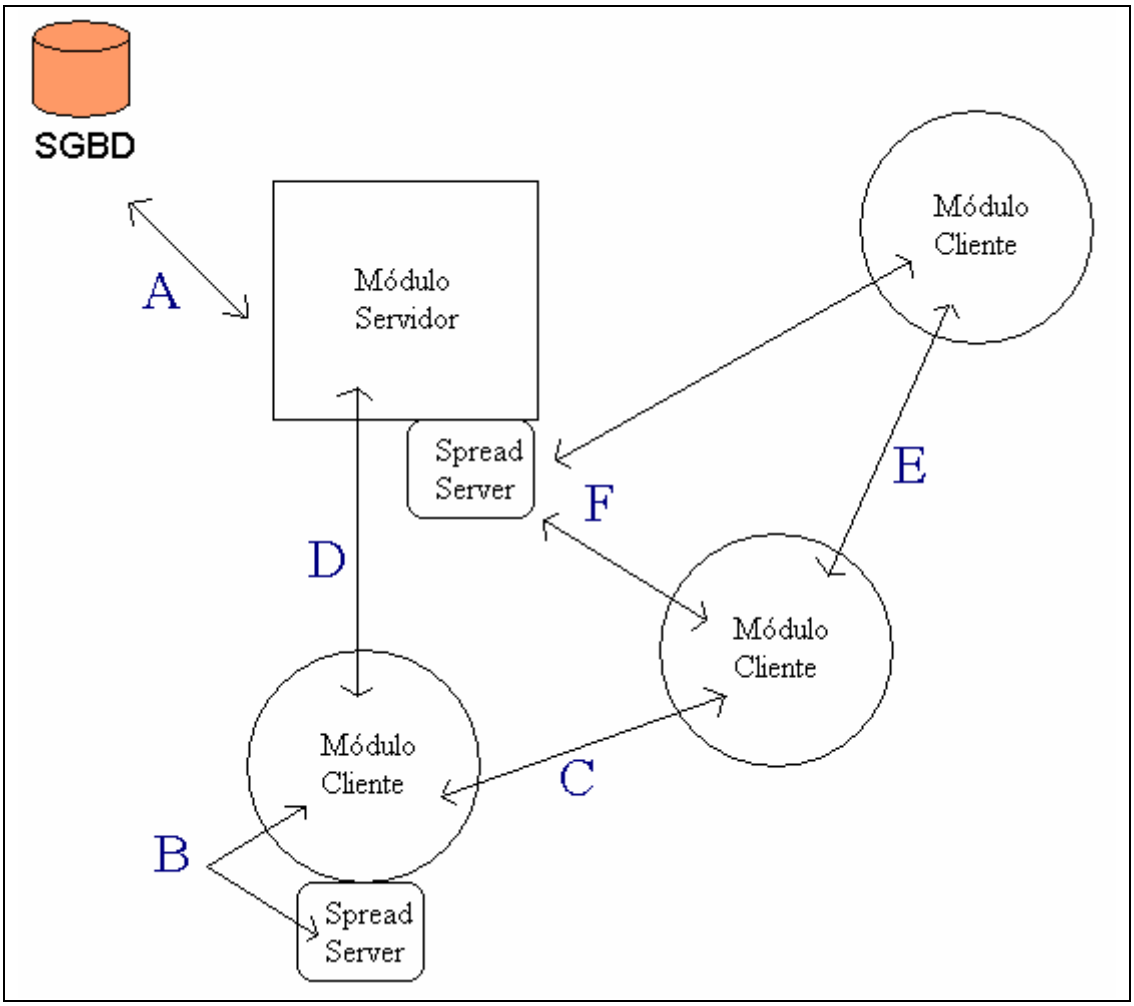


Figura 34 – Contextualização do funcionamento do protótipo

Na seta “A” é representado o uso do banco de dados pelo módulo servidor, podendo ser uma consulta de usuários, um novo cadastro de usuário, uma autenticação, modificação de perfil, modificação de lista de contatos ou ainda manipulação de mensagens *offline*.

Em “B” tem-se o uso do servidor Spread que foi inicializado no próprio módulo cliente, podendo-se também inicializar um servidor Spread no módulo servidor como é representado nas ligações “F”.

Nas ligações “C” e “E” é representado conexões do tipo P2P, esta conexão é utilizada para enviar arquivos e mensagens entre dois usuários autenticados no mesmo módulo servidor.

A ligação entre o módulo servidor e o módulo cliente está representada pela seta “D”, esta ligação possibilita a autenticação do usuário, envio de mensagens *offline* ou entre usuários que não puderam ligar-se de forma P2P, mudança de status, consulta de perfil, recebimento de notificações sobre outros contatos, cadastro de novos usuários e a manutenção da lista de contatos.

3.3 IMPLEMENTAÇÃO

Neste tópico serão apresentadas considerações referentes à implementação do modelo proposto neste trabalho, descrevendo ferramentas e técnicas utilizadas para a realização do mesmo, e posteriormente a operacionabilidade do protótipo.

3.3.1 TÉCNICAS E FERRAMENTAS UTILIZADAS

Os itens encontrados nesta seção abordam assuntos referente às ferramentas, técnicas e mecanismos utilizados na implementação do protótipo. A implementação ocorreu em dois módulos, um atuando como servidor e outro como cliente, como visto anteriormente, ambos implementados na linguagem Object-Pascal utilizando o ambiente de programação Delphi 7 da BORLAND (2004). O banco de dados ao qual o módulo servidor se relaciona é o MySQL 5.0.1-alpha da MYSQL AB (2004). Para gerenciar a comunicação das salas de Chat foi empregado o mecanismo de comunicação em grupo Spread versão 3.16.02.

3.3.1.1 SPREAD

O mecanismo Spread pode ser obtido em “*www.spread.org*” e encontra-se em versões binárias ou fonte, disponíveis em várias linguagens, entre elas Java, C++ e Python. Para estas três linguagens é possível obter-se também a documentação com informações referentes a instalação, configuração e utilização do mecanismo para implementação.

No protótipo que foi desenvolvido empregou-se uma biblioteca para utilização do Spread com o Delphi 7. Na parte cliente da aplicação encontra-se o arquivo “libtsp.dll”, que é chamado a partir do Delphi e executa o papel de cliente Spread. No mesmo computador que é executado o módulo servidor, ou em qualquer outro, é possível executar o servidor Spread

permitindo que os módulos clientes se conectem a este mecanismo, ressaltando que tanto o endereço de conexão do servidor Spread, quanto o módulo servidor do sistema não se encontram fixos, permitindo ao usuário selecionar os mesmos.

Na figura 35 pode-se observar um trecho da *unit* “sp_func” do Delphi, a qual liga-se à biblioteca “libtsp.dll” que realizará as funções de cliente Spread.

```
function SP_multigroup_multicast(mbox, service_type, num_groups:Integer;
         const Groups:TGroups; mess_type:SmallInt;
         mess_len:Integer; const mess:pChar):Integer; CDECL; external 'spread\libtsp.dll';


function SP_multigroup_scat_multicast(mbox, service_type, num_groups:Integer;
              const Groups:TGroups; mess_type:SmallInt;
              const scat_mess:pScatter):Integer; CDECL; external 'spread\libtsp.dll';

function SP_receive(mbox:Integer; service_type:pInteger;
        sender:TGroup; max_groups:Integer;
        num_groups:pInteger; Groups:TGroups;
        mess_type:pSmallInt; endian_mismatch:pInteger;
        max_mess_len:Integer; mess:pChar ):Integer; CDECL; external 'spread\libtsp.dll';

function SP_scat_receive(mbox:Integer; service_type:pInteger;
       sender:TGroup; max_groups:Integer;
       num_groups:pInteger; Groups:TGroups;
       mess_type:pSmallInt; endian_mismatch:pInteger;
       scat_mess:pScatter):Integer; CDECL; external 'spread\libtsp.dll';

{ returns offset in memb. message of gid (group id), num_vs and vs_set }
function  SP_get_gid_offset_memb_mess:Integer; CDECL; external 'spread\libtsp.dll';
function  SP_get_num_vs_offset_memb_mess:Integer; CDECL; external 'spread\libtsp.dll';
function  SP_get_vs_set_offset_memb_mess:Integer; CDECL; external 'spread\libtsp.dll';
function  SP_poll(mBox:Integer):Integer; CDECL; external 'spread\libtsp.dll';
function  SP_equal_group_ids(g1, g2:group_id):Integer; CDECL; external 'spread\libtsp.dll';
procedure SP_error(error:integer); CDECL; external 'spread\libtsp.dll';
```

Figura 35 – Código fonte (unit sp_func)

Para o funcionamento e ativação do servidor do Spread deve-se configurar inicialmente os arquivos “spread.conf” e “spread_access_ip”. O arquivo “spread.conf” fornece a configuração que o servidor Spread irá utilizar, algumas desta configurações são:

a) DebugFlags: especifica quais tipos de informações serão relatadas ou não, como por exemplo “DebugFlags = { ALL !DATA_LINK}”, onde será relatado todas as operações com exceção dos dados recebidos como mensagens, o símbolo “!” nega a opção seguinte;

b) EventLogFile: seleciona em qual arquivo será gerado o relatório contendo as informações geradas durante a execução do servidor;

c) EventTimeStamp: permite selecionar a inserção ou não de etiquetas de data e hora a cada operação relatada, bem como que formato da data e hora preferido;

d) DangerousMonitor: permite o desligamento de algumas operações que podem danificar o sistema se ocorridas de forma inesperada, muito indicado o seu desligamento caso a rede na qual opera o servidor não esteja devidamente protegida;

e) Spread_Segment: especifica o conjunto de segmentos de endereço IP que serão aceitos em tentativas de conexão ao mecanismo, liberando acesso as funcionalidades do Spread somente a estes endereços pré-definidos.

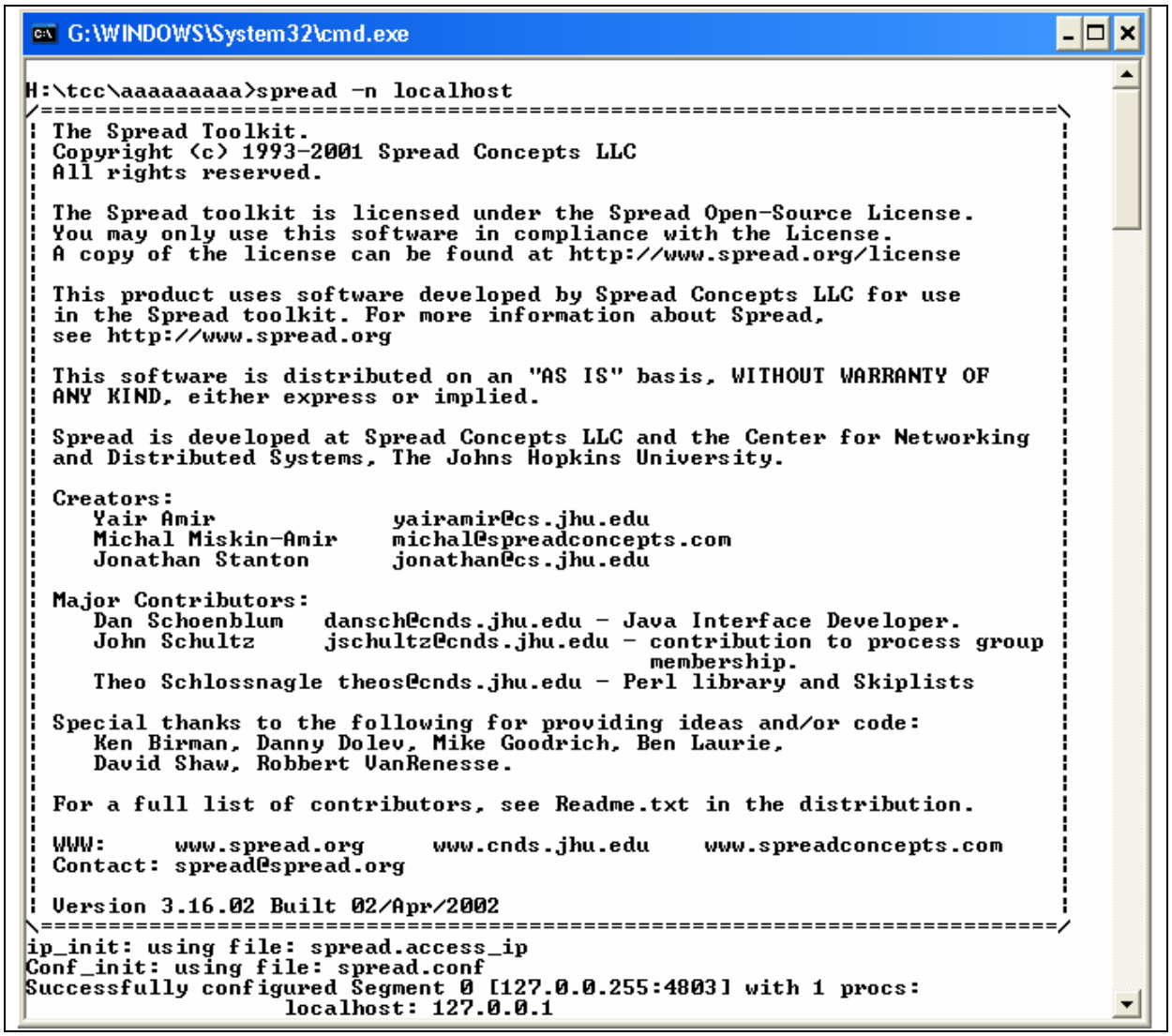


Figura 36 – Servidor Spread em execução

O arquivo “spread_access_ip” não é necessariamente utilizado, serve para restringir o acesso ao servidor, permitindo somente os endereços IP previamente listados neste arquivo.

Após devidamente configurados os arquivos, o servidor Spread pode ser executado através do arquivo “spread.exe”. Alguns parâmetros podem ser passados na chamada deste executável, para uma lista das opções disponíveis passa-se o parâmetro “/?”.

Caso obtenha-se sucesso na execução do servidor, tem-se uma tela similar a mostrada na figura 36.

As principais funções implementadas no cliente Spread e empregadas no módulo cliente do protótipo são:

a) SP_connect: utilizado para conectar-se a um servidor Spread;
b) SP_Receive: recebe uma mensagem do buffer de mensagens do Cliente;
c) SP_disconnect: desconecta cliente do servidor Spread;
d) SP_join: integra cliente a um grupo do servidor ao qual encontra-se conectado;
e) SP_Leave: deixa um grupo ao qual faz parte;
f) SP_multicast: envia uma mensagem a um grupo específico.

```
function TthrdSpreadReceiver.JoinGroup(const stGroupName: String): Boolean;
var
  iSPResult: Integer;
begin
  Result:=  false;
  if self.Connected then
  begin
    iSPResult:= SP_join(FiMailBox, PChar(stGroupName));
    if iSPResult= 0 then
      result:= True
    else
  end;
end;


end.
```

Figura 37 – Rotina para integrar um grupo Spread

Na figura 37 é representada a rotina na qual utiliza o comando *SP_join* para integrar um grupo. Neste exemplo é passado como parâmetro uma string contendo o nome do grupo ao qual deseja-se integrar. Os parâmetros aplicados no comando *SP_join* são respectivamente

o endereço de memória onde localiza-se o cliente em execução e o endereço do grupo. O endereço de memória do cliente é instanciado durante a própria tentativa de conexão da *unit* SpreadReceiverUnt. Na rotina apresentada na figura 37 é inicialmente verificado se o usuário encontra-se conectado e então é chamada a função de integrar o grupo. Caso esta função retorne zero a operação ocorreu de forma correta.

3.3.1.2 MYSQL

Segundo MYSQL AB (2004), o servidor de banco de dados MySQL é o banco de dados de código aberto mais popular no mundo. Sua arquitetura torna-o extremamente rápido e de fácil personalização. O alto grau de re-uso do código e um método minucioso para produzir características ricas em funcionalidades resultaram em um sistema gerenciador de banco de dados incomparável em velocidade, solidez, estabilidade e de fácil desenvolvimento.

Segundo INFOWESTER (2004), o MySQL é um banco de dados relacional gratuito, eficiente e otimizado para aplicações *web*. Por ser direcionado a aplicações que não exijam muito do banco de dados, a quantidade de recursos presentes é relativamente menor, se comparado a outros bancos de dados. Todavia, esta simplificação na quantidade de recursos possibilita que o MySQL execute de forma extremamente rápida e eficiente.

O MySQL é desenvolvido e mantido pela empresa MySQL AB, que também oferece uma versão comercial. Esse SGBD é multi-plataforma, sendo compatível entre outros sistemas operacionais com o Windows, Linux e BSDs. As tabelas criadas podem ter tamanho de até 4 GB. O MySQL é também compatível com várias linguagens de programação, tais como PHP, C, Java, Visual Basic.

3.3.1.3 GERENCIAMENTO DE CLIENTES NO MÓDULO SERVIDOR

O módulo servidor trata cada cliente conectado como um objeto do tipo “tCliente”. Ao receber uma nova tentativa de conexão, um temporizador é inicializado delimitando o tempo limite para que ocorra a autenticação desta nova conexão através da passagem do *login* e a senha, caso este tempo termine antes ou ocorra uma tentativa de autenticação incorreta, esta conexão é desligada.

Em uma conexão que seja autenticada com um cliente válido, é averiguado entre os usuários ativos se algum deles possui o mesmo identificador deste que acabou de ser

autenticado, se for encontrado verifica-se se o *socket* de conexão é o mesmo, neste caso é ignorada a requisição e tratada como um re-envio de conexão, caso contrário, subentende-se que a conexão do cliente caiu por algum motivo inesperado e este está tentando conectar-se novamente, para esta ocorrência são atualizados os dados do cliente de acordo com a nova conexão, em casos onde não se encontrou uma instância deste cliente ainda, este é conectado e uma novo objeto de "tCliente" é criado.

```
if autenticaConexao(id,senha) then
  begin
  tempCliente:=buscaCliente(strToInt(id));
  if  not (tempCliente=nil) then // trata redundancia de pedido de conexao
    if tempCliente.socket=conexao then
      begin
      relatar('Envio redundante de pedido de conexao!');
      exit;
      end;
  i:=buscaTemporizador(conexao);
  deletaTemporizador(i);        // elimina limite de tempo de autenticacao
  if tempCliente=nil then // cliente ainda nao instanciado
    begin
    relatar('Criando usuario '+id);
    tempCliente:=adicionarCliente(strToInt(id));
    end
  else
    begin
    relatar('Cliente ja instanciado');
    end;
  if tempCliente.socket=nil then // cliente estava offline
    begin
    relatar('Procedimento normal, cliente estava offline!');
    end
  else // cliente estava logado mas perdeu conexao anterior
    begin  // redundancia de pedido de conexao tratado anteriormente
    tempCliente.socket.Close;
    relatar('Cliente caiu a conexao e retornou em outra!');
    end;
  tempCliente.socket:=conexao; // atualiza ou atribui conexao
  tempCliente.horario:=time; // renova tempo para n cair por timeOut
  protocoloConectar:=true;
  end
else
  relatar('Senha invalida para usuario '+id);
```

Figura 38 – Rotina para conectar novo cliente no módulo servidor

A seguir é demonstrado na figura 38 um trecho do código fonte encontrado na *unit* "uServidor" responsável em tratar a conexão de um usuário.

Um objeto do tipo tCliente, utilizado no módulo servidor para representar um cliente, é formado principalmente por um identificador único de conta, um apelido ao qual o usuário é visto por outros usuários, um status do usuário, um horário, dois vetores de outros clientes e um *socket* de comunicação.

```
type
  tCliente = class
    socket: TCustomWinSocket;
    id: integer;
    nickName: string;
    status: byte;
    horario: tTime;
    posicao: integer;
    vetorInterno: array of tCliente;  // lista de contatos do cliente
    vetorExterno: array of tCliente;  // lista de contatos que possuem este cli
    public
    constructor Create(identificador,pos:integer);
    procedure receberNotificacao(cliente: tCliente); // recebe notificacao de o
    procedure enviarNotificacao;  // envair notificacao aos usuarios dependentes
    function deletarCliente:boolean; // deletar cliente caso nao tenha dependen
    function desconectar:boolean;
    function modificaStatus(novoStatus: byte):boolean;
    function pertenceListaInterna(identificador:integer):boolean;
    // controle de ligacao entra clientes
    // usando funcoes diferentes para futura implementacao de contatos internos
    function adicionaExterno(quemMeObserva: tCliente):boolean; // adiciona novo
    function deletaExterno(quemMeObservava: tCliente):boolean; // deleta novo c
    function adicionaInterno(quemEuObservo: tCliente):boolean; // adiciona amig
    function deletaInterno(quemEuObservo: tCliente):boolean; // deleta amigo qu
    function trazerNickName(id:integer):string;
    function adicionarAmigo(amigoId:integer):boolean;
    function deletarAmigo(amigoId:integer):boolean;
```

Figura 39 – Estrutura de um objeto tCliente no módulo servidor

A figura 39 demonstra parte do código fonte onde é definido um objeto do tipo tCliente, usado para manipular um cliente conectado ao módulo servidor.

O atributo horário armazena a hora da última operação enviada pelo cliente, desta forma são controlados quais usuários tornam-se ociosos no sistema, ocasionando o desligamento dos mesmos após um período.

Os dois vetores de objetos tCliente foram denominados de “vetorInterno” e “vetorExterno”, estes atributos servem para controlar a interligação entre cada cliente. O vetor “vetorInterno” se encarrega de ligar-se aos usuários encontrados na lista de contatos de um cliente, enquanto que o vetor “vetorExterno” contém ligação com os clientes que possuam em

suas listas de contatos o usuário em questão, proprietário deste “vetorExterno”. Desta forma, a cada modificação sofrida por um cliente, todos os outros clientes relacionados a este usuário através do “vetorExterno” serão notificados da mudança, da mesma forma que quando um cliente tiver algum contato que sofreu modificação, portanto contido no seu “vetorInterno”, uma notificação será recebida no objeto tCliente e encaminhada via *socket* ao usuário final que executa o módulo cliente do protótipo.

Para uma melhor compreensão sobre a relação entre cada tCliente é representado na Figura 40 um exemplo destas ligações. Nesta situação têm-se três usuários, o usuário A possui na sua lista de amigos o usuário B, este por sua vez tem na sua lista os usuários A e C, e o usuário C possui o usuário A. No módulo servidor estes clientes são tratados em três objetos tCliente, o usuário B por possuir dois contatos na sua lista, terá no seu “vetorInterno” ligação com os objetos tCliente do usuário C e do usuário A, entretanto é composto de somente uma ligação no seu “vetorExteno”, pois pertence unicamente da lista de contatos do usuário A.

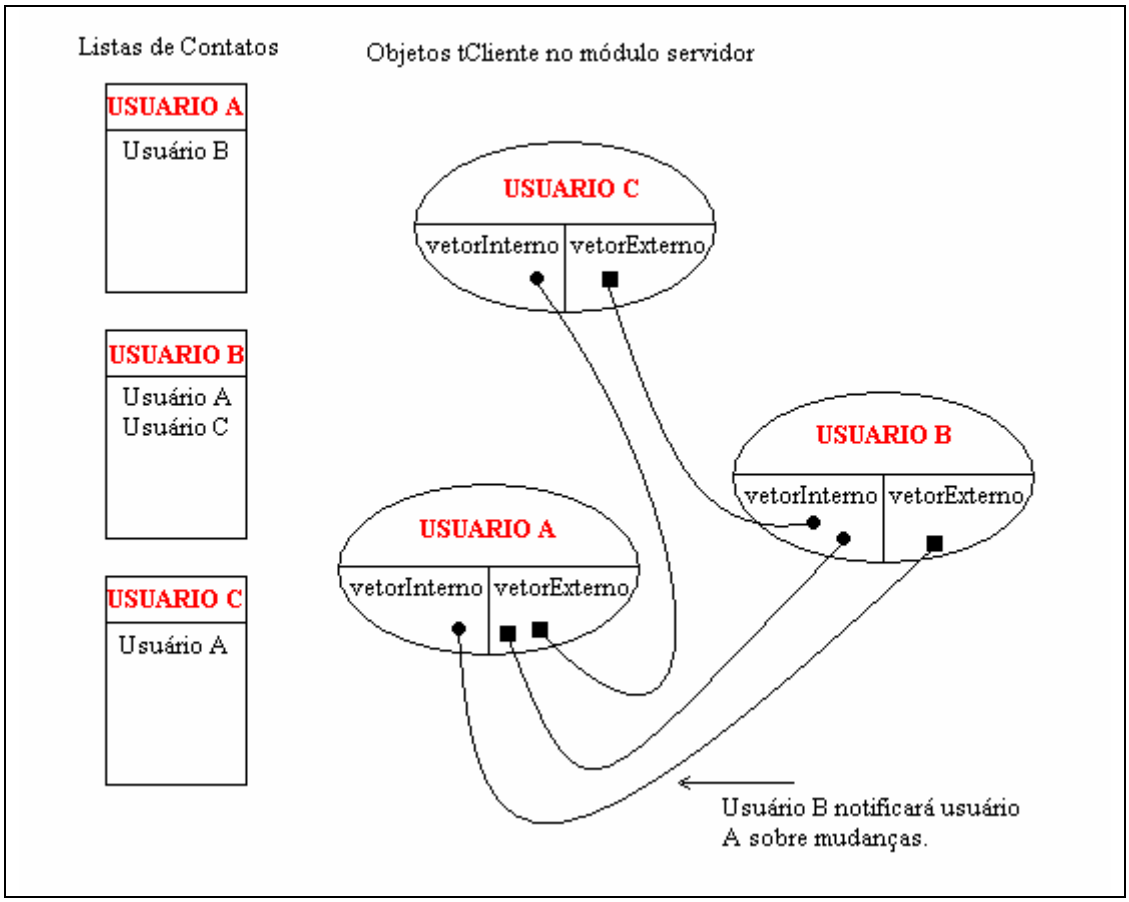


Figura 40 – Estrutura de um objeto tCliente no módulo servidor

Quando um novo cliente for reconhecido pelo módulo servidor e seu objeto tCliente for integrado ao sistema, é buscado no banco de dados sua lista de contatos formando seu próprio “vetorInterno”. Para cada um destes contatos encontrados, o próprio objeto tCliente adiciona-se no “vetorExterno” de seus contatos, assim informando-os que deseja receber notificações sobre seus estados. Logo após um novo objeto ser adicionado ao “vetorExterno”, o “tCliente” proprietário deste “vetorExteno” envia imediatamente a primeira notificação sobre seu comportamento atual ao novo membro.

É importante ressaltar que é mantida fisicamente no banco de dados somente a lista de contatos, sendo que a lista dos clientes que desejam serem notificados é criada dinâmicamente conforme a necessidade. Caso um cliente deseja ser notificado sobre outro ainda não instanciado, este o instancia mantendo-o *offline*, desta forma quando o cliente instanciado conectar-se, ele simplesmente atualiza os dados do seu objeto “tCliente”, conseqüentemente notificando os outros usuários já encontrados no seu “vetorExterno”.

3.3.1.4 TRANSFERÊNCIA DE ARQUIVOS

A transferência de arquivos ocorre unicamente de forma P2P, não sendo possível o envio de arquivos passados através do módulo servidor. Esta limitação tem como objetivo evitar uma sobrecarga que seria ocasionada pelo alto fluxo de dados que passaria no servidor desta forma consumindo muita banda de conexão e prejudicando as principais funcionalidades do mesmo. Entretanto este problema merece um estudo mais aprofundado das reais possibilidades desta operação, tendo em vista que o Yahoo Messenger (YAHOO, 2004) possibilita estas transações via servidor quando não é possível estabelecer uma conexão P2P.

Utilizou-se na implementação da transferência de arquivos os componentes “TIdTCPClient” e “TIdTCPServer” para realizar a transferência, estes componentes respectivamente implementam um cliente e um servidor *multi-threaded*, ambos através do protocolo TCP. Empregou-se também o componente “TIdCompressionIntercept” que tem como o objetivo compactar e descompactar o fluxo de dados da transferência. Todos estes componentes encontram-se nativos no Borland Delphi 7 Enterprise.

3.3.1.5 COMUNICAÇÃO P2P

Além do *status*, identificador e apelido tem-se também o IP de cada usuário ativo na lista de contatos do módulo cliente. Através deste endereço IP é possível estabelecer uma

conexão direta com o outro usuário quando deseja-se enviar um arquivo ou mensagem, com isso alivia-se drasticamente o fluxo de dados no módulo servidor. É representada na figura 41 a estrutura de um objeto que representa um contato da lista de contatos no módulo cliente.

```
type
  itemContato = class
                id: integer;              // identificador do contato
                posicao: integer;         // posicionamento no vetor de conta
                texto: tStaticText;       // texto de nick
                icone: tImage;            // icone de status
                ip: string;               // ip
                status: byte;             // status do usuario
                mensagem: tformMensagem; // form de mensagens
                socket: TClientSocket;    // socket de comunicacao p2p
                end;
```

Figura 41 – Estrutura de objeto que atua como um contato no módulo cliente

Para o envio de mensagens para outro contato o protótipo, inicialmente tenta realizar uma conexão direta com o outro contato através deste endereço IP que é informado pelo servidor, caso a operação não seja bem sucedida, a mensagem é enviada ao módulo servidor que re-encaminhará a mensagem ao destinatário desejado.

3.3.2 OPERACIONALIDADE DA IMPLEMENTAÇÃO

O protótipo deste trabalho encontra-se dividido em dois módulos, cliente e servidor. O protótipo funciona tanto em redes internas como na Internet sem exigir muita banda de conexão (podendo-se utilizá-lo perfeitamente em *modems* de 33kbps) ou máquinas rápidas (podendo-se executá-lo em computadores *Pentium* 150Mhz) para sua execução. Entretanto recomenda-se que o módulo servidor seja executado em computadores mais recentes e tenha acesso rápido à Internet caso deseje-se empregar o protótipo para quantidades superiores a cem clientes. A seguir será demonstrado o funcionamento dos dois módulos sendo executados em um computador pessoal sobre a plataforma Windows XP.

3.3.2.1 MÓDULO SERVIDOR

O módulo servidor atua somente com objetivo de gerenciar os clientes e não tem qualquer interação direta com o usuário além das operações básicas de ativar ou desativar o servidor.

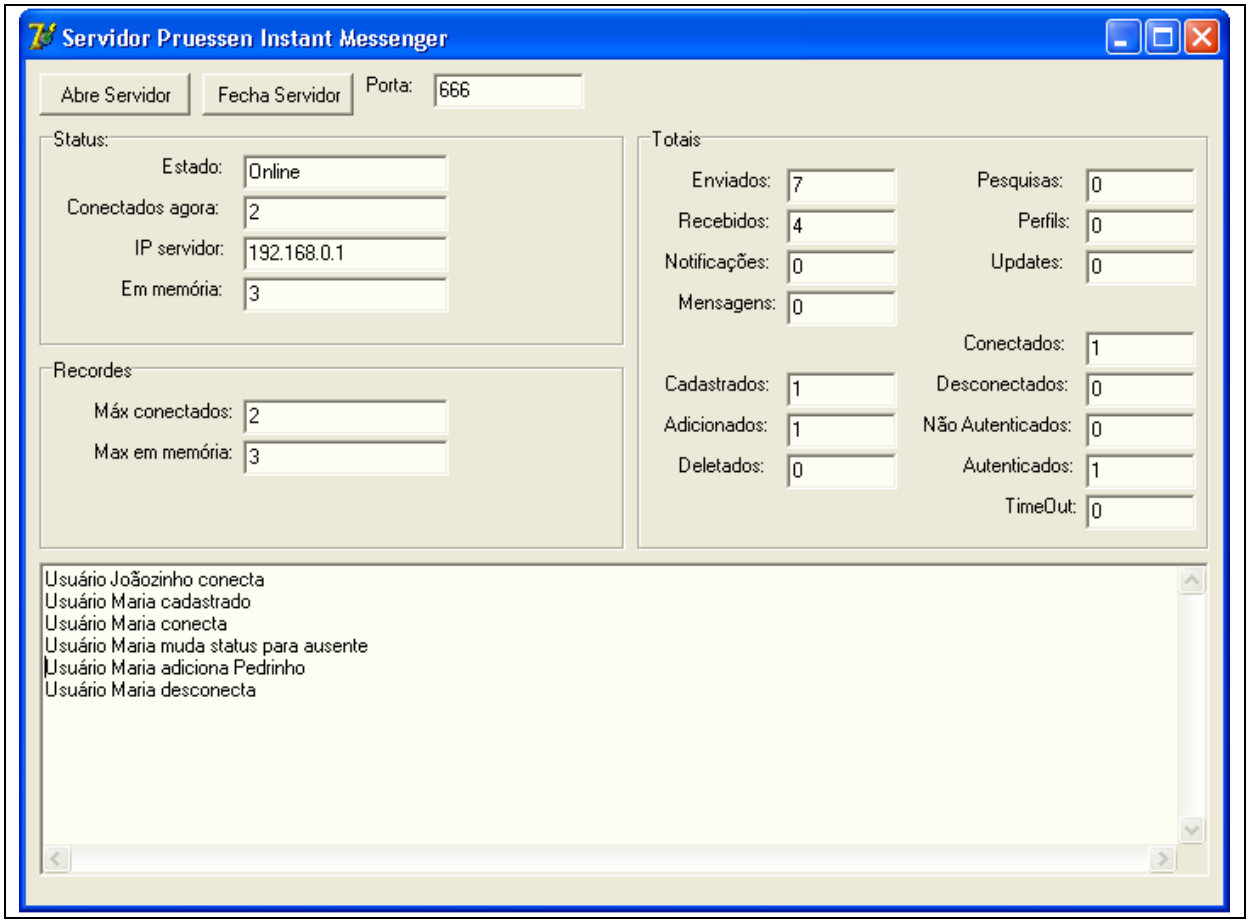


Figura 42 – Tela Principal do módulo servidor

Na figura 42 encontra-se a tela principal do módulo servidor. A seguir são relatadas as informações dispostas pelo módulo com suas respectivas descrições:

a) estado: estado o qual o servidor encontra-se no momento atual;
b) clientes conectados: quantidade de usuário “logados” ao sistema no momento corrente;
c) IP servidor: endereço IP da máquina ao qual o servidor está sendo executado;
d) usuários em memória: quantidade de instâncias de clientes na memória;
e) conectados: recorde de usuários “logados” ao sistema ao mesmo tempo;
f) em memória: quantidade recorde de instâncias de clientes na memória ao mesmo tempo;
g) enviados: quantidade de mensagens enviadas pelo servidor;
h) recebidos: quantidade de mensagens recebidas pelo servidor;
i) notificações: quantidade de notificações enviadas pelo servidor a clientes;

j) mensagens: quantidade de mensagens enviadas através do servidor de um cliente à outro;

k) cadastrados: quantidade de cadastros ocorridas;

l) adicionados: quantidade de contatos adicionados à listas de contatos.

Na figura 42 pode-se observar os usuários Joãozinho e Maria utilizando o módulo cliente, que diretamente age sobre o módulo servidor.

3.3.2.2 MÓDULO CLIENTE

Esta seção descreve a operacionalidade do módulo cliente, o qual permite aos usuários acesso às funcionalidades do protótipo.

Figura 43 – Tela principal do módulo cliente

Ao inicializar a aplicação, o usuário depara-se-á com uma janela similar a mostrada na figura 43. Na parte central encontra-se a lista de contatos do usuário, em um primeiro momento estará vazia sendo carregada com contatos a partir do instante em que o usuário

conectar-se ao módulo servidor, assim recebendo sua relação de contatos caso possua algum. Caso o usuário deseje ter outro contato que ainda não se encontre na sua lista, é possível utilizar-se do botão “Adicionar usuário”.

No canto inferior direito localiza-se um *combo-box* destinado a mudança de status do usuário, também é utilizado para a chamada da tela de conexão ao servidor caso o aplicativo ainda não se encontre “logado”.

Ao lado do *combo-box* de status, no canto esquerdo inferior, encontra-se o botão nomeado de “Spread Chat”. Esta opção refere-se a funcionalidade de *chat* que utiliza um servidor Spread para a disseminação dos dados.

No botão “Cadastrar Conta” o usuário poderá cadastrar uma nova conta de usuário no servidor, esta conta cadastrada possibilitará que o usuário realize o *login* no módulo servidor do sistema.

O botão titulado de “Perfil” permite que o usuário modifique seu perfil, o qual permanece cadastrado no banco de dados acessado pelo módulo servidor e pode ser consultado por outros usuários. É a partir destas informações preenchidas no perfil que são realizadas pesquisar por outros usuários do sistema.

Nomeado de “Procurar usuário”, este botão acessa a tela relacionada às funcionalidades de busca por usuários através de algumas informações que são utilizadas como filtro da pesquisa.

O botão “Sair” desconecta o usuário caso esteja “logado” no servidor e encerra o módulo cliente do protótipo. No botão “Sobre PIM” é apresentado um pequeno resumo sobre a aplicação.

Cada contato apresentado na lista de contatos é representado por um apelido e uma imagem que representa seu status atual. Quando clicado com o botão direito sobre um destes contatos, visualiza-se um menu do tipo *pop-up* idêntico ao apresentado na figura 44, contendo no topo o apelido e o identificador do contato ao qual o menu se refere. A seguir são relatadas as funções encontradas neste menu com uma breve descrição:

a) Enviar Mensagem: acessa a tela de envio de mensagens;
b) Enviar Arquivo: acessa a tela para envio de arquivos;

c) Verificar Perfil: exibe o perfil do contato;
d) Deletar Contato: elimina o contato da lista de amigos do usuário.

Na figura 43 pode-se observar a lista de contatos do usuário Maria, a qual possui na sua lista de contatos os usuários Joãozinho e Pedrinho.

Figura 44 – Menu com funcionalidades de um contato

Na figura 45 é demonstrada a tela de cadastro de uma nova conta. O usuário deve preenche os campos: “NickName”, “Senha” e “Confirmar senha”. Tem-se ainda opção de um cadastro mais detalhado através do botão “Detalhes”, estes dados também poderão ser preenchidos ou alterados futuramente. Após a confirmação no botão “Próximo ->”, os dados são enviados ao servidor e em caso de sucesso na operação, uma chave de identificação será apresentada ao usuário, assim possibilitando o *login* no servidor através deste identificador e da senha fornecida durante a tela de cadastro.

Os campos encontrados na tela de cadastro são:

a) Nome: nome do usuário;
b) Sobrenome: sobrenome do usuário;
c) NickName: apelido pelo qual o usuário deseja ser apresentado na lista de contatos de outros usuários;
d) Email: endereço de e-mail do usuário;
e) Senha: senha para a autenticação do cliente no módulo servidor;
f) Confirmar senha: precaução para que o usuário realmente conheça a senha que esta fornecendo;
g) Comentário: comentário que o usuário deseja apresentar no seu perfil.

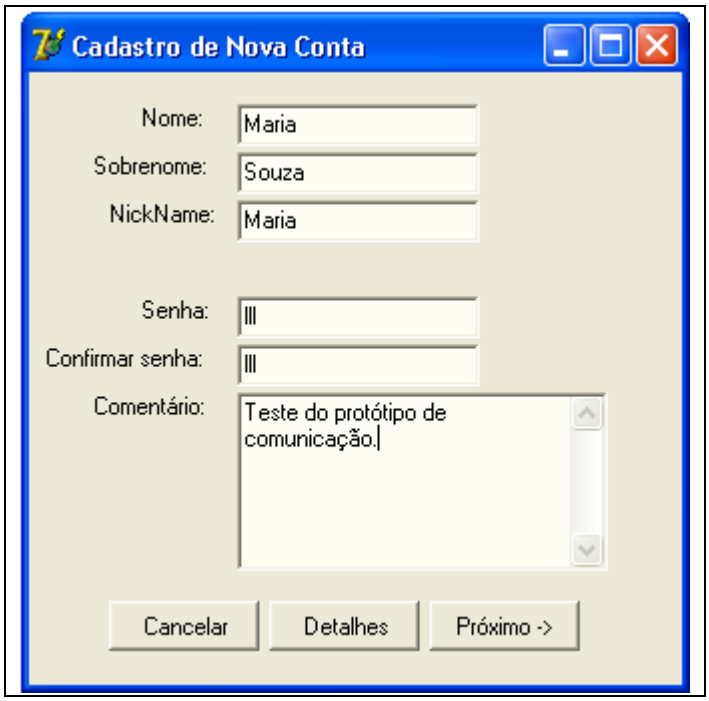


Figura 45 – Tela de cadastro de nova conta

Conforme pode-se observar na figura 45, o usuário Maria esta se cadastrando no sistema, neste momento pode-se optar em cadastrar mais detalhes ou manter um cadastro mais simplificado e rápido.

Figura 46 – Tela de login

Na figura 46 é representada a tela de conexão do usuário ao sistema, esta tela é apresentada toda vez que o status for modificado para outra opção que não a *offline*. Nesta tela o usuário deve informar seu código de identificação que lhe foi informado após um cadastro, sua senha e o endereço onde se encontra em execução o módulo servidor. Ao confirmar no botão “Conectar” é iniciado o processo de tentativa de conexão e autenticação do usuário no

sistema, caso a operação seja bem sucedida será liberado acesso às demais funcionalidades do módulo cliente e será recebida a lista de contatos do usuário.

Pode-se observar na figura 46 o usuário Maria conectando-se ao sistema, para isto, ela informa o seu identificador e sua senha, neste exemplo utiliza-se o IP “127.0.0.1” pois o módulo servidor encontra-se em execução no próprio computador local que a Maria usa o módulo cliente.

A figura 47 representa a tela mostrada ao usuário quando o mesmo deseja pesquisar por outros usuários que estão cadastrados no sistema, não necessariamente *online*. Nesta tela o usuário pode preencher os campos que deseja utilizar como filtro da pesquisa, depois de confirmar no botão “Pesquisar” a requisição é enviada ao módulo servidor que consultará o banco de dados devolvendo ao usuário o resultado da pesquisa.

Os campos que o usuário pode preencher para utilizar como filtros são:

a) NickName: apelido do usuário;
b) Nome: nome do usuário;
c) Sobrenome: sobrenome do usuário;
d) Idade: idade do usuário;
e) Email: Endereço de e-mail do usuário;
f) Cidade: cidade em que reside o usuário;
g) Estado: estado em que reside o usuário;
h) País: país em que reside o usuário;
i) Empresa: empresa no qual o usuário trabalha;
j) Cargo: cargo ocupado profissionalmente pelo usuário;
k) Área: área de trabalho que o usuário atua;
l) Sexo: sexo do usuário.

Conforme pode-se observar na figura 47, um usuário procura por Maria, a qual cadastrou-se anteriormente.

Após o módulo servidor retornar o resultado da pesquisa, é exibida uma tela como a apresentada na figura 48, listando os usuários encontrados, nesta tela é possível enviar mensagem e adicionar à lista de contatos um dos usuários encontrado.

Figura 47 – Tela de pesquisa de usuários

Na figura 48 é mostrado o usuário Maria que foi encontrada na pesquisa demonstrada na figura 47.

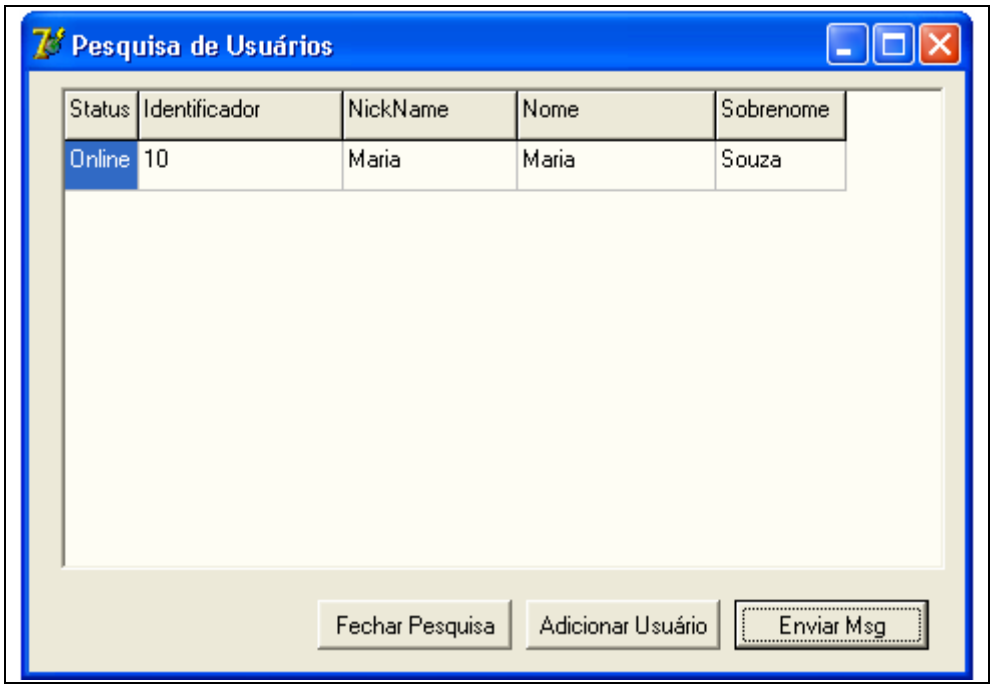


Figura 48 – Tela de resultados de uma pesquisa por usuários

A tela de perfil apresentada na figura 49 pode ser chamada em algumas circunstâncias no módulo cliente. Esta tela apresenta o perfil do usuário corrente ou de um outro usuário do sistema, quando visualizado o perfil do próprio usuário que se encontra “logado” no aplicativo, é permitido alterações no seu perfil.

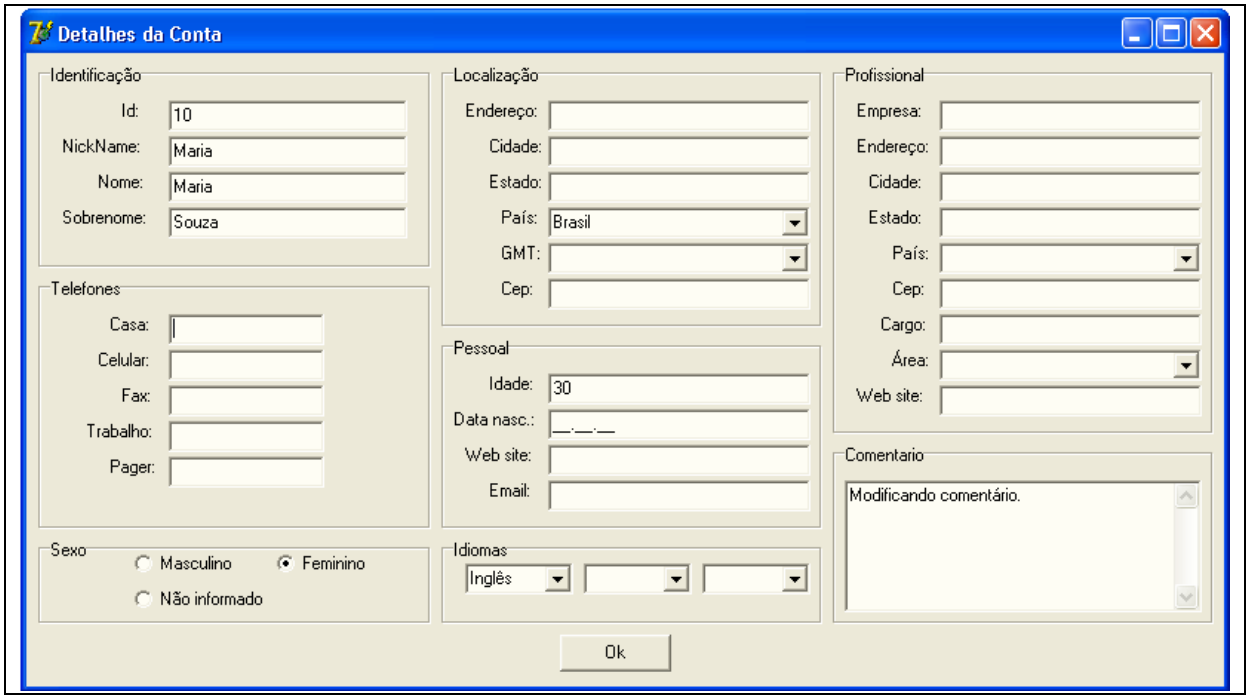


Figura 49 – Tela de perfil de usuário

A alteração do perfil do usuário Maria pode ser observada na figura 49, onde foram preenchidos alguns campos e o campo de comentário foi modificado.

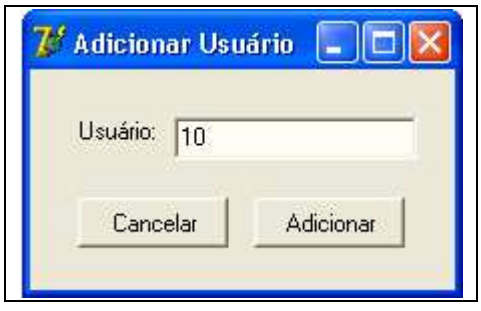


Figura 50 – Tela de adição de usuário na lista de contatos

Na figura 50 é representada a tela utilizada para adicionar um novo usuário à lista de contatos, nesta caso específico pode-se observar alguém adicionando o usuário 10 a sua lista de contatos. Esta tela é unicamente de um campo “Usuário”, no qual deve-se informar o

código de identificação do usuário que deseja-se adicionar à lista de contatos. Depois de confirmado no botão “Adicionar”, a requisição é enviada ao servidor, caso não ocorra problemas, o novo usuário aparecerá com seu status e apelido na lista de contatos.

Ao entrar na opção “Chat Spread” encontrada na tela principal do módulo cliente, é apresentada ao usuário uma tela similar à encontrada na figura 51. Esta tela possui um campo para que o usuário informe a sala de *chat* que deseja integrar e o servidor no qual encontra-se o servidor Spread, este endereço é fornecido no formato: porta @ endereço IP. Exemplo: “1234@192.160.0.5”.

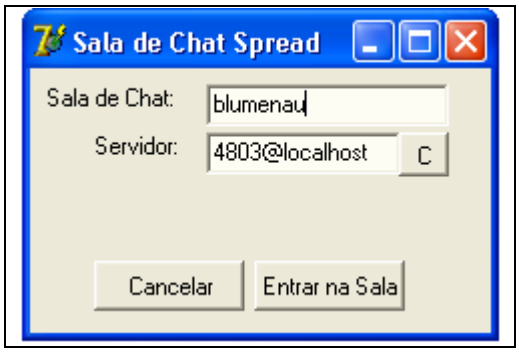


Figura 51 – Tela de acesso aos recursos do mecanismo Spread

Ao lado do campo de endereço do servidor Spread encontra-se um botão responsável por conectar ou desconectar o mecanismo Spread. Quando conectado, é possível através do botão “Entrar na Sala” integrar uma sala de *chat*, esta sala comporta-se internamente como um grupo de usuários do mecanismo Spread e é gerenciado pelo mesmo. Não há um número limite de salas que pode-se participar simultaneamente. Para cada sala é criada uma nova tela de “Sala de Chat” no módulo cliente, conforme a figura 51. Para cada nova sala de *chat* é criado um grupo no servidor Spread, caso ainda não existente.

Na figura 51 pode-se observar um usuário conectando-se ao mecanismo Spread para entrar na sala de *chat* “blumenau" e na figura 52 o usuário já se encontra integrado a sala de *chat*.

A tela para envio de mensagens a outro usuário pode ser acessada através de três modos: com um clique duplo em um usuário da lista de contatos; através do menu *pop-up* de um usuário da lista de contatos; e a opção de enviar mensagem encontrada na tela resultante

de uma pesquisa por usuários. Esta tela pode ser visualizada na figura 53, onde o usuário Joãozinho recebeu uma mensagem da Maria, a quem encontra-se respondendo.

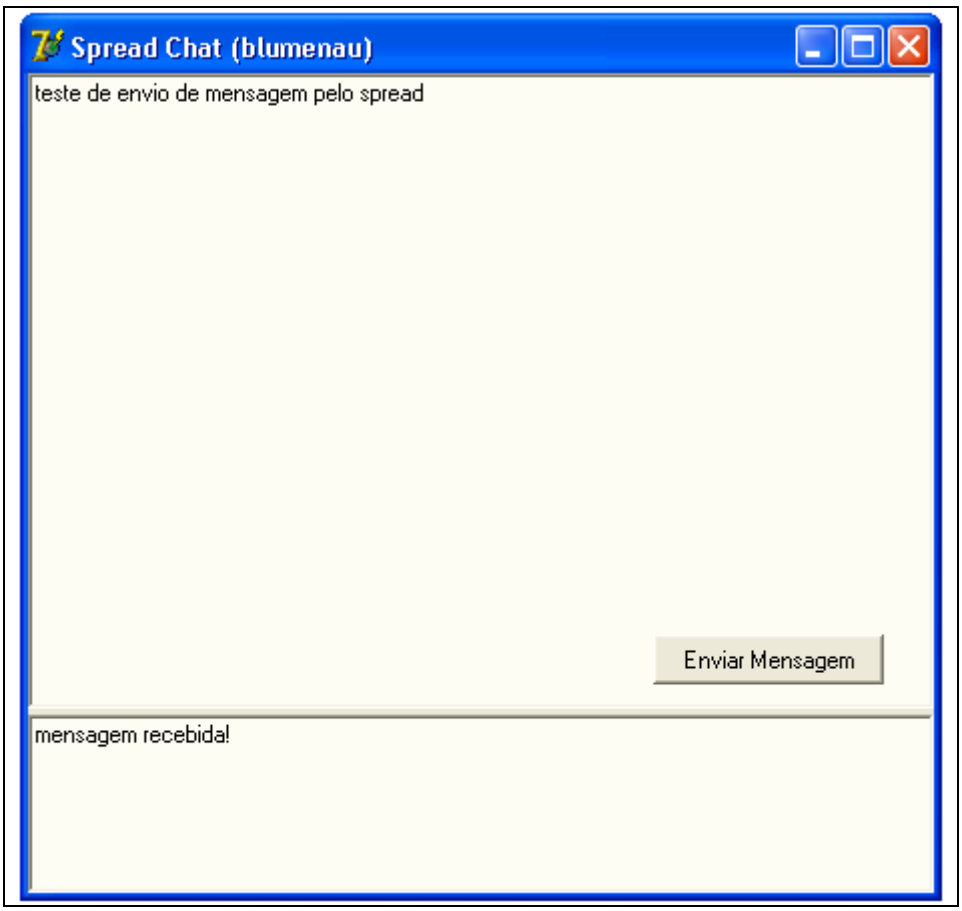


Figura 52 – Tela de chat através do mecanismo Spread

Quando alguma mensagem for recebida de um outro usuário, um alerta será mostrado ao cliente e a mensagem será exibida através desta tela. São encontrados dois botões nesta tela, um responsável pelo envio de uma mensagem que pode ser inserida na parte inferior desta janela e outro que chama a tela de envio de arquivos. Ambas as funções destinadas ao contato que a janela se refere.

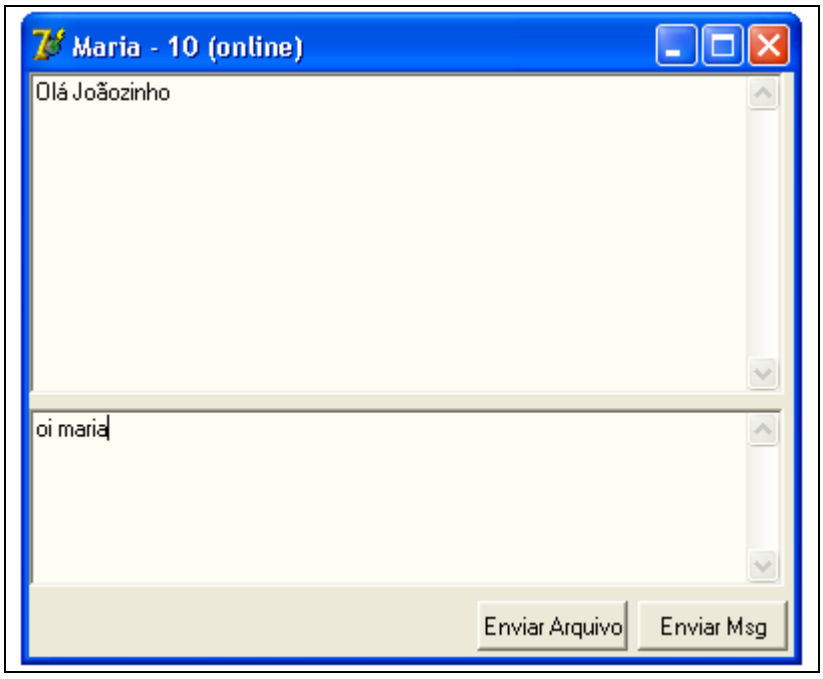


Figura 53 – Tela de envio e recebimento de mensagens

A figura 54 mostra a tela usada para enviar um convite de transferência de arquivo a outro contato da lista. O usuário pode informar o endereço do arquivo no campo “Arquivo” ou clicar em “Selecionar Arquivo” e navegar até o arquivo desejado selecionando-o. No campo “Para” é informado o apelido do usuário que receberá o arquivo, no campo “IP” é informado o IP que o receptor do arquivo encontra-se. Após a confirmação do envio através do botão “Enviar” é mostrado ao usuário uma janela com o *status* da transferência do arquivo, conforme vista na figura 55, podendo cancelar a transferência se desejar.

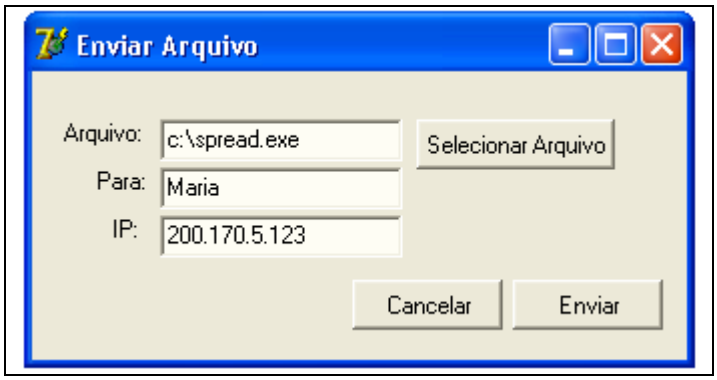


Figura 54 – Tela de envio de arquivo

Na figura 54 pode-se observar um usuário tentando enviar o arquivo “spread.exe” para a Maria, a qual encontra-se conectada no IP “200.170.5.123”. Após o usuário Maria aceitar é visualizada a transferência do arquivo conforme a figura 55.

Figura 55 – Tela de transferência de arquivo

Na figura 56 é apresentado o convite para receber um arquivo enviado por outro usuário. Nesta tela é possível optar por aceitar ou rejeitar o arquivo oferecido, quando aceito é mostrado o *status* da transferência.

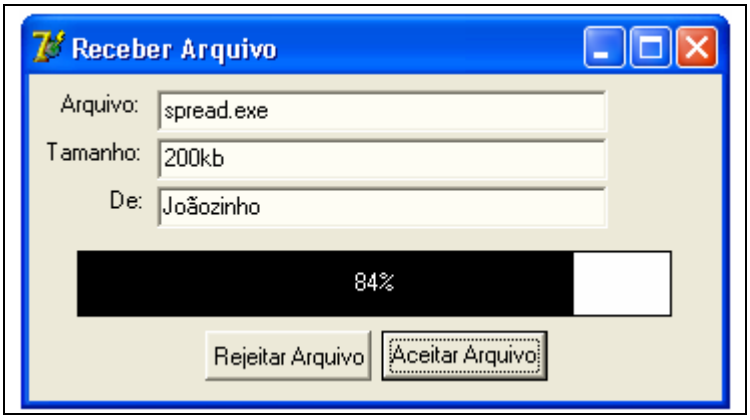


Figura 56 – Tela de oferta de arquivo

Pode-se observar na figura 56 a tela mostrada ao usuário Maria quando lhe foi enviada um convite de recebimento de arquivo, o qual é enviado pelo usuário Pedrinho. A transferência de arquivo foi aceita e encontra-se com 84% já transferido.

3.4 RESULTADOS E DISCUSSÃO

O protótipo apresentado neste trabalho de conclusão de curso não encontra-se no mesmo nível dos programas desenvolvidos por grandes empresas como o ICQ, Yahoo Messenger e o MSN Messenger, entretanto se mantém funcionando de forma estável e realizando bem suas tarefas.

O mecanismo de comunicação em grupo Spread permite uma série de facilidades como ordenação e garantia de entrega das mensagens. Uma desvantagem encontrada durante a implementação relacionada ao Spread é a necessidade da especificação dos possíveis clientes anteriormente à execução do servidor, não sendo possível à inclusão de um novo membro ao mecanismo em tempo de execução caso este não tenha sido previamente incluído nos arquivos de configuração do servidor. Outra limitação é a falta de flexibilidade no servidor do Spread, impedindo tratamentos nas mensagens trafegadas pelo servidor como por exemplo a adição de controles extras nas mensagens. Devido a estas restrições observou-se que seria mais indicado o uso do mecanismo nas funcionalidades referentes à parte das salas de *chat*, onde foi muito bem empregado, entretanto seria mais interessante a implementação por *sockets* nas demais operacionalidades (envio de mensagens, envio de arquivos, alteração de status, autenticação de usuário, pesquisa por usuários e acessos ao módulo servidor em geral), e desta forma foi feito.

A implementação através do uso de *sockets* infelizmente dificultou em muito a implementação, todavia, ampliou largamente a flexibilidade quanto a mudanças e personalizações no protótipo. Em momento algum pode-se observar perda de mensagens, desordenamento na seqüência ou qualquer outra irregularidade durante o uso do Spread, o que comprova sua eficiência quanto a confiabilidade e ordenamento na entrega de mensagens.

Contudo o mecanismo Spread simplificou em muito a implementação da parte referente às salas de *chat*, permitindo que esta funcionalidade obtivesse excelente confiabilidade, segurança e velocidade, coisas que certamente não se alcançaria utilizando-se da implementação convencional sem o uso de um mecanismo de comunicação em grupo. Pode-se observar que o Spread destina-se mais à replicação de dados entre máquinas em ambientes fechados como LANs onde se conhece os clientes do mecanismo.

Como resultado geral do trabalho obteve-se um protótipo que permite a comunicação entre dois ou mais usuários, além de permitir uma série de funcionalidades.

4 CONCLUSÕES

Com o desenvolvimento deste trabalho foi possível aplicar grande parte dos conhecimentos adquiridos durante o curso e em especial observar a complexidade envolvida em implementar um aplicativo utilizando o paradigma cliente/servidor.

Foi possível ampliar os conhecimentos relacionados ao uso do P2P quando implementou-se a transmissão de mensagens e arquivos de forma direta entre usuários, sem necessitar consumir recursos do módulo servidor.

Os objetivos propostos para este trabalho foram alcançados. A troca de mensagens através do servidor pode ocorrer de forma P2P com os demais clientes. O módulo servidor gerencia seus clientes de forma estável e otimizada, trabalhando com boa parte dos dados de forma dinâmica reduzindo o número de acessos ao disco. A troca de arquivos ocorre de forma eficiente, onde emprega-se um componente que compacta os pacotes de dados, desta forma acelerando a transferência. O cadastro pelo próprio módulo cliente possibilita que um usuário novo possa registrar-se a partir da própria aplicação que utilizará. A pesquisa por clientes cadastrados no sistema facilita a localização de pessoas conhecidas e dispensa a necessidade de memorizar os identificadores numéricos.

Há uma grande quantidade de detalhes que devem ser tratados para que se possa realmente utilizar o protótipo na prática, entre eles tratamentos quanto à segurança do sistema. Contudo, o protótipo estabelece uma base para futuras implementações podendo-se chegar a um nível comercial.

Por fim, o desenvolvimento deste trabalho foi de grande valia, possibilitando a aquisição de novos conhecimentos, principalmente na área de sistemas distribuídos, bem como empregando vários dos conceitos lecionados no curso.

4.1 EXTENSÕES

Como extensão a este protótipo sugere-se um melhor controle na parte da segurança do sistema, desta forma habilitando-o para uso comercial.

Pode-se também adicionar novos recursos como uso de *webcams*, conversa por voz, melhor interface gráfica e talvez uma possível compilação do protótipo no Borland Kylix, expandindo o uso do aplicativo a plataformas Linux.

Vislumbra-se ainda a possibilidade da integração de novos recursos relacionados com a Universidade e a biblioteca, permitindo serviços como por exemplo aviso de multa sobre atraso de um livro, consulta de notas, horário de aulas.

REFERÊNCIAS BIBLIOGRÁFICAS

BORLAND. **Borland Delphi.** USA, 2004. Disponível em: <http://www.borland.com/delphi/>. Acesso em: nov. 2004.

PUCRS, **Comunicação entre processo.** Porto Alegre, 2002. Disponível em: <http://www.inf.pucrs.br/~fldotti/sod/comgrupo.pdf> Acesso em: 30 abr. 2004.

COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim. **Distributed systems**: concepts and design. 3. ed. Harlow: Addison-Wesley, 2001.

FIGUEIREDO, José Otavio Coutinho. **Comunicação de grupo confiável em tempo-real para o sistema operacional RT-Linux**. Porto Alegre, 1999. Disponível em: <http://www.inf.ufrgs.br/pos/SemanaAcademica/Semana99/joseotavio/joseotavio.html>. Acesso em: 10 maio 2004.

FLEISCH, Brett D. **Research projects**. Riverside, USA, 2004. Disponível em: <http://www.cs.ucr.edu/~brett/research.html>. Acesso em: 10 dez. 2004.

FURLAN, Jose Davi. **Modelagem de objetos através da UML**: The unified modeling language. São Paulo: Makron Books, 1998. 329p.

GOULART, Ademir. **Avaliação de mecanismos de comunicação em grupo para ambientes WAN**. 2002. 169f. Dissertação (Mestrado em Ciências da Computação) - Universidade Federal de Santa Catarina, Florianópolis.

GUERRAOUI, Rachid; SCHIPER, André. **Fault tolerance by replication in distributed systems**. Lausanne, Suíça, 1996. Disponível em: <http://lsewww.epfl.ch/Documents/acrobat/GS96e.pdf >. Acesso em: 20 abr. 2004.

ICQ INSTANT MESSENGER. **The Community.** Atlanta, USA, 2004. Disponível em: <http://company.icq.com/info/icqstory.html>. Acesso em: 10 nov. 2004.

INFOWESTER. **Banco de dados PostgreSQL e MySQL.** Brasil, 2004. Disponível em: <http://www.infowester.com/postgremysql.php>. Acesso em: 10 nov. 2004.

JOHNS HOPKINS UNIVERSITY. **Wackamole**: use your resources. Baltimore, USA, 2004. Disponível em: <http://www.backhand.org/wackamole/>. Acesso em: 10 dez. 2004.

MARTINS, Luciano; MOREIRA, Edson dos Santos. **Uso do protocolo IPv6 e de multicasting para transmissão de vídeo**. São Carlos, São Paulo, 2001. Disponível em: <http://www.rnp.br/wrnp2/2001/palestras_engenharia/res_engen_13.pdf>. Acesso em: 10 out. 2004.

MICROSOFT CORPORATION. **Msn Messenger.** São Paulo, 2004. Disponível em: <http://messenger.msn.com>. Acesso em: 10 nov. 2004.

MOSER, L. E. et al. **Totem**: A fault-tolerant multicast group communication system. In: COMMUNICATIONS OF THE ACM, v. 39, n. 4, abr. 1996. p. 54-63.

MYSQL AB. **MySQL**: the world's most popular open source database. USA, 2004. Disponível em: <http://www.mysql.com/>. Acesso em: 10 nov. 2004.

OLIVEIRA, Etienne. **Arquitetura TCP/IP**. UNIGRANRIO – Universidade do Grande Rio Redes de Computadores II, ago. 2002.

RATIONAL. **IBM Rational Software.** USA, 2004. Disponível em: <http://www-306.ibm.com/software/rational/>. Acesso em: 20 nov. 2004.

STANTON, Jonathan. **A users guide to Spread**. Baltimore, USA, 2002. Disponível em: <http://www.spread.org/docs/guide/>. Acesso em: 10 ago. 2004.

STANTON, Jonathan; AMIR Yair. **The Spread wide area group communication system**. Baltimore, USA, 1998. Disponível em: <www.cnds.jhu.edu/pub/papers/spread.pdf>. Acesso em: 10 ago. 2004.

SYBASE. **Sybase PowerDesigner.** USA, 2004. Disponível em: <http://www.sybase.com/products/developmentintegration/powerdesigner>. Acesso em: nov. 2004.

TANEMBAUM, A. **Sistemas operacionais modernos**. Rio de Janeiro: Prentice Hall do Brasil, 1995.

THE SPREAD GROUP COMMUNICATION TOOLKIT, **The Spread tookit**. Baltimore, USA, 2004. Disponível em: <http://www.spread.org/>. Acesso em: 15 ago. 2004.

YAHOO, **Yahoo! Messenger.** USA, 2004. Disponível em: <http://br.download.yahoo.com/messenger/>. Acesso em: 20 nov. 2004.