Universidade Federal da Paraíba

Centro de Ciências e Tecnologia

Coordenação de Pós-Graduação em Engenharia Elétrica

NetComm

Uma ferramenta para o desenvolvimento de aplicações multimídia distribuídas e gerência de redes em ambiente Windows

Marcos Sebastian Alsina

Campina Grande - PB

Setembro, 1994

Marcos Sebastian Alsina

NetComm

Uma ferramenta para o desenvolvimento de aplicações multimídia distribuídas e gerência de redes em ambiente Windows

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

Jacques Phillipe Sauvé

(*Orientador*)

Campina Grande - PB

Setembro, 1994

A461n Alsina, Marcos Sebastian.
NetComm : uma ferramenta para o desenvolvimento de aplicações multimídia distribuídas e gerência de redes em ambiente windows / Marcos Sebastian Alsina. - Campina Grande, 1994.
109 f.

Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal da Paraíba, Centro de Ciências e Tecnologia, 1994.
"Orientação : Prof. Dr. Jacques Phillipe Sauvé".
Referências.

1. Engenharia Elétrica. 2. Multimídia. 3. Redes de Computadores. 4. Dissertação - Engenharia Elétrica. I. Sauvé, Jacques Phillipe. II. Universidade Federal da Paraíba - Campina Grande (PB). III. Título

CDU 621:004.032.6(043)

NetComm
UMA FERRAMENTA PARA O DESENVOLVIMENTO DE APLICAÇÕES MULTIMÍDIA DISTRIBUÍDAS E GERÊNCIA DE REDES EM AMBIENTE WINDOWS

MARCOS SEBASTIAN ALSINA

Dissertação Aprovada em 26.08.1994

JACQUES PHILIPPE SAUVÉ, Ph.D., UFPB
Orientador

MARIA DE FÁTIMA QUEIROZ VIEIRA TURNELL, Ph.D., UFPB
Componente da Banca

EDILSON FERNEDA, Dr., UFPB
Componente da Banca

CAMPINA GRANDE - PB
Agosto - 1994

Agradecimentos

Muitas pessoas contribuíram para a realização deste trabalho. A essas pessoas externo o meu agradecimento, e especialmente:

- A minha mãe pelo amor e pelos anos dedicados à minha educação.

- A meu pai que não chegou a ver este trabalho concluído.

- A minha esposa pelo carinho, amor e compreensão nas horas que investi frente ao computador poucos dias após nosso casamento.

- A meu orientador Jacques Philippe Sauvé por ter confiado em mim, por ter-me ajudado a concluir este trabalho, pela dedicação ao longo dos anos de convívio, por compartilhar seus conhecimentos comigo e por sua amizade.

- A INFOCON inicialmente e a FAST posteriormente por possibilitarem a execução deste trabalho.

- A Adriano Sérgio pela amizade e por sua ajuda essencial na fase de implementação do trabalho.

- A Alessandro Jatobá por sua inestimável colaboração.

- A Walfredo Cirne Filho por compartilhar seus conhecimentos e por sua ajuda nos momentos difíceis.

- A Gedeon Santos Filho pela amizade e poder de persuasão que me deram ânimo para a conclusão deste trabalho.

- Aos funcionários da FAST.

- A cafeína e aos litros de café que foram responsáveis pelas melhores idéias e as Pizas que saciaram a fome. Aos neurônios queimados.

- A cerveja, vodka, vinho e todas as coisas boas da vida e as pessoas que delas partilham: Léo, Arycesar, Adriano, Tárcio, Jacques, Gedeon, Walfredo, Dalmer, Coriolano, Katyusko, Raul, Johnson, ...

- A todas as forças da natureza e do universo, ao mais ínfimo grão de areia, a força dos mortos, ao poder dos vivos, a todos que desejam um mundo melhor, ao computador, ao ar que respiro, a felicidade e a tristeza, ao mais sábio e ao mais humilde dos homens, as estrelas, ao mar, ao entardecer e ao amanhecer, enfim, a Deus.

Resumo

Multimídia, computação distribuída, interfaces gráficas e gerência de redes são assuntos ditos estado-da-arte na informática dos anos 90. Este trabalho descreve o projeto e implementação de uma ferramenta para o desenvolvimento de aplicações multimídia distribuídas e para a gerência de redes locais de computadores fazendo uso de eventos distribuídos sob ambiente Windows.

Abstract

Multimedia, distributed computing, graphic user interfaces and network management are topics "state of art" in the 90's computing. This work describes the design and the coding stages of a tool for the development of distributed multimedia applications and for LANs (Local Area Networks) management by using distributed events in a Windows Environment.

Índice Geral

Índice de Figuras e Tabelas

Capítulo 1 - Introdução

Os computadores ao longo dos anos têm se mostrado um excelente ferramental nas tarefas mais diversas. Na década de 60, os computadores eram utilizados como grandes calculadoras. Na década de 70 eles eram usados principalmente como fonte de acesso a bases de dados com terminais de consulta tipo reserva de passagens aéreas. A década de 80 foi da micro-informática, os computadores ganharam conotação pessoal e diversificaram seu uso, passando a serem usados no processamento e armazenamento de informação, no controle de máquinas, na diversão, no escritório entre tantos outros. A década de 90 é a década da comunicação. Os computadores estão sendo usados para substituir os meios tradicionais de comunicação; diga-se: fax, telefone, televisão, vídeo, música, entre outros. Nesse contexto, o *software* têm extrema importância. Ele é o responsável por garantir fácil acesso às novas dimensões de uso dos computadores relacionados a comunicação.

1.1 Direções do *Software*

O *hardware* evoluiu muito rapidamente nos últimos anos. O *software* é a interface entre o homem e a máquina e o principal alavancador das novas tecnologias de *hardware*. Nas próximas seções nos deteremos nas principais direções tomadas pelo *software* nos últimos anos: ambientes gráficos, multimídia, computação distribuída, *software* para grupos de trabalho e gerência de redes de computadores.

1.1.1 Ambientes gráficos

A forma com que o usuário interage com o computador evoluiu muito com o advento das interface gráficas. O usuário passou a comandar o computador através da manipulação direta de objetos gráficos ao invés de descrever textualmente a instrução desejada. Assim, para executar a aplicação de sua escolha, o usuário passou a clicar sobre ícones ao invés de ativá-la pelo seu nome.

O *hardware* teve papel fundamental na evolução do *software* para o uso de interfaces gráficas. No início dos anos 80 um PC não tinha nem memória nem velocidade suficientes para tal tarefa. Nos meados da década de 80 porém isso se tornou uma realidade e surgiram as primeiras interfaces gráficas como a do Macintosh, Windows, Open Look, Motif entre outras. O Windows foi o grande responsável pela popularização dessa nova tecnologia sendo hoje um dos *softwares* mais vendidos do mundo; são vendidas mensalmente mais de um milhão de cópias do Windows.

As interfaces gráficas tornaram mais agradável o uso do computador e permitiram que pessoas leigas tivessem acesso ao mesmo com pouco ou quase nenhum treinamento. Tendo em vista esse processo de massificação no uso de interfaces gráficas propiciado pelo Windows, a indústria de *software* passou a incorporar essa tecnologia. Hoje a maior parte do *software* desenvolvido para PCs é para o ambiente Windows.

1.1.2 Multimídia

Multimídia é uma das direções que o *software* tomou nos anos noventa. A capacidade de armazenar e manipular sons, imagens de vídeo, animações e textos modificou a maneira de enxergar a informação. Uma enciclopédia impressa em papel é uma coisa; em CD-ROM essa mesma enciclopédia pode incorporar imagens e sons. O volume de informação armazenada nesses casos é enorme; a evolução dos dispositivos ópticos de armazenamento possibilitou essa mudança. Ao dispor da informação em meios digitais mudou também a forma de acessá-la. Novos métodos de indexação da informação foram desenvolvidos. Para pesquisar bases de dados textuais foram desenvolvidos métodos de pesquisa textual por proximidade ou similaridade de palavras, por exemplo. No caso de imagens e sons, métodos de recuperação de informação por similaridade de padrões têm sido utilizados e no caso específico de imagens por similaridade de cores também.

O barateamento do *hardware* necessário têm viabilizado a massificação da multimídia, passando a fazer parte do dia a dia da população. Hoje essa tecnologia está disponível além de em um grande número de PCs, em terminais de auto-atendimento bancários, em grandes shopping centers, supermercados, bibliotecas, entre outros. Vislumbrando um grande mercado emergente as empresas de *hardware* começaram a oferecer pacotes que incluem os dispositivos

multimídia mais comuns: CD-ROM, placa de som, microfone e alto-falantes. As empresas de *software*, por sua vez, lançam cada vez mais produtos que incorporam recursos multimídia.

A multimídia, em consonância com o desenvolvimento dos meios de comunicação como a telefonia celular e das redes públicas de alta velocidade, possibilitou a disseminação de novas aplicações do computador. Hoje com um computador portátil e um telefone celular é possível receber e transmitir fax para qualquer parte do mundo e brevemente o computador poderá ser utilizado como video-fone.

1.1.3 Computação distribuída

Com a micro-informática e as redes de computadores, a informação deixou de ser armazenada de forma centralizada passando a ser distribuída. Os processos de gerenciamento e manipulação da informação passaram a ser também distribuídos.

As redes de computadores estão hoje presentes na maioria das organizações sejam elas de nível local ou mundial. A maior rede de computadores do mundo, a Internet, cresce em proporções cada vez maiores e congrega entidades públicas e privadas da maioria dos países do mundo.

O *software* acompanhou o crescimento das redes de computadores. O paradigma mais utilizado no desenvolvimento de aplicações distribuídas é o cliente-servidor. Nesse paradigma existem dois processos: um chamado servidor e outro cliente. O servidor oferece serviços e atende a pedidos dos clientes. A maioria dos bancos de dados hoje em dia por exemplo, é um exemplo de aplicação cliente-servidor. O banco de dados em si é o servidor e as aplicações que acessam a base de dados para consultas ou atualizações são os clientes. No caso específico de bancos de dados, a evolução para a arquitetura cliente-servidor aliado a evolução das interfaces gráficas fez surgir um grande número de ferramentas de desenvolvimento rápido de aplicações cliente-servidor como SQL-Windows, Magic e Power Builder entre outras. Maiores informações sobre a arquitetura cliente-servidor podem ser vistam em [SANTOS93].

1.1.4 *Groupware*

A massificação das redes locais de computadores fez surgir uma nova categoria de *software* denominado *groupware* ou "*software* para grupos de trabalho". A idéia é usar o suporte das redes locais e preferencialmente, mas não obrigatoriamente, das interfaces gráficas para proporcionar ferramentas de *software* para grupos de usuários.

Inicialmente, com o advento das interfaces gráficas, o *software* era usado como ferramenta pessoal. Os dados gerados por Planilhas de cálculo, agendas eletrônicas, processadores de texto entre tantas outras aplicações não eram compartilhados pelos usuários da rede. O *software* para grupos de trabalho deu nova dinâmica às redes de computadores. Por exemplo, a agenda eletrônica de uso individual passou a ser uma agenda de grupos, agendando reuniões e solicitando confirmação das mesmas. O correio eletrônico passou a substituir os tradicionais recados, memorandos e protocolos. Sistemas de vídeo-conferência e de apresentação distribuída são outros exemplos de *groupware*. Maiores informações sobre *software* para grupos de trabalho podem ser vistas em [BRSCHMIDT93][BOYCE92]

1.1.5 Gerência de redes

As redes de computadores trouxeram um trabalho adicional para os administradores de sistemas. Os problemas de administração deixaram de ser centralizados para passarem a ser distribuídos. Como, por exemplo, fazer o backup dos dados que estão nos PCs conectados à rede a partir da console de um servidor de arquivos Netware ou Unix? A gerência de redes trata desse tipo de problemas, entre outros.

Gerência de redes por *software* não é um problema exatamente bem resolvido. Há carência de *software* para auxiliar os administradores de redes. As interfaces de alto nível para sistemas de gerência de redes como o NetView da HP são ainda muito caros. Soluções mais baratas para redes de pequeno porte tornam-se necessárias.

1.2 Descrição do trabalho

Informação é um ponto crucial na administração de qualquer empresa ou instituição, e a velocidade com que a mesma circula, bem como a garantia de que a mesma será digerida é que garante um diferencial importante em dias competitivos como os que vivemos. Não adianta só ter a informação, ela precisa ser distribuída e digerida. De que vale um memorando que ninguém lê?

As redes de computadores estão, hoje, presentes na maioria das organizações. Existe portanto, todo um suporte de comunicação dentro desses ambientes pouco explorado. Vejamos o seguinte exemplo: em uma determinada empresa, cada diretor tem a seu dispor, uma estação de trabalho de última geração, conectada em rede local e usa a máquina basicamente para edição de textos. Nessa mesma empresa existe um servidor de arquivos enorme, alimentado com dados de todas as filiais, e diariamente o CPD gera um relatório de vendas em uma impressora a laser colorida para a diretoria. Ora, se existe uma rede conectando os computadores da empresa, porque não disponibilizar toda essa informação na mesa dos diretores usando como meio de comunicação essa rede ao invés de papel? Vários são os motivos que levam a essa sub-utilização da rede, entre eles podemos citar a falta de cultura em computação distribuída, a carência e o alto custo de *software* para grupos de trabalho (*groupware*).

Usuários de micro-computadores em um ambiente distribuído têm à sua disposição, como vimos, um ferramental de comunicação pouco explorado. Seja pelas dificuldades encontradas em se desenvolver *software* distribuído ou pelo alto custo desse *software*. Gerentes/administradores de ambientes distribuídos têm por sua vez, "problemas distribuídos", e pouco ferramental de ajuda na sua tarefa de administração desses ambientes.

O NetComm é uma ferramenta que por sua versatilidade permite a construção de um grande número de aplicações multimídia distribuídas. O NetComm possibilita o uso de recursos multimídia como sons e imagens na comunicação entre usuários de uma rede local. A liberdade propiciada na definição de como e quando a informação será apresentada, bem como a variedade de objetos que o NetComm manipula, tornam-no uma ferramenta versátil para a construção de aplicações multimídia distribuidas. O NetComm é um *software* que permite também administrar redes locais de computadores, usando um conceito inovador que é a ativação de eventos remotos.

Por exemplo, o administrador da rede pode usar o NetComm para ativar remotamente aplicações de backup nos PCs da rede e controlar essas aplicações usando eventos remotos.

1.3 Contribuições do trabalho

O NetComm é um software que incorpora vários assuntos ditos "estado-da-arte" em informática: interfaces gráficas, multimídia, gerência de redes e arquitetura cliente-servidor. O NetComm faz uso do suporte de comunicação proporcionado pela rede para, através de uma arquitetura cliente-servidor, possibilitar a comunicação entre usuários usando recursos multimídia. Além disso, o NetComm possibilita a gerência de redes fazendo uso de eventos remotos.

O conceito de eventos remotos para administrar redes de computadores é inovador. A idéia é utilizar o paradigma de eventos propiciado pelas interfaces gráficas, gravando sequências de eventos de, por exemplo, mouse e teclado para posterior reprodução em máquinas remotas.

O NetComm possibilita o desenvolvimento de novas aplicações para grupos de trabalho sendo de grande importância para o desenvolvimento de novos trabalhos de pós-graduação no Departamento de Engenharia Elétrica.

1.4 Organização do trabalho

Este trabalho foi organizado em 6 capítulos. O primeiro, este, é uma introdução que dá uma visão geral sobre as tendências no desenvolvimento de *software*. O segundo capítulo apresenta algumas considerações sobre eventos distribuídos e a maneira como o NetComm implementa essa funcionalidade. O terceiro capítulo apresenta o NetComm, mostrando suas aplicações, objetos, e principais telas. No quarto capítulo se encontra uma descrição da implementação do NetComm, sua arquitetura, protocolos e classes de objetos. O quinto capítulo apresenta detalhes importantes da implementação como por exemplo, a interface de rede. O sexto e último capítulo está dedicado às considerações finais, conclusão e propostas de trabalhos futuros.

Capítulo 2 - Eventos Distribuídos

Os ambientes gráficos como o Windows [MSOFT92a] ou Motif [SILVA93] são orientados a eventos. Neste contexto, um dado evento ocorre e uma ação é tomada em função do mesmo. As aplicações escritas para esses ambientes e que executam sob os mesmos também funcionam dessa maneira; eventos ocorrem e ações são tomadas para tratá-los. Este capítulo é apresenta uma forma de, aproveitando o suporte de orientação a eventos fornecido por essas interfaces gráficas, gerenciar redes de computadores interferindo na execução de aplicações remotas.

O problema de gerência de redes não é fácil de resolver. Os protocolos de gerência de redes como SNMP[1] [STALLINGS93], apesar de ser a alternativa mais difundida, esbarram no pouco volume de aplicações que entendam esses protocolos. Nossa proposta permite que aplicações não SNMP gerenciáveis sejam gerenciadas.

2.1 Eventos distribuídos como alternativa para gerência de redes.

As aplicações para ambientes gráficos como Motif ou Windows usam o suporte de orientação a eventos fornecido por esses ambientes. Qualquer ação do usuário é considerada um evento e esses eventos são informados ou repassados para a aplicação na forma de mensagens. Desta maneira, o pressionar de uma tecla é considerado um evento e passado na forma de uma mensagem para que a aplicação faça o tratamento adequado. Neste contexto a mensagem é apenas uma estrutura de dados para representar o evento em questão.

A idéia de eventos distribuídos é basicamente uma maneira de simular a existência de eventos para controlar aplicações remotas. Ou seja, os eventos são simulados e repassados para a aplicação na forma de mensagens para que a mesma faça o tratamento previamente programado. Com essa filosofia podemos ativar uma aplicação remota e enviar para ela uma sequência de eventos ativando menus, alterando valores de campos, entre outros. Por exemplo, se a aplicação

---

[1] SNMP é sigla de *Simple Network Management Protocol*, um protocolo para gerência de redes.

remota fosse um spooler de impressão que não suportasse gerência remota, poderíamos gerenciá-lo usando eventos remotos simulando sequências de teclado e/ou mouse que, por exemplo, removessem pedidos da fila local de impressão. Essa forma de gerenciar aplicações remotas, apesar de conceitualmente simples, apresenta vários problemas de difícil solução que serão tratados neste capítulo. Iniciaremos apresentado algumas definições que serão importantes para o melhor entendimento desses problemas e para delimitação do nosso contexto.

2.2 Definição, classificação e representação de eventos

Para melhor entender os problemas a serem resolvidos em uma implementação de eventos distribuídos formalizamos uma definição para evento e o dividimos em classes.

2.2.1 Evento

Neste contexto, evento é qualquer coisa que ocorre seja por intervenção do usuário, do *hardware*, de dispositivos externos, do gerenciador de janelas, do sistema operacional ou das aplicações que executam sob o mesmo e que a aplicação toma conhecimento através de uma fila de mensagens. Maiores informações sobre filas de mensagens podem ser encontradas no capítulo 4.

2.2.2 Classes de eventos

Dividimos os eventos em classes de acordo com a figura a seguir. Os eventos são divididos em duas classes básicas: **eventos de *software*** e **eventos de *hardware***. Os eventos de *hardware* são aqueles gerados por dispositivos periféricos em geral como: *mouse*, teclado, porta serial, entre outros. Os eventos de *hardware* se subdividem em duas subclasses: **eventos de interação com o usuário** e **outros**. Os eventos de interação com o usuário são gerados por dispositivos de interface com o usuário como: *mouse*, teclado, caneta óptica, entre outros. Na subclasse "outros" incluímos os eventos não gerados por intervenção do usuário como: eventos de portas seriais ou paralelas e eventos de rede. Um exemplo desse tipo de evento seria a chegada de um dado na porta serial.

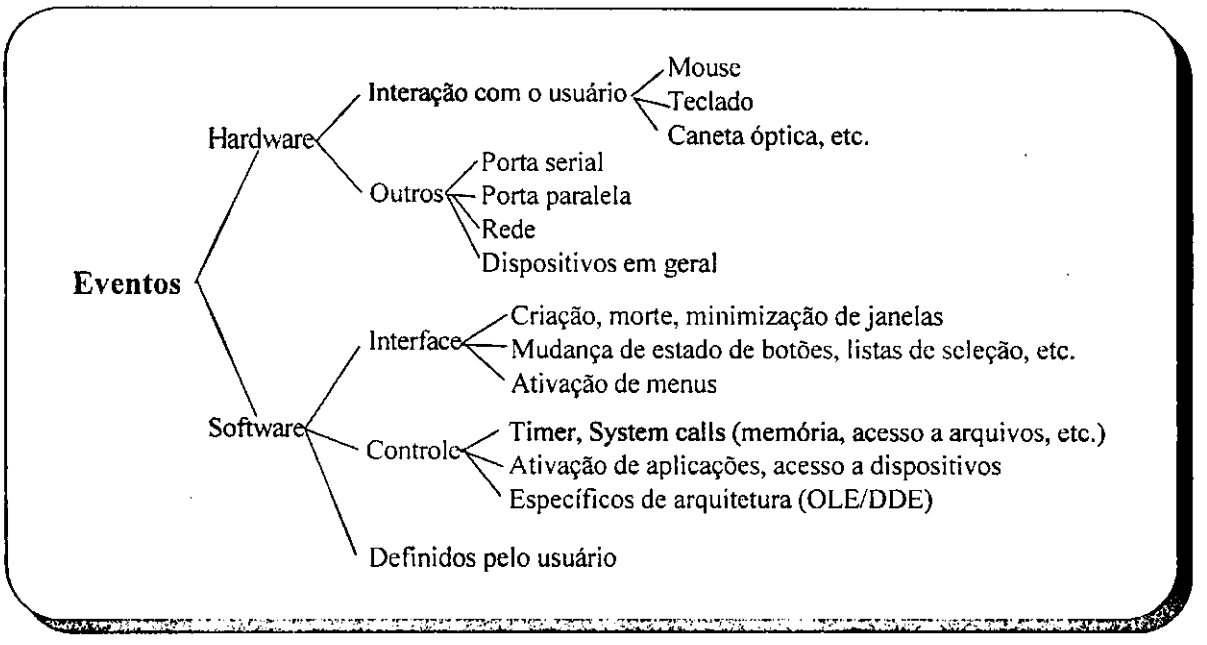

Figura 2.1 - Classes de eventos

Os eventos de *software* são consequência de eventos de *hardware* ou de ações tomadas pela própria aplicação. Por exemplo, quando o usuário move uma janela de um lugar para outro arrastando-a com o mouse são gerados vários eventos de *hardware* relacionados ao mouse. Esses eventos geram um grande número de eventos de *software*, para por exemplo, informar que a parte que estava anteriormente coberta pela janela e que ficou descoberta em consequência de sua movimentação deve ser repintada. Dividimos os eventos de software em três subclasses: **eventos de interface com o usuário**, **eventos de controle** e **eventos definidos pelo usuário**. Os eventos de interface com o usuário são normalmente consequência de eventos de interação com o usuário (*hardware*), como por exemplo, o pressionar de um botão ou a ativação de um menu. Os eventos de controle são mais gerais e são relativos tanto a eventos de *hardware* (*timer*) como a eventos gerados por ações da aplicação ou do próprio Sistema Operacional (chamadas a funções do sistema, acesso a dispositivos, ativação de aplicações). Os eventos definidos pelo usuário são eventos não reconhecidos pelo ambiente gráfico e específicos da aplicação. São eventos que o usuário define para representar e/ou tratar determinadas situações de sua aplicação. Por exemplo, uma aplicação poderia definir um evento X qualquer que ocorresse sempre que a quantidade de espaço em disco disponível ficasse abaixo de um determinado valor.

2.2.3 Representação de eventos

Para representar eventos resolvemos usar uma estrutura de dados com alguns campos fixos e um outro dependente do tipo do evento. A estrutura por nós definida para representar um evento é uma mensagem, e é apresentada abaixo:

Mensagem:

- Código do evento
- Tempo de ocorrência
- Tipo do evento
- Dados dependentes do tipo

O código do evento garante uma identificação única do mesmo. O tempo de ocorrência indica o momento em que o evento ocorreu. O tipo do evento determina de qual classe e subclasse o mesmo é. Existe ainda, um campo que armazena dados dependentes do tipo do evento em questão. Por exemplo, um evento de teclado teria dados indicando o código da tecla pressionada já um evento de mouse teria dados relacionados à posição (X,Y) em que o mesmo ocorreu.

2.3 Eventos gerenciando redes de computadores

Voltando ao problema inicial que era gerenciar redes de computadores interferindo na execução de aplicações remotas, vejamos como isso funciona. Precisamos gerar eventos que serão repassados para uma aplicação remota fazendo uso do suporte físico de uma rede de computadores. Em primeiro lugar precisamos ter um mecanismo para gerar esses eventos de forma automática, em segundo lugar um mecanismo para transmitir esses eventos até seu destino e em terceiro lugar, uma forma de repassar esses eventos para a aplicação em questão. Na figura 2.2 a seguir podemos ver como esse processo funciona. Na máquina 1, à medida que os eventos destinados à aplicação X ocorrem, eles são interceptados por um gravador de eventos, armazenados em arquivo ou memória e repassados para a aplicação. Em um segundo passo, os eventos são transmitidos para a máquina 2 através de um transmissor de eventos. Na máquina 2 um receptor de eventos recebe os eventos enviados pela máquina 1. Esses eventos são então

capturados por um reprodutor de eventos que se encarrega de repassá-los para que a aplicação os trate como se eles tivessem sido gerados localmente.

Partindo do pressuposto que as aplicações já existem, nosso problema se limita à construção do gravador/reprodutor de eventos e do transmissor/receptor. A construção do transmissor e do receptor é bastante simples. O transmissor é simplesmente um processo que envia os dados relativos aos eventos, já o gravador/reprodutor precisa de um suporte adicional da interface gráfica. No caso do Windows, esse suporte é fornecido por meio de *hooks*[2] [RITCHTER92]. Esses *hooks* são responsáveis por interceptar os eventos destinados à aplicação antes que os mesmos sejam repassados para ela através da fila de mensagens.

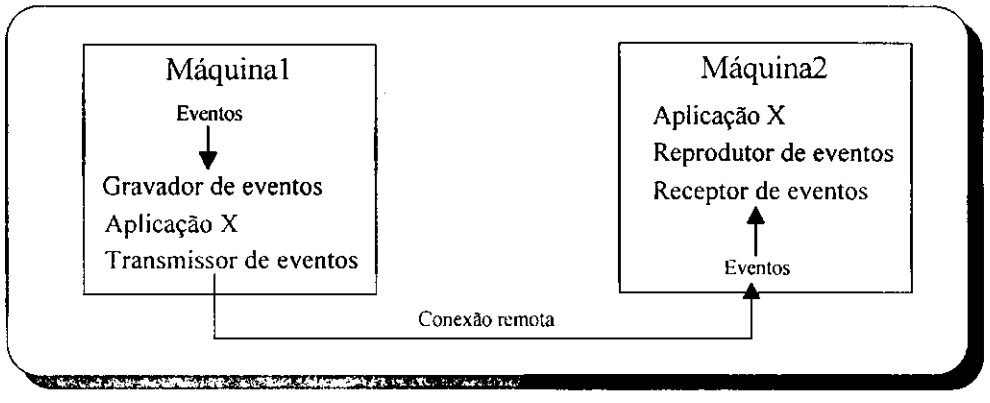

Figura 2.2 - Mecanismo de transmissão de eventos

Arquiteturalmente, o suporte para eventos distribuídos está definido. Esta arquitetura, porém, tem algumas limitações que serão tratadas a seguir.

2.4 Considerações sobre eventos distribuídos

Nesta seção apresentaremos algumas considerações sobre eventos distribuídos que dizem respeito a problemas encontrados na arquitetura definida. No final trataremos sobre questões a serem resolvidas em uma implementação de eventos distribuídos em redes heterogêneas.

[2] Hooks, ganchos em Inglês, são funções especiais fornecidas pelo Windows para interceptar determinados eventos antes que os mesmos sejam repassados para a aplicação.

2.4.1 Considerações sobre dispositivos

As primeiras considerações a serem formuladas sobre eventos distribuídos dizem respeito às diferenças existentes entre os dispositivos. Como vimos, classificamos os eventos em duas classes básicas: eventos de *hardware* e eventos de *software*. Ao gravarmos sequências de eventos de *hardware* de um dispositivo qualquer como, por exemplo, o *mouse* nos defrontamos com questões relativas à semântica do evento em questão.

Na estrutura definida para representar eventos definimos um campo dependente do tipo do evento. Para eventos gerados pelo *mouse*, armazenamos nesse campo a coordenada (x, y) na qual se encontrava o mouse no momento em que o evento ocorreu. O grande problema é que a semântica associada ao evento não foi armazenada. Assim, um clique de *mouse* na posição (x, y) pode significar ativar uma opção de menu em uma máquina, mas ao reproduzirmos esse evento em outra máquina esse mesmo evento pode vir a não acessar uma opção de menu e sim ter outro significado qualquer. Bastaria que no momento da reprodução, o menu estivesse ligeiramente deslocado para que esse clique não o ativasse. Ou seja, o clique foi ativado na posição correta, porém a semântica associada a esse clique não foi a mesma.

Esse problema poderia ser contornado usando o teclado para acessar o menu ao invés do *mouse*, mas essa solução não é adequada. Em primeiro lugar estaríamos limitando o usuário ao uso do teclado e em segundo lugar não poderíamos tratar os casos de aplicações que não aceitam o controle via teclado.

Uma outra solução é usar coordenadas relativas não à tela toda e sim à janela da aplicação. Isso resolve o problema na teoria. Na prática deixa de funcionar a partir do momento em que nos defrontarmos com diferentes controladoras de vídeo ou mesmo diferentes configurações. Por exemplo, na máquina em que o evento foi gerado a controladora de vídeo estava configurada para uma resolução de 800 por 600 pontos enquanto na que estávamos tentando reproduzir o evento, a controladora de vídeo usava uma resolução de 640 por 480 pontos. O centro da janela, por exemplo, terá coordenadas distintas nas duas máquinas. Mesmo usando-se controladoras de vídeo de mesma resolução podem ocorrer pequenas distorções devido a *drivers*[3] de fabricantes distintos.

---

[3] Termo usado para programas controladores de dispositivos em geral.

A solução seria então armazenarmos um evento de *software* ao invés de um evento de *hardware*. Esse evento seria a ativação da opção de menu e não o clique do mouse. Esta solução é a melhor possível porém apresenta complicações maiores em se tratando de ambientes heterogêneos.

2.4.2 Considerações sobre configuração e localização

Ao gravarmos sequências de eventos para posterior reprodução devemos levar em consideração a existência de máquinas com diferentes configurações, versões de software distintas ou mesmo diferenças entre línguas.

Ao gravarmos eventos de teclado para acessar uma opção de menu estamos considerando que a tecla aceleradora será a mesma para todas as aplicações mas isso pode não ser verdadeiro. Uma aplicação pode ter sido instalada em uma máquina em sua versão em inglês e em outra em sua versão em português. Nesse caso a tecla aceleradora para acessar um menu poderia ser diferente. Por exemplo, a opção *File* (tecla aceleradora F) presente na maioria dos aplicativos para Windows tem sua versão em português sendo Arquivo (tecla aceleradora A). Ou seja, o evento seria gravado como o pressionar da tecla F que ativa o menu *File* mas no momento da reprodução do evento a tecla F não ativaria esse menu.

Uma solução é forçar o usuário a usar as teclas de direção ao invés de teclas aceleradoras. Novamente estaríamos impondo restrições ao usuário. Isso nos leva novamente ao uso de eventos de *software* ao invés de eventos de *hardware,* mas mesmo essa solução apresenta problemas.

As aplicações para Windows ou para interfaces gráficas em geral cresceram enormemente em complexidade e é normal que as mesmas permitam que o usuário as configure de acordo com seu nível de aprendizado nas mesmas. É comum encontrar aplicações que permitem que o usuário configure os menus de acordo com suas necessidades. Nessa situação, o evento, seja de *hardware* (*mouse* ou teclado) ou de *software* (ativação de menu) , não seria corretamente reproduzido pois a opção de menu existente em uma máquina simplesmente não existiria na outra. Esse efeito ocorre também para opções de menu que são desativadas/ativadas em determinadas situações.

2.4.3 Considerações sobre eventos interativos

Na arquitetura para eventos distribuídos proposta não existe o conceito de interatividade na reprodução de eventos. O processamento é feito em lote. Ou seja, um conjunto de eventos é gravado para posterior reprodução.

Essa arquitetura poderia prever conjuntos de eventos de um único elemento o que garantiria o processamento *on line* dos mesmos. Desta forma, o sistema poderia transmitir os eventos à medida que os mesmos fossem sendo gerados. Neste caso, a interatividade seria fundamental. É interessante que ao transmitir um evento obtenhamos respostas relacionadas à reprodução do mesmo especialmente no que diz respeito à sincronização entre as diversas máquinas da rede.

2.4.4 Considerações sobre sincronização

Na estrutura de dados usada para representar um evento incluímos um campo indicando o tempo em que o evento ocorreu. Os eventos são reproduzidos um a um de acordo com esse tempo. Ou seja, a velocidade com que os eventos foram gravados será a mesma com a qual eles serão reproduzidos. Essa forma de reprodução pode por vezes não ser interessante. As vezes o usuário deseja que a reprodução de eventos seja feita de forma rápida. Para conseguirmos uma reprodução rápida bastaria atualizar o campo "tempo" do conjunto de eventos.

Não existe o conceito de sincronização de reprodução entre distintas máquinas. Um mesmo conjunto de eventos poderia ser reproduzido em diversas máquinas da rede ao mesmo tempo e, por vezes, seria interessante que a reprodução desses eventos fosse sincronizada de alguma maneira. O processo de sincronização entre máquinas poderia ser feito através de pontos de sincronização [RYMER93][MULLENDER89][TANENBAUM92] inseridos automaticamente no conjunto de eventos. Ao se atingir um desses pontos o sistema de reprodução entraria em sincronização com os demais sistemas da rede. Essas considerações são de especial valia ao trabalharmos com a possibilidade de eventos interativos. Após a reprodução de cada evento poderíamos ter um ponto de sincronização com a máquina servidora (gravadora de eventos).

2.4.5 Considerações sobre heterogeneidade

A arquitetura definida para tratamento de eventos distribuídos não prevê heterogeneidade no ambiente. Para tratar ambientes heterogêneos algumas considerações devem ser observadas.

2.4.5.1 Problemas de representação e classificação

Em um ambiente heterogêneo se faz necessário um mapeamento de eventos. Um evento X qualquer não necessariamente dispõe da mesma representação em ambientes gráficos diferentes; esse evento pode inclusive não existir ou os seus dados específicos podem ser completamente distintos. Por exemplo, os dados específicos para eventos de mouse em uma determinada arquitetura podem ser simplesmente a coordenada (x,y) na qual o cursor estava posicionado mas em outra arquitetura esse dado pode ser outro ou usar um sistema de coordenadas diferente. Para resolver esse problema sugerimos uma tabela de mapeamento que transformaria todos os eventos em uma forma normal no momento da gravação e que os "destransformaria" no momento da reprodução de acordo com o ambiente gráfico utilizado.

A arquitetura de eventos usando o mapeamento descrito acima pode ser observado na figura 2.3 a seguir. Nessa nova arquitetura, um passo adicional de mapeamento é efetuado antes da gravação e antes da reprodução dos eventos. Ao efetuarmos a gravação o evento é normalizado e ao reproduzi-lo, o evento é desnormalizado segundo uma tabela de mapeamento dependente da arquitetura utilizada.

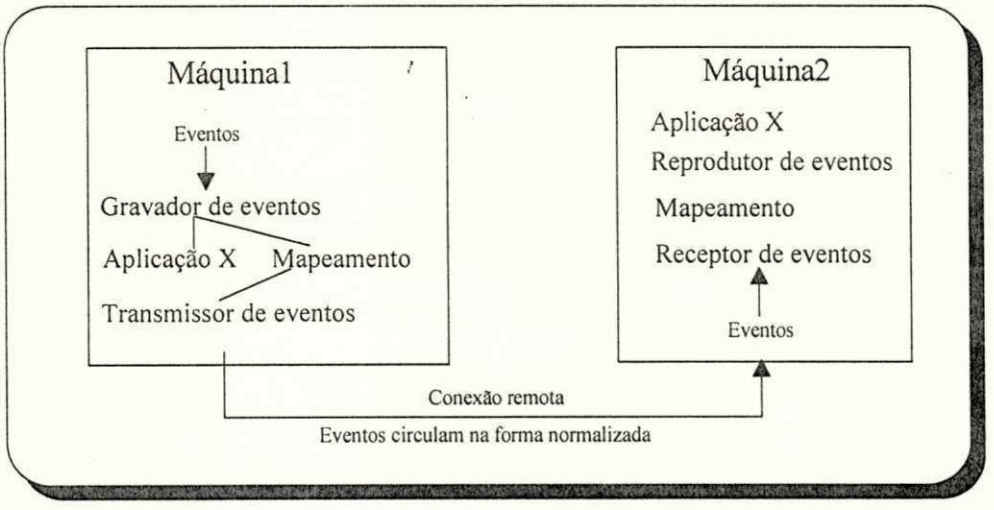

Figura 2.3 - Mecanismo de transmissão de eventos usando normalizador

A existência das tabelas de mapeamento e do processo de normalização não resolve o problema. Existem eventos que são dependentes de arquitetura ou mesmo eventos que não são mapeáveis de uma arquitetura para outra. Por exemplo, em uma determinada interface gráfica podem existir eventos para representar a minimização de uma janela mas em outra interface pode nem sequer existir o conceito de minimização. Ainda, detalhes específicos de arquitetura como compartilhamento de dados entre aplicações usando protocolos como DDE[4] [CLARK92] podem não ser mapeáveis em outros ambientes. Para esses casos seria necessário a construção de rotinas que implementassem o comportamento original em relação a esses eventos. Uma outra solução seria limitar o uso a eventos de *hardware*, mais facilmente mapeáveis.

2.4.5.2 Interfaces diferentes para mesmas aplicações

Aplicações construídas para ambientes heterogêneos tendem a ter interfaces diferentes. Por exemplo, as aplicações escritas para Windows possuem em sua grande maioria um menu para acessar o *clipboard*[5]. Esse menu pode não estar presente na mesma aplicação escrita para ambientes Motif, logo os eventos relacionados a esse menu não refletirão a mesma função em seu par remoto.

2.5 Uma implementação homogênea de eventos distribuídos

Neste trabalho apresentamos uma implementação de eventos distribuídos como parte do NetComm. Essa implementação restringe os eventos gravados a eventos de *hardware* e de interação com o usuário (eventos de *mouse* e teclado) e é uma implementação apenas para Windows, não se preocupando com detalhes de heterogeneidade. Esta implementação também não faz uso do conceito de eventos interativos apresentado na seção 2.4.3. Por serem gravados apenas eventos de *hardware*, é recomendável que não se use o *mouse* para acionar controles como botões ou menus, e sim o teclado.

---

[4] DDE, *Dynamic Data Exchange* em inglês, é um protocolo que permite o compartilhamento de dados entre aplicações no ambiente Windows.

[5] Área de transferência usada para troca de dados entre aplicações em ambiente Windows

Capítulo 3 - Entendendo o NetComm

3.1 O que é o NetComm

O NetComm é um *software* para Windows que permite a comunicação entre usuários de um ambiente distribuído local usando recursos audiovisuais. O NetComm é uma ferramenta que permite a construção de aplicações multimídia distribuídas. Ele é ainda um ferramental de auxílio na gerência de redes, já que permite a ativação de eventos e aplicações remotas. O NetComm é um *software* que incorpora quatro assuntos estado-da-arte da informática: multimídia, gerência de redes, software distribuído e Windows.

O NetComm é um software que manipula informação multimídia e essa informação é apresentada para o usuário na forma de **projetos**. A próxima seção descreve o que é um projeto e sua importância para o NetComm.

3.2 Projetos NetComm

Denominamos o principal objeto manipulado pelo NetComm de **projeto**. Existem três fases básicas pelas quais passa um projeto: criação, distribuição e apresentação. Quando um usuário deseja se comunicar com outro ele constrói um projeto, que vai ser distribuído pelas máquinas da rede e posteriormente apresentado. São portanto, projetos tudo aquilo que o usuário vai poder construir, distribuir e apresentar fazendo uso do NetComm.

Para construir um projeto o usuário precisa definir: que objetos o comporão, como e quando o projeto deverá ser apresentado e a quem o mesmo se destina. A essas partes envolvidas na fase de construção de um projeto, acrescida de outras duas que são definidas automaticamente (quem é o usuário criador do projeto e qual a máquina servidora do mesmo) denominamos **dimensões de um projeto**.

3.3 Dimensões de um projeto

Um projeto é composto por diversas partes definidas pelo usuário as quais denominamos dimensões de um projeto. Para entender melhor esse conceito e o que vem a ser um projeto, descrevemos a seguir cada uma das dimensões que o compõem.

3.3.1 "O quê"

Esta dimensão define quais os objetos que comporão um projeto. O NetComm permite uma grande variedade de objetos que podem ser usados na definição da dimensão "O quê" de um projeto. O usuário pode incluir gráficos, sons, textos e eventos em um projeto e determinar quando cada um deles deve ser apresentado. Para facilitar a compreensão, dividimos estes objetos em duas classes: visuais e não visuais. Os objetos visuais têm representação na tela e podem ou não ser animados. Os objetos não visuais não possuem representação visual, ou seja, não são visíveis para o usuário (por exemplo, som). Para animar os objetos visuais o NetComm dispõe de um poderoso editor de animações. O NetComm usa, desta forma, recursos avançados de multimídia. O usuário pode, por exemplo, construir um projeto e incluir sua própria voz no mesmo, de forma a que enquanto um gráfico percorre a tela sua própria voz descreva o que o gráfico representa. Mais exaustivamente, os objetos manipulados são:

- Gráfico:
  - Um projeto NetComm pode incluir gráficos no formato padrão do Windows (DIB, sigla de *Device Independent Bitmap* ou simplesmente *Bitmap*) e esses gráficos podem ou não ser animados. Existem dois tipos de animação para objetos gráficos: **animação de percurso** e **animação de transformação**. A animação de percurso define como o objeto vai se movimentar pela tela e a animação de transformação define como o próprio objeto vai se transformar à medida que essa movimentação ocorre. Por exemplo, imaginemos uma bola que inicia uma trajetória horizontal no lado esquerdo de uma janela e termina seu caminho no lado direito da mesma. A esse caminho percorrido denominamos de animação de percurso. Agora imagine que essa mesma bola executa um movimento de rotação

sobre seu próprio eixo enquanto se dirige ao lado direito da janela. A esse movimento de rotação demos o nome de animação de transformação.

- Texto:
  - Um texto pode ser incluído em um projeto NetComm. Esse texto pode, de forma análoga a um gráfico, ser animado, mas só é possível definir animação de percurso para o mesmo.

- Som:
  - Os sons podem ser incluídos em projetos NetComm desde que eles sigam o padrão *Wave* (arquivos com terminação .WAV) da Microsoft Corporation. Outros padrões tais como VOC, por exemplo, não são aceitos. Essa escolha foi feita em virtude da API (*Application Programming Interface*) de multimídia do Windows aceitar diretamente o formato *Wave*.
- Evento:
  - Objetos deste tipo dão ao NetComm o poder de administrar computadores em um ambiente distribuído e é, com certeza, um dos mais poderosos recursos que o mesmo dispõe. O funcionamento é equivalente a um gravador/reprodutor de macros baseado em eventos como, por exemplo, o *Recorder*[6] que acompanha o Windows. Com ele o usuário grava uma sequência de eventos de mouse ou teclado que poderá ser reproduzida posteriormente. No NetComm a idéia é a mesma, com a diferença de que esses eventos podem ser reproduzidos remotamente e de que é possível definir uma aplicação a ser executada remotamente para receber esses eventos.

O poder que a combinação desses objetos dentro de um projeto dá ao NetComm será descrito mais adiante quando falarmos de aplicações do NetComm. Por enquanto, podemos imaginar as possibilidades que essas combinações podem gerar. Ainda, a forma ortogonal com que são tratados os diferentes tipos de objeto, permite que novos tipos sejam incluídos. Por

---

[6] O Recorder (Recorder.exe) é uma aplicação gravadora/reprodutora de macros que acompanha o Windows.

exemplo, objetos do tipo "imagem de vídeo" poderiam ser incluídos em um projeto em uma versão futura do NetComm.

3.3.2 "Como"

Esta dimensão define como os objetos descritos na parte "O quê" deverão ser apresentados ou melhor dizendo, ela define atributos da janela onde esses objetos serão apresentados. Como exemplos desse atributos podemos citar: o tamanho da janela, a cor de fundo da mesma, se ela têm título, entre outros. Cabe salientar que pelo fato de existirem objetos que não precisam de "janelas" para serem apresentados (som e evento), ou seja, não têm representação visual, a parte "Como" de um projeto é dispensável se somente objetos desse tipo forem definidos para um projeto.

3.3.3 "Quando"

Esta dimensão define quando um projeto construído deverá ser apresentado. Existem várias formas de apresentação de um projeto:

- Imediato
- Por escalonamento
- Por *time-out* de inoperação

Quando um projeto é imediato (dimensão "Quando" definida assim), ele é apresentado para o usuário a quem se destina imediatamente após ter sido enviado para o mesmo. Esta forma de apresentação é útil para, por exemplo, enviar uma mensagem rápida para algum usuário da rede o qual sabemos estar conectado.

Quando o projeto é por escalonamento, o usuário define quando o mesmo deve começar e terminar de ser apresentado e em que períodos de tempo. Com esse tipo de apresentação podemos, por exemplo, definir um projeto que só vai ser apresentado no dia primeiro de cada mês. Ou seja, podemos determinar o(s) momento(s) exato(s) em que será apresentado um projeto.

Um projeto ser ativado por *time-out* de inoperação significa que o mesmo só vai ser apresentado em períodos de inatividade do usuário a quem se destina. Isto é, o projeto só será

apresentado quando o usuário não estiver fazendo nenhuma atividade com o *mouse* ou com o teclado. Esta forma de ativar a apresentação de um projeto é interessante já que não interfere no trabalho do usuário destino. Esta também é a maneira com que os programas do tipo *screen-saver* trabalham.

Existe ainda, uma quarta forma de apresentação de um projeto mas que só é definida localmente, na máquina destino do projeto. O usuário que recebeu um projeto de alguém pode definir um conjunto de teclas de ativação rápida, que servirão para apresentar o projeto sempre que as mesmas forem pressionadas simultaneamente, independentemente de como foi definida a parte Como do projeto. Um conjunto de teclas de ativação rápida (*short cut key*) poderia ser, por exemplo, CTRL+SHIFT+ENTER.

3.3.4 "Para quem"

Esta dimensão do projeto define a quem o mesmo se destina. É basicamente uma lista de pares ordenados (usuário, máquina) com o nome dos usuários/máquinas aos quais se destina o projeto. Na definição desta dimensão existe o conceito de usuário e máquina "qualquer". Isso que dizer que um projeto pode ser construído para ser apresentado em, por exemplo, qualquer máquina da rede.

3.3.5 "Por quem"

Esta dimensão é definida automaticamente pelo NetComm. Ele determina qual o usuário que construiu o projeto e em que máquina da rede. Isso é útil por questões de segurança e auditoria. É sempre interessante saber quem faz o quê em um ambiente distribuído.

3.3.6 "Máquina servidora"

Esta dimensão também é definida automaticamente pelo NetComm com o nome da máquina onde o projeto foi construído. Porém é permitido alterar esse valor para permitir que um projeto seja construído em uma máquina mas que após essa fase de construção, outra máquina fique como servidora de dados do mesmo.

Essa é uma característica interessante já que em um ambiente distribuído, são poucas as máquinas que ficam conectadas à rede 24 horas por dia. A definição dessa dimensão permite que um projeto seja construído em uma máquina qualquer (que não fique ligada as 24 horas) e que depois outra máquina da rede passe a manipular os dados desse projeto.

3.4 Objetos dinâmicos

Um atributo comum tanto aos objetos visuais quanto aos não visuais e de suma importância, é o dinamismo. Um objeto dinâmico tem a capacidade de se atualizar periodicamente e essa atualização se reflete em todas as máquinas da rede. Suponha que um determinado diretor de empresa necessite de um gráfico representando a evolução das vendas, de hora em hora em suas mãos. Esse gráfico é gerado por uma planilha de cálculo e atualizado constantemente de acordo com a emissão de notas da loja. Um projeto NetComm poderia conter esse gráfico e bastaria ligar o atributo de dinamismo para esse objeto de forma que o mesmo estaria sempre atualizado na hora de ser apresentado. É interessante ressaltar que qualquer objeto pode ter o atributo de dinamismo ligado independentemente dele ser visual ou não. Aos objetos com esse atributo denominamos de dinâmicos, enquanto aos que não o possuem denominamos de estáticos.

Na figura abaixo podemos ver como essa atualização se efetua. O diretor da empresa do exemplo anterior está conectado ao computador A e o gráfico de evolução de vendas está sendo gerado no computador B de outro departamento. Os computadores A e B estão conectados em uma rede local e o NetComm está rodando em ambos, sendo que o assessor de vendas construiu um projeto para o diretor contendo esse gráfico. No momento de apresentar esse projeto para o diretor, o NetComm descobre que o gráfico é um objeto dinâmico, portanto deve ser atualizado. Ele pede ao NetComm que está rodando no computador B que atualize seu gráfico e o computador B envia o novo gráfico já atualizado.

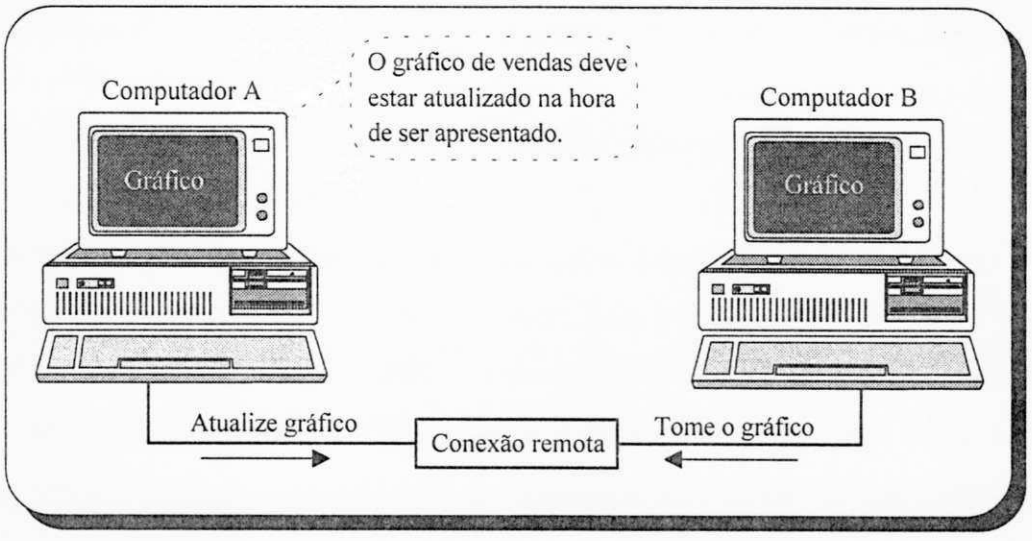

Figura 3.1 - Atualização de dados dinâmicos

3.5 Aplicações do NetComm

O NetComm é um software, com um espectro de aplicações enorme. A combinação dos objetos que compõem a parte "O Quê" de um projeto com as outras dimensões permite que uma grande gama de aplicações possa ser construída usando o NetComm para automatizar o processo de distribuição, armazenamento, apresentação e gerência de informação em uma rede local.

O uso do NetComm pode variar desde aplicações simples como um *screen-saver* padronizado com o logotipo da empresa ou um serviço de lembretes distribuído até poderosas aplicações de gerência como um backup distribuído em rede ou um software de apresentação distribuído. A seguir apresentamos alguns exemplos de aplicações do NetComm que são de largo uso pelas empresas. Não vamos nos deter em aplicações triviais, como simplesmente o envio de mensagens para outros usuários da rede que é provavelmente a aplicação mais imediata do NetComm.

3.5.1 Um serviço de lembretes distribuído.

Como primeira aplicação do NetComm, apresentamos um serviço de lembretes distribuído. Agendas eletrônicas para ambiente Windows são extremamente comuns. O próprio Windows vem acompanhado de uma (*calendar.exe*). Os programas do tipo PIM (*Personal*

Information Manager), como o Lotus Organizer, são exemplos de uma agenda eletrônica com um número maior de recursos. O que o NetComm possibilita fazer é um serviço de lembretes distribuido, ou melhor dizendo, uma forma de lembrar/avisar grupos de usuários em uma rede local de reuniões ou compromissos.

Por exemplo, quando o gerente de uma empresa desejar marcar uma reunião com seus acessores, basta ele construir um projeto NetComm com a dimensão "Quando" definida para lembrá-los da reunião alguns minutos antes e a dimensão "Para quem" destinado às pessoas que deverão comparecer à reunião. Caso a reunião seja de frequência semanal, bastaria definir a dimensão "Quando" para, por exemplo, toda segunda-feira às 8:30 caso a reunião fosse nesse dia e pelo período da manhã.

A dimensão "O quê" do projeto poderia trazer as informações relevantes ao assunto da reunião. Por exemplo, digamos que uma determinada empresa se reúne todas as segundas-feiras às 9:00 da manhã para discutir as metas de vendas da semana e o quanto foi vendido na semana anterior. A dimensão "O quê" poderia conter um objeto gráfico dinâmico gerado pela planilha de cálculo onde estão as informações sobre a evolução das vendas da empresa. Pelo fato do gráfico ser dinâmico, toda segunda-feira, minutos antes da reunião os participantes teriam a informação atualizada sobre a evolução das vendas disponível em sua mesa (no computador). Todo o processo de composição e distribuição da informação é automático eliminando problemas de ordem humana. A empresa com isso ganha tempo, e cada um chega na reunião já sabendo o que vai ser discutido e com dados atualizados. O processo de construção do projeto precisa ser feito uma única vez e a partir de então tudo é automático.

3.5.2 Um sistema de backup distribuído

Informação é uma coisa essencial para qualquer empresa ou instituição e a segurança com que essa informação é mantida é mais essencial ainda. Políticas de backup dos dados de uma empresa têm sido um dos tendões de Aquiles dos administradores de sistemas quando o ambiente não é centralizado ou quando os dados são sempre armazenados em um servidor de arquivos.

Na maioria das empresas, o backup de arquivos é feito, quando muito, dos dados do servidor de arquivos. Com o advento das redes locais tornou-se necessário fazer backup dos dados existentes nas estações de trabalho ou nos PCs conectados à rede. Agora, convencer um

usuário de PC, como uma secretária ou um presidente de empresa de que ele deve tirar backup de seus dados é uma tarefa de resultados duvidosos. A maioria deles corre desesperadamente em direção ao responsável pelo CPD para tentar saber se existe a possibilidade de recuperar os estragos causados por vírus, crash ou erros humanos (del *.*, por exemplo).

Ao dispor de várias máquinas conectadas a uma rede local existe uma distribuição de dados dentro da empresa. O banco de dados da empresa está protegido por uma política de backup, já que o mesmo se encontra no servidor de arquivos. Porém, os dados manipulados e armazenados em disco local pelos PCs não. Fazer backup dos dados é uma tarefa que parece escapar da compreensão dos usuários comuns, ou seja, usuários dos PCs.

Com o NetComm, poderíamos criar uma política de backup única para todos os PCs da empresa. Basta simplesmente criar um projeto como descrevemos a seguir. Queremos que o backup seja feito diariamente às 12:00 e às 18:00 horas em todas as máquinas da empresa. Logo, a dimensão "Para quem" deve ser definida como máquina=qualquer e usuário=qualquer. A dimensão "Quando" deve ser definida para toda segunda, terça, quarta, quinta e sexta às 12:00 e 18:00 horas e a dimensão "O Que" deve conter um objeto do tipo evento. Esse objeto deve abrir a aplicação responsável pelo backup e ativar a sequência de eventos necessária para concluir a operação. Caso desejássemos que alguma mensagem fosse apresentada na hora do backup ser iniciado ("Coloque o disquete de backup no drive", por exemplo) bastaria incluir um outro objeto, este agora do tipo texto. Pronto, o backup dos dados da empresa está feito sem maiores problemas! A aplicação de backup não representa custo adicional, já que o próprio DOS vem acompanhado de um sistema de backup para Windows de muito boa qualidade. Uma outra forma seria ativar um arquivo de comandos do DOS que copiasse os arquivos locais pela rede para o servidor de arquivos da empresa. O backup seria então feito normalmente pelo servidor.

3.5.3 Um screen-saver personalizado

A maioria dos PCs hoje em dia possui algum *software* do tipo *screen-saver*. O Windows, que é o ambiente operacional mais usado em PCs, já vem acompanhado de alguns e a própria Microsoft incorporou uma nova API (*Application Programming Interface*) no Windows especificamente para o desenvolvimento desse tipo de *software*. O *screen-saver* é utilizado para evitar que o fósforo do monitor de vídeo queime pontos específicos. Ele é composto por imagens

animadas que se movimentam pela tela e inicia sua execução após alguns minutos de inatividade do usuário. Ao menor sinal de atividade de mouse ou teclado, o screen-saver morre e o controle retorna para o usuário no ponto onde ele se encontrava.

Essa forma de apresentar informação para o usuário é pouco explorada e é interessantíssima, já que a mesma é mostrada sempre em períodos de inatividade do mesmo. Usuários de computadores não gostam de ser interrompidos no trabalho sempre que recebem uma mensagem do administrador da rede ou um alarme da agenda eletrônica é ativado. Ao apresentar informação para o usuário em seus períodos de inatividade não estaremos interferindo em seu trabalho.

O NetComm dá uma nova vida ao tradicional *screen-saver* usando de forma mais inteligente essa maneira de apresentar a informação. Para construir um *screen-saver* com o NetComm, basta definir a dimensão "Quando" para "inatividade do usuário". A dimensão "Como" deve ser definida com a janela maximizada e sem título. A dimensão "O quê" deve conter objetos animados, sejam eles do tipo gráfico ou texto ou mesmo sons.

A grande vantagem de se usar o NetComm para construir *screen-savers* é que podemos personalisá-lo de acordo com nossos interesses. Por exemplo, o administrador da rede pode usar o NetComm para padronizar os *screen-savers* de todas as máquinas da empresa. Ainda, pode ser usado esse tipo de interação com o usuário para apresentar as mensagens do dia do tipo "Não esqueça de fazer seu re-cadastramento junto ao departamento de pessoal" ou o que assim desejar. Caso seu chefe seja daqueles que não gostam de interrupções em seu trabalho esta é a forma ideal de apresentar algum dado para ele. Cabe salientar que como as dimensões são completamente ortogonais e independentes uma das outras, nada impede que usemos a dimensão "Quando" como "inatividade do usuário" e a dimensão "Como" definida como uma janela comum ou minimizada. A versatilidade do NetComm é grande para possibilitar qualquer tipo de combinação.

3.5.4 Um sistema de apresentação distribuído

Sistemas de apresentação como o Harvard Graphics, o Power Point ou o Freelance fazem sucesso com aqueles que usam o computador para construir suas apresentações ou palestras. Ora, mas se temos uma rede de computadores em nossa empresa, porque deslocar todos os

funcionários para uma determinada sala onde possam assistir a apresentação. Essa apresentação poderia ser feita pela rede, cada um na sua mesa de trabalho e economizando o cafezinho da sala de reuniões. O NetComm pode ser usado para construir uma apresentação distribuída.

Digamos que um diretor gostaria de apresentar os novos resultados das vendas de seu setor para o conselho administrativo da empresa. Ele poderia construir um projeto NetComm com a dimensão "Para quem" destinada aos membros do conselho administrativo. A dimensão "O quê" poderia conter gráficos, textos demonstrando a evolução das vendas e sua própria voz poderia ser usada para explicar detalhes mais obscuros.

3.5.5 Gerador automático de relatórios

Suponhamos que todos os dias, um diretor administrativo precise de um relatório com as contas a pagar e a receber. O que é feito normalmente é imprimir um relatório contas a pagar e receber e colocá-lo na mesa desse diretor. Ora, se dispomos de uma rede de computadores, por que não usá-la? Melhor ainda se pudermos automatizar esse processo. Um projeto NetComm poderia ser construído para que o diretor dispusesse sempre de um relatório atualizado em seu PC na hora que ele desejasse. O NetComm poderia ser usado para primeiro, construir o relatório e depois apresentá-lo para o diretor.

Para tanto, basta que a dimensão "O quê" contenha um objeto do tipo evento. Ele seria responsável por ativar a aplicação que constrói o relatório e depois o imprimiria em um arquivo. Esse arquivo faria parte do projeto como um objeto do tipo texto e dinâmico. Ou seja, sempre que o projeto for apresentado, nova informação é gerada para o mesmo. Esse é só um exemplo, muitas outras coisas podem ser imaginadas para aproveitar o poder do NetComm.

3.6 Módulos do NetComm

Como dissemos anteriormente na seção 3.2, existem três fases pelas quais passa um projeto. Pois bem, para cada uma dessas fases existem um ou mais módulos do NetComm envolvidos. O NetComm é composto por quatro módulos básicos: NCMaker, NCPlayer, NCClient e NCServer. O NCMaker está envolvido na fase construção de um projeto, o NCPlayer

na fase de apresentação e o NCClient e o NCServer na fase de distribuição. Existe ainda um quinto módulo que se relaciona com todas essas fases que denominamos NCLogin e é responsável pela autenticação do usuário do NetComm.

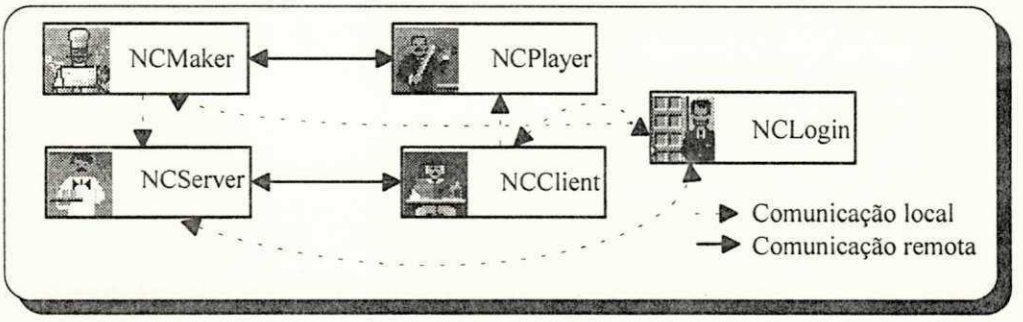

Figura 3.2 - Comunicação inter-processo do NetComm

A figura 3.2 mostra como os processos se inter-relacionam. Vamos tentar explicar resumidamente como se processa essa comunicação. O usuário constrói um projeto no NCMaker que faz uma comunicação inter-processo, a nível local, com o NCServer para avisá-lo que há um projeto para ser enviado. O NCServer se comunica de forma remota com o NCClient de cada máquina que deve receber o projeto, que se encarrega de ativar o NCPlayer no momento de apresentar o projeto em seu destino. O NCMaker conversa com o NCPlayer quando o usuário pede para que o projeto que ele está construindo seja apresentado (demonstração). O NCLogin trabalha em todas as etapas. Por exemplo, o NCClient precisa saber se o usuário a quem se destina o projeto está logado na máquina antes de ativar o NCPlayer. Maiores detalhes sobre a comunicação inter-processo dentro do NetComm serão vistos no capítulo 4 na seção referente ao protocolo de comunicação e no capítulo 5 seção 5.2.

3.7 Construindo um projeto - NCMaker


O NCMaker é o primeiro módulo que iremos ver, pois construir um projeto é o primeiro passo a ser dado por qualquer usuário do NetComm. O NCMaker é o módulo responsável pela fase de construção de projetos. Como foi dito na seção 3.1.3, existem 6

dimensões que compõem um projeto. Esta seção vai seguir essa divisão e a ordem apresentada é apenas didática não existindo uma sequência pré-definida de passos para a construção de um projeto.

Na figura 3.3 vemos a janela de abertura do NCMaker. Estão disponíveis os menus e uma caixa de ferramentas, ou barra de ícones como alguns a chamam. Dentro da janela do NCMaker já podemos ver a dimensão "O quê" do projeto (onde as linhas representam os objetos) e que será apresentada mais adiante.

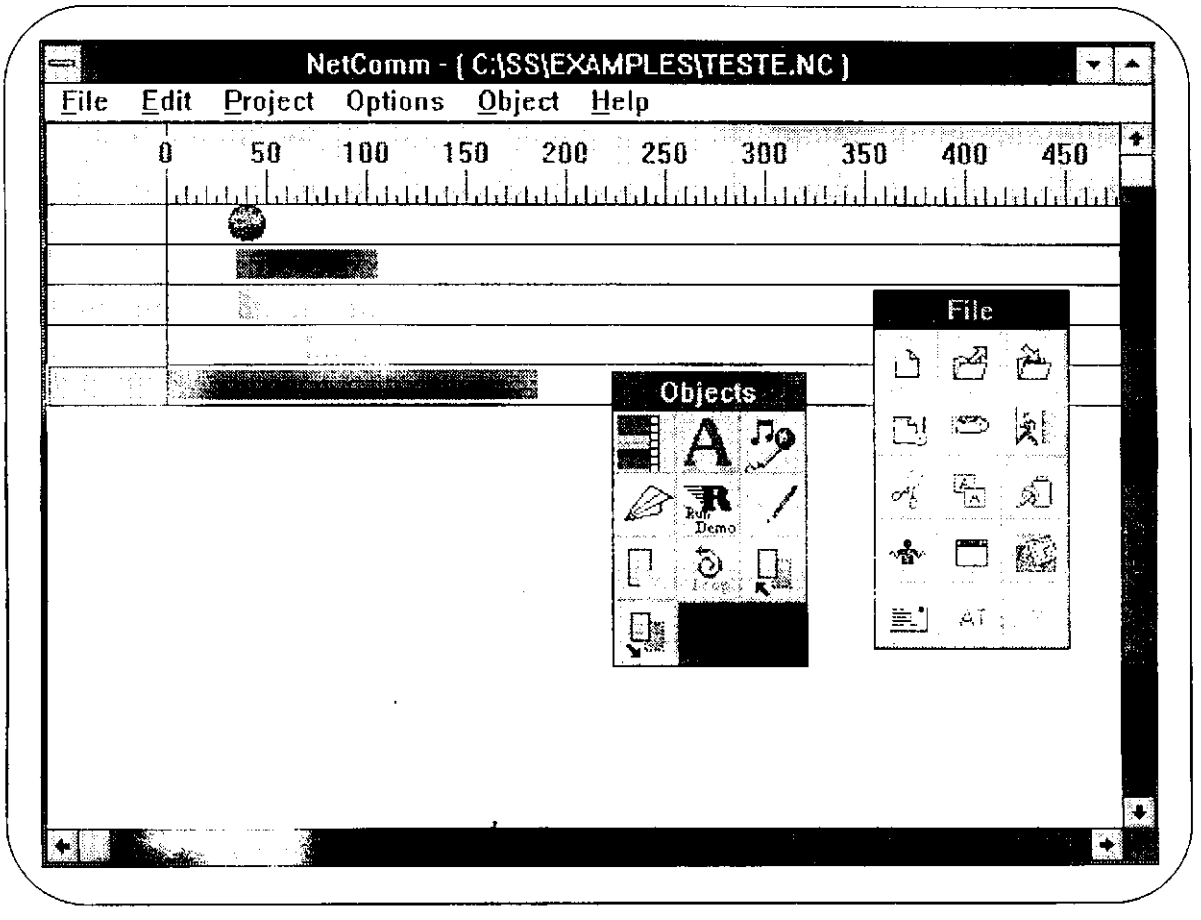

Figura 3.3 - Janela de abertura do NCMaker

Nesta seção nos deteremos às características e menus do NCMaker não relacionadas com as dimensões de um projeto. Isso porque teremos seções específicas para cada uma delas.

3.7.1 Caixa de ferramentas (*Tool Box*)

Apesar de não fazer parte da definição da interface Windows [MSOFT92a], a maioria das aplicações para Windows hoje, dispõe desse tipo de interação com o usuário. Qualquer um dos ícones dentro da caixa de ferramentas tem seu equivalente nos menus do NCMaker. A comodidade e a facilidade que o uso de ícones, ao invés de menus, traz é indiscutível, sem contar que é muito mais rápido posicionar o mouse em um ícone do que caminhar por diversos níveis de menus. Como toda boa aplicação para Windows, o NetComm disponibiliza essa facilidade para o usuário e permite que o mesmo configure a caixa de ferramentas com as funções mais frequentes que ele costuma acessar. As caixas de ferramentas do NetComm (*Object* e *File*) podem ser vistas na figura 3.3.

3.7.2 Menu *File*

O menu *File* têm as opções de tratamento de arquivos de projeto. A terminação dos arquivos do NetComm é .NC por definição, mas qualquer outra pode ser usada se o usuário assim desejar. A vantagem de se ter uma terminação de nomes de arquivo definida no ambiente Windows é que pode ser feita uma associação entre esses arquivos e uma aplicação. No caso do NetComm, o usuário pode escolher se a associação deve ser feita com o NCMaker ou com o NCPlayer, construção e apresentação respectivamente.

As opções dentro do menu *File* permitem que o usuário crie um novo projeto (New), abra um projeto existente (Open), salve um projeto aberto (Save), salve um projeto definindo o nome (Save As), Envie o projeto para tratamento pelo NCServer (Send), ou abandone o NCMaker (Exit). A opção Send ativa uma comunicação local com o NCServer para incluir o projeto criado na rede; esta opção será melhor explicada quando falarmos do NCServer e de detalhes de comunicação inter-processo nos capítulos 3 e 4.

Os menus de abertura e salvamento de arquivos são pré-definidos pelo Windows e as caixas de diálogo também. A figura 3.4 a seguir mostra uma dessas caixas de diálogo.

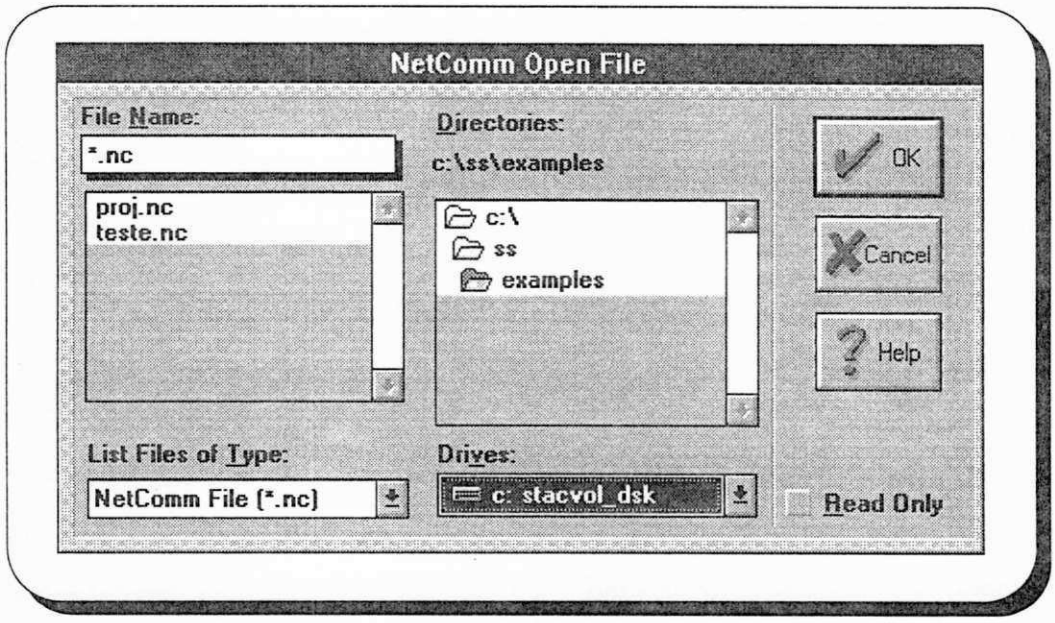

Figura 3.4 - Caixa de diálogo para abertura de arquivos

3.7.3 Menu *Edit*

As opções do menu *Edit* não são padronizadas pela interface Windows [MSOFT92a], mas quase todas as aplicações seguem um padrão, para felicidade dos usuários. Nesse menu estão as opções para acesso à área de transferência (*Clipboard*). O NetComm permite que qualquer um de seus objetos sejam colocados na área de transferência seja ele visual ou não. As opções do menu *Edit* são: recortar (*Cut*), copiar (*Copy*), Colar (*Paste*).

3.7.4 Menu *Project*

No menu *Project* estão as opções para acesso às caixas de diálogo de definição das dimensões de um projeto: "Como", "Quando", "Para quem". A dimensão "O quê" é apresentada na própria janela do NetComm. Além das opções para acesso às dimensões de um projeto estão os atributos de um projeto como *global loop* e a opção para pedir uma demonstração do que foi feito até agora (*demo*). Ao acionar a opção *demo* o NetComm simplesmente ativa o NCPlayer passando como parâmetro o projeto que está sendo editado.

3.7.5 Menu *Options*

Neste menu estão as opções de configuração do NetComm. O usuário pode definir os valores *default* para alguns atributos das dimensões de um projeto - cor da janela na dimensão "Como", por exemplo. Podem ser configurados também os diretórios onde normalmente devem ser buscados os arquivos gráficos, de texto e de som entre outras coisas.

3.7.6 Menu *Object*

Este menu é relativo à dimensão "O quê" do projeto. Como essa dimensão está sempre aberta para edição, existem opções neste menu para incluir objetos de todos os tipos, excluir e editar objetos, ativar laços para os objetos e mudar sua ordem de apresentação. Este menu será melhor detalhado na seção 3.7.8, ao apresentarmos a dimensão "O quê".

3.7.7 Menu *Help*

Praticamente toda aplicação Windows tem um menu de ajuda. O NetComm não foge à regra e através deste menu é possível ter acesso a um *help on-line* com hypertexto [BUGG93] onde estão detalhadas todas as características do NetComm. Caso se precise de ajuda para executar alguma ação, basta acessar este menu. A seguir descreveremos cada uma das dimensões que compõem um projeto, começando pela dimensão "O quê".

3.7.8 Definindo a dimensão "O quê"

A dimensão "O quê" de um projeto é responsável por definir que objetos serão apresentados, por quanto tempo, e em que momento eles deverão ser apresentados. Por apresentar um nível um pouco maior de complexidade e por apresentar informação relevante para o projeto como um todo decidimos que essa dimensão deveria ficar sempre disponível para o usuário. Portanto, ela se tornou a janela principal do NetComm (figura 3.3).

Na dimensão "O quê" de um projeto é que determinamos que objetos farão parte do mesmo. Existem quatro tipos básicos de objetos divididos em duas classes: visuais e não visuais.

Os objetos visuais são do tipo gráfico ou texto, já os objetos não visuais são do tipo som ou evento.

3.7.8.1 Acrescentando novos objetos a um projeto

Ao iniciar um novo projeto no NetComm não existe nenhum objeto definido. Cabe ao usuário determinar que objetos farão parte do projeto. A medida que esses objetos vão sendo criados eles vão sendo mostrados na janela como linhas de uma partitura. É justamente essa partitura que rege a apresentação do projeto.

Cada tipo de objeto é representado por uma cor diferente dentro da partitura. Assim, temos vermelho para objetos gráficos, verde para objetos texto, azul para objetos som e bege para objetos evento. Como podemos observar na figura 3.3, os objetos visuais são representados por barras de diversos tamanhos enquanto os não visuais são representados por esferas. O tempo de vida de um objeto visual é determinado pelo tamanho da barra.

Na parte superior da janela do NetComm existe uma régua. Essa régua serve como marcadora para indicar em que momento o objeto será apresentado e em que momento ele deixará de existir. A métrica da régua mede o número de passos de uma apresentação, não o tempo. Por exemplo, o segundo objeto da figura 3.3 começara a ser mostrado no passo 35 da apresentação e desaparecerá aproximadamente no passo 103. Caso desejemos saber com precisão em que passo se inicia e em que passo termina a apresentação de um objeto, podemos acessar a opção *Information* do menu *Object* ou o ícone correspondente na caixa de ferramentas. Ao fazermos isso, uma janela será aberta como podemos ver na figura 3.5. Para alterar o passo inicial de um objeto seja ele visual ou não basta posicionar o cursor do mouse sobre ele e arrastá-lo pela linha até a posição desejada.

Os objetos não visuais são representados por uma esfera porque eles ocupam apenas um passo da apresentação. Isso tem razão de ser, já que não é possível medir durante quantos passos um som será tocado. Caso o usuário disponha de uma placa de som, a apresentação do objeto som será paralela. Ou seja, enquanto o som está sendo tocado, os outros objetos estão sendo apresentados e animados se for o caso também. Caso o usuário use o alto-falante do PC como saída de som, a apresentação do som será sequencial. Isso quer dizer que enquanto o som estiver sendo tocado, nenhum passo a mais será dado na apresentação do projeto até que o som termine.

As placas de som possuem sua própria memória e UCP (Unidade Central de Processamento). Os *drivers*[7] dessas placas se aproveitam disso, fazendo com que a UCP da placa se encarregue de tocar o som enquanto a UCP do PC fica liberada para executar outras tarefas. Quando o alto-falante interno do PC é usado como saída de som, a UCP do PC é responsável por tocar o som e por garantir que o mesmo seja audível (velocidade ao enviar os pulsos para o alto-falante é imprescindível nesse caso) o *driver* ocupa totalmente a UCP. No caso de objetos do tipo evento a situação é a mesma. Enquanto o evento estiver sendo apresentado, nenhum passo será dado na apresentação do projeto. Estas são as razões para os objetos visuais ocuparem apenas um passo da apresentação. Maiores detalhes sobre *drivers* do Windows podem ser vistos em [PIETREK93].

Dentro da partitura existe o conceito de objeto corrente. É sobre esse objeto que serão efetuadas as operações de edição, remoção, movimentação, entre outras. Para selecionar um objeto basta posicionar o cursor sobre a linha que representa o objeto e pressionar o botão esquerdo do *mouse*. O objeto corrente é diferenciado dos demais por apresentar uma marca cinza escura no canto esquerdo da janela.

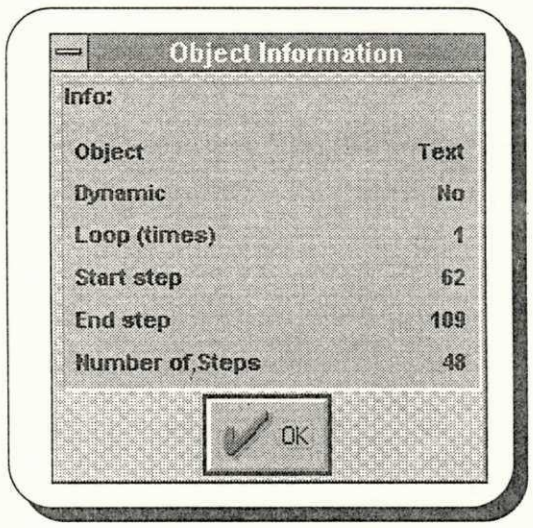

Figura 3.5 - Informações do objeto corrente

Ao incluir um objeto ele é colocado em uma nova linha no final da partitura e é imediatamente aberta a caixa de diálogo para definição de seus atributos. A ordem com que os objetos são colocados na partitura é importante para determinar, por exemplo, quem vai ficar

[7] Programa que controla dispositivos de *hardware*.

sobre quem no momento da apresentação no caso, obviamente, de objetos visuais. Isso é importante pois se, por exemplo, tivermos um objeto gráfico que representa uma parede e um objeto texto com uma mensagem qualquer, é interessante que o gráfico não sobreponha o texto.

O usuário pode mudar a ordem de sobreposição dos objetos da forma que desejar. Para isso ele dispõe das opções *To front* e *To back* no menu *Objects* e dos ícones correspondentes na caixa de ferramentas. A opção *Loops* permite que a apresentação do objeto corrente seja repetida um número determinado de vezes. Ao selecionarmos essa opção do menu *Objects* uma caixa de diálogo é aberta para possibilitar a entrada do número de vezes que queremos que a apresentação seja repetida.

3.7.8.2 Animação de objetos visuais

Ao incluirmos um objeto visual em nosso projeto, uma caixa de diálogo como a apresentada na figura 3.6 é aberta para que possamos definir alguns atributos do objeto. A caixa de diálogo para os objetos texto e som difere pouca coisa no que diz respeito à animação. O lado esquerdo da caixa de diálogo se mantém o mesmo nos dois casos, apenas o lado direito sofre alterações.

No lado esquerdo da caixa de diálogo podemos observar um retângulo branco cercado por duas barras de rolamento. Dentro dele há um retângulo maior que representa a janela definida na dimensão Como do projeto em tamanho proporcional ao retângulo externo. Na parte inferior do retângulo com as barras de rolamento temos um outro retângulo um pouco menor. Ele representa o objeto em tamanho proporcional aos outros dois. Ou seja, é o tamanho que o objeto ocuparia na janela definida na dimensão Como, proporcionalmente ao retângulo que representa a janela. Caso desejemos criar uma animação basta arrastar o objeto, usando o mouse com o botão esquerdo pressionado, e posicioná-lo onde o mesmo iniciaria seus movimentos. Para criar movimentos para o objeto basta arrastá-lo da mesma maneira descrita anteriormente porém com a tecla *Shift* pressionada. Desta forma estaremos criando vetores de animação por onde o objeto se movimentará. Por exemplo, na figura 3.6 há uma animação definida para um objeto do tipo texto. Esse objeto iniciará seus movimentos fora da janela, em sua parte superior, e desaparecerá pelo lado inferior da mesma. Todo o trajeto que será percorrido pelo objeto é representado por uma

seta. Essa forma de criar animações é muito intuitiva, tendo sido testada com sucesso por vários usuários do sistema antes de ser adotada.

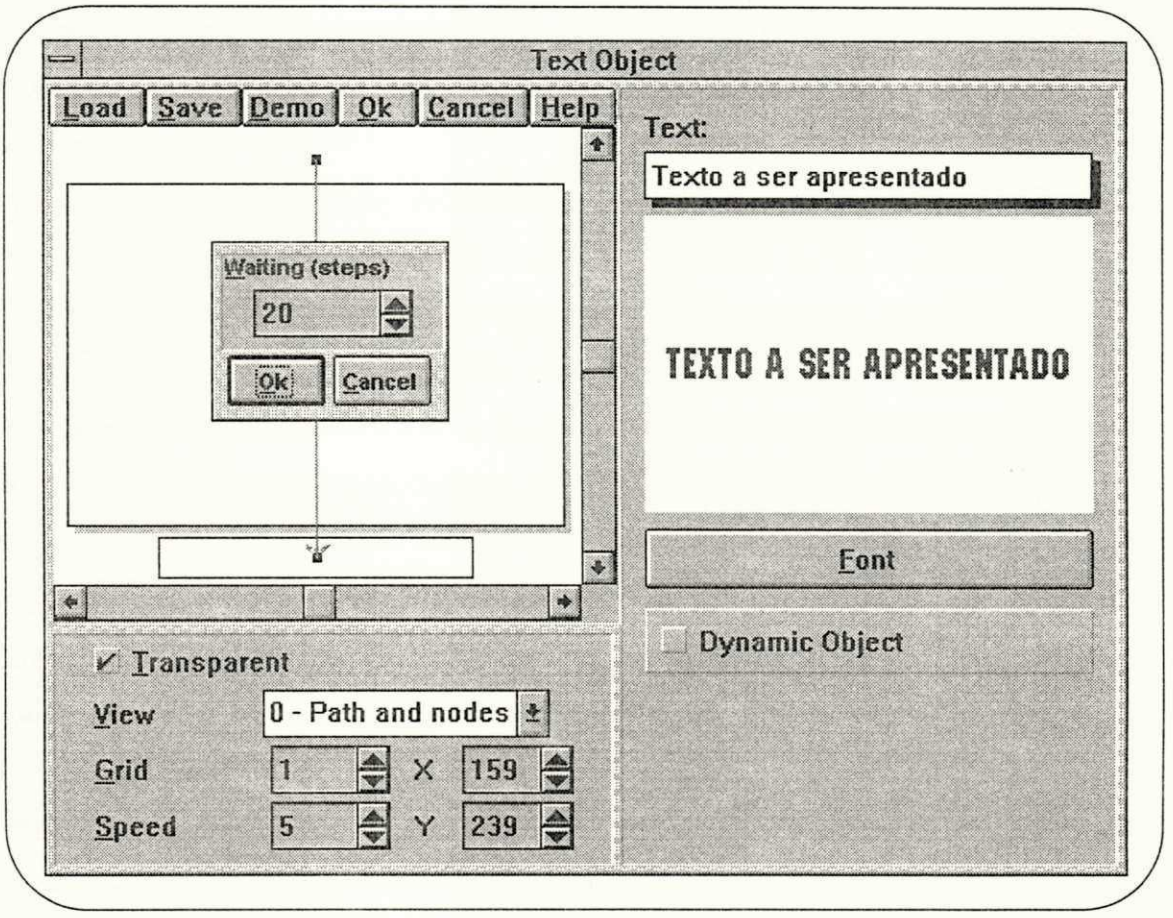

Figura 3.6 - Definindo animações para objetos visuais

Para cada vetor de animação construído existe um nó de interligação com os demais vetores. Caso o usuário deseje alterar a trajetória de um vetor já definido, basta ele se posicionar com o mouse sobre o nó e arrastá-lo conforme já foi explicado. Para cada nó é possível determinar um tempo de espera medido em número de passos. Por exemplo, na figura 3.6 podemos observar que um tempo de espera de 20 passos está sendo definido para um nó no centro da janela. O objeto então sairá da parte superior da janela, ficará parado durante 20 passos no centro e depois sairá da janela pela sua parte inferior.

Para que o usuário possa determinar com precisão a posição dos objetos, existem dois *spin-boxes* (X e Y) que permitem ajustes finos. Um *grid* pode ser utilizado caso não se queira

uma precisão tão grande. Existe ainda um *spin-box* para que o usuário possa determinar a velocidade com que o objeto se movimentará pela janela.

O atributo de transparência de um objeto indica se desejamos que ao passar sobre outros objetos, a cor de fundo do mesmo não sobreponha o de baixo produzindo um efeito indesejado. Esse atributo não é recomendado para grandes objetos gráficos, pois representa uma perda considerável de velocidade na apresentação. A cor de transparência para objetos gráficos é a negra. Ou seja, as partes negras de um objeto gráfico não serão apresentadas, só as demais cores.

3.7.8.3 Construindo objetos do tipo texto

Um dos tipos de objeto que o NetComm usa é o texto. Tanto podem ser pequenas mensagens como documentos completos. Para incluir um objeto do tipo texto em uma apresentação basta acionar a opção *New Text* do menu *Object*. Uma caixa de diálogo como a da figura 3.6 será apresentada.

Para incluir o texto que será apresentado basta posicionar-se no campo de edição *Text* e digitar a mensagem desejada. Caso o texto faça parte de um arquivo, basta colocar seu nome nesse campo e marcar o atributo de dinâmico que já foi explicado na seção 3.4. O texto do arquivo será usado na apresentação do projeto. O usuário pode também selecionar o tipo de fonte que será usado e qual a cor com que o texto deverá ser apresentado.

Um objeto do tipo texto pode ser animado. Isto é ele pode se movimentar por toda a janela definida na dimensão Como. Para definir uma animação para o objeto texto deve-se seguir o procedimento explicado na seção 3.7.8.2.

3.7.8.4 Construindo objetos do tipo gráfico


O primeiro passo para se incluir objetos do tipo gráfico em um projeto é ter o gráfico propriamente dito. O NetComm aceita gráficos no padrão *bitmap* do Windows (arquivos com terminação .BMP). Para construir gráficos usando esse padrão pode ser utilizado qualquer utilitário para criação de gráficos como o Corel Draw, o Lotus 123, o Word for Windows, o Paint Brush ou o que melhor se adequar ao tipo de gráfico a ser gerado. Para incluir um determinado

bitmap em um projeto basta acionar a opção New Graphic do menu Object ou simplesmente pressionar o botão do mouse sobre o ícone correspondente. A caixa de diálogo para a definição do objeto gráfico será aberta como mostra a figura 3.7.

Para construir uma animação para o objeto gráfico deve-se seguir o procedimento explicado na seção 3.7.8.2. Existe porém, um outro tipo de animação que pode ser definida para objetos gráficos: a animação interna.

A animação interna de um objeto gráfico permite que se definam até cinco quadros que serão intercalados à medida que o objeto se movimenta. Esses quadros permitem que um objeto aparente mudar de forma enquanto se movimenta pela janela. Na figura 3.7 podemos observar 5 quadrados no canto inferior direito da caixa de diálogo. Usamos esses cinco quadrados para definir a animação interna do objeto gráfico. Caso usemos apenas um dos quadrados o objeto não terá animação interna.

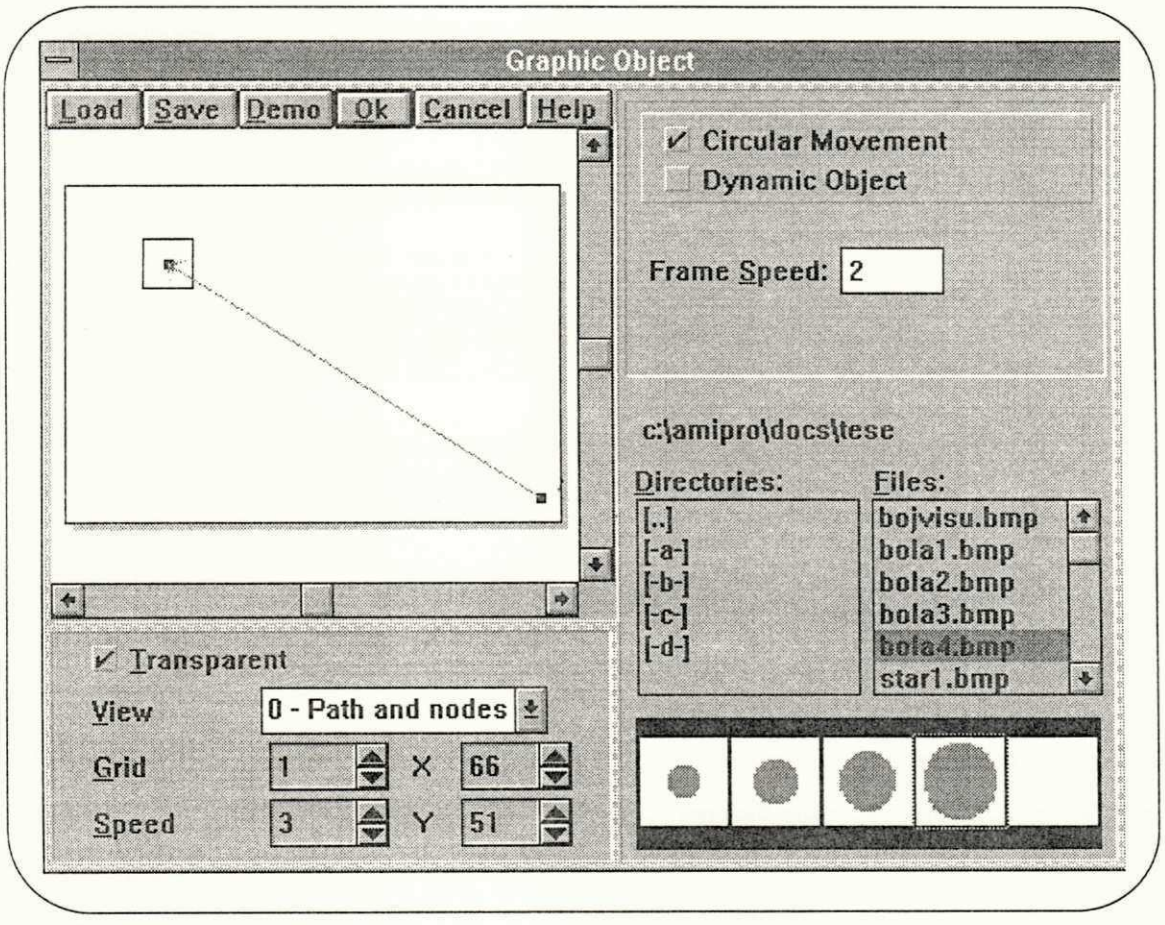

Figura 3.7 - Caixa de diálogo para definição de um objeto gráfico

Para determinar quais os gráficos do tipo *bitmap* que farão parte da animação interna do objeto gráfico basta posicionar-se no quadrado correspondente à sequência de animação e escolher um arquivo com terminação .BMP usando as listas de seleção *Directories* e *Files*. No exemplo da figura 3.7 usamos 4 gráficos (bola1.bmp, bola2.bmp, bola3.bmp e bola4.bmp) para compor a animação interna do objeto. Ao serem intercalados eles darão a sensação para o observador de uma bola crescendo e decrescendo. Maiores informações sobre animação de *bitmaps* podem ser vistas em [ADAMS93].

O atributo *Circular Movement* serve para alterar a forma com que os cinco gráficos da animação interna serão intercalados. Caso este atributo esteja marcado, os gráficos serão apresentados do primeiro até o último e depois do último ao primeiro e assim sucessivamente. Caso o atributo *Circular Movement* não esteja marcado os gráficos serão apresentados sempre do primeiro ao último sucessivamente. Na figura 3.8 podemos ver as diferenças entre as duas sequências de apresentação usando os *bitmaps* do exemplo anterior.

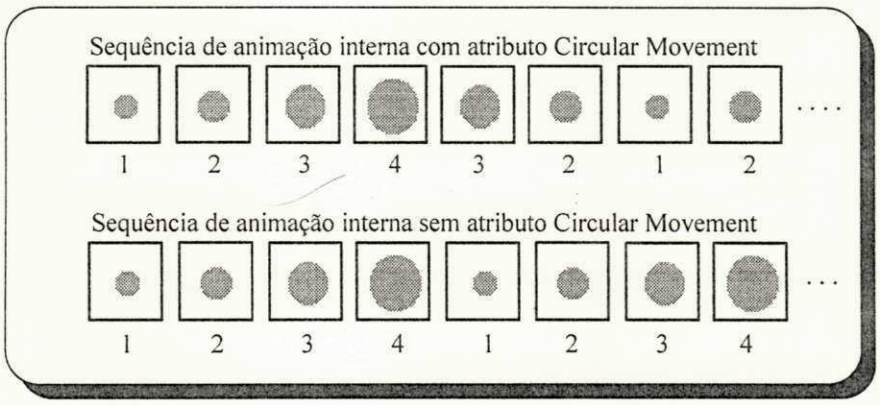

Figura 3.8 - Sequência de animação interna X atributo *Circular Movement*

3.7.8.5 Construindo objetos do tipo som

Hoje em dia é cada vez mais comum encontrarmos PCs com recursos multimídia como: CD-ROM, placas de som, alto-falantes e microfones. Os próprios fabricantes de *hardware*, atendendo o mercado, estão lançando PCs que já incorporam esses recursos a preços cada vez menores. *Hardware* e *software* com capacidades multimídia são uma tendência

mundial, e como não poderia deixar de ser, esse recurso foi incorporado ao NetComm na forma de objetos tipo som.

A Microsoft tem um padrão para o armazenamento de sons em arquivos: os arquivos com terminação .WAV (de *wave*, onda em inglês). O NetComm usa esses arquivos para possibilitar a comunicação entre usuários e a construção de projetos no NetComm. Os arquivos .WAV podem ser gerados por quem tem uma placa de som como a Sound Blaster usando, por exemplo, o microfone.

Para incluir objetos do tipo som em um projeto NetComm basta simplesmente acionar a opção *New Sound* do menu *Object* do NCMaker. Uma caixa de diálogo como mostrada na figura 3.9 será aberta para a definição dos atributos do objeto criado. Para escolher o som a ser apresentado basta posicionar-se no diretório desejado usando a lista de seleção *Directories* e marcar o arquivo .WAV desejado na lista *Files*.

Uma particularidade de objetos tipo som diz respeito ao cuidado que se deve ter ao incluir laços. Caso a máquina a qual se destina o projeto não disponha de placa de som e use o alto falante do PC para tocar os sons, eles serão tocados de forma síncrona. Isso quer dizer que o Windows pára até que o arquivo de som seja completamente tocado. Colocando laços no objeto podemos simplesmente parar a máquina destino e não dar oportunidade do usuário retomar o controle da aplicação com facilidade.

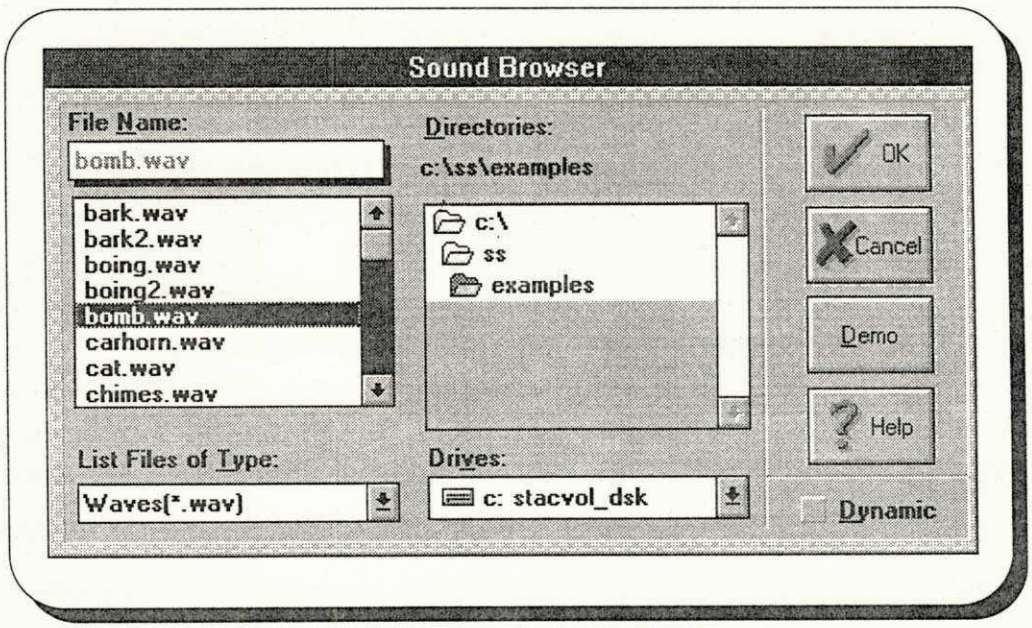

Figura 3.9 - Definindo um objeto tipo som

3.7.8.6 Construindo objetos do tipo evento

Quem já usou um gravador de macros como os que acompanham a maioria dos aplicativos para Windows vai se sentir familiarizado com objetos do tipo evento. A idéia básica é a mesma em todos eles: gravar uma sequência de eventos de mouse e teclado para ser posteriormente reproduzida. Objetos do tipo evento são também sequências de eventos de mouse e teclado gravados para posterior reprodução. A diferença é que com o NetComm, os eventos podem ser reproduzidos remotamente e que é permitida a abertura de um aplicativo que irá receber esses eventos.

Figura 3.10 - Caixa de diálogo para definição de um objeto tipo evento

Para incluir um objeto do tipo evento em um projeto basta acionar a opção New Event do menu Object do NCMaker ou clicar no ícone correspondente na caixa de ferramentas. Uma caixa de diálogo como apresentada na figura 3.10 será aberta. O usuário pode então determinar o nome da aplicação a ser aberta, se existir, e o nome do arquivo onde será gravada a sequência de

eventos também se existir. Para iniciar a gravação de uma sequência de eventos basta pressionar o botão Recorder. o NCMaker será minimizado e se foi definida uma aplicação a ser aberta ela o será. A partir desse momento todos os eventos de mouse e teclado serão gravados e para finalizar o processo basta clicar sobre o ícone do NCMaker.

Os objetos do tipo evento são os mais difíceis de entender e os que permitem que o NetComm possa ser utilizado em, por exemplo, tarefas de gerência de redes ou de automatização de tarefas. O poder que esse tipo de objetos dá ao usuário pode ser melhor compreendido na seção 3.5 destinada a aplicações do NetComm.

3.7.9 Definindo a dimensão "Como"

A dimensão "Como" de um projeto define a janela em que os objetos visuais serão apresentados. A janela definida será aberta pelo NCPlayer no momento da apresentação do projeto. Para definir a dimensão "Como" de um projeto basta acionar a opção When do menu Project do NCMaker. Será então aberta uma caixa de diálogo como mostrada na figura 3.11.

Na caixa de diálogo é possível definir a posição (x,y) e a largura e altura da janela. Também podemos definir o título da janela bem como o estado em que ela será mostrada: maximizada, minimizada, normal ou escondida.

Uma janela escondida é útil para a construção de automatizadores que executam em *background*. Ao definirmos a janela como escondida, o NCPlayer executará mas a janela não ficará visível para o usuário não atrapalhando seu trabalho. Isso, aliado ao fato de incluirmos objetos do tipo evento permite que, por exemplo, um backup seja executado automaticamente sem atrapalhar a atividade normal do usuário.

É possível ainda, determinar as cores de fundo e do texto para a janela usando a palheta de cores disponível. Quando o atributo *Fixed Place* está marcado, a janela será apresentada nas coordenadas especificadas. Quando esse atributo não está marcado o NCPlayer se encarrega de escolher uma posição para apresentar a janela.

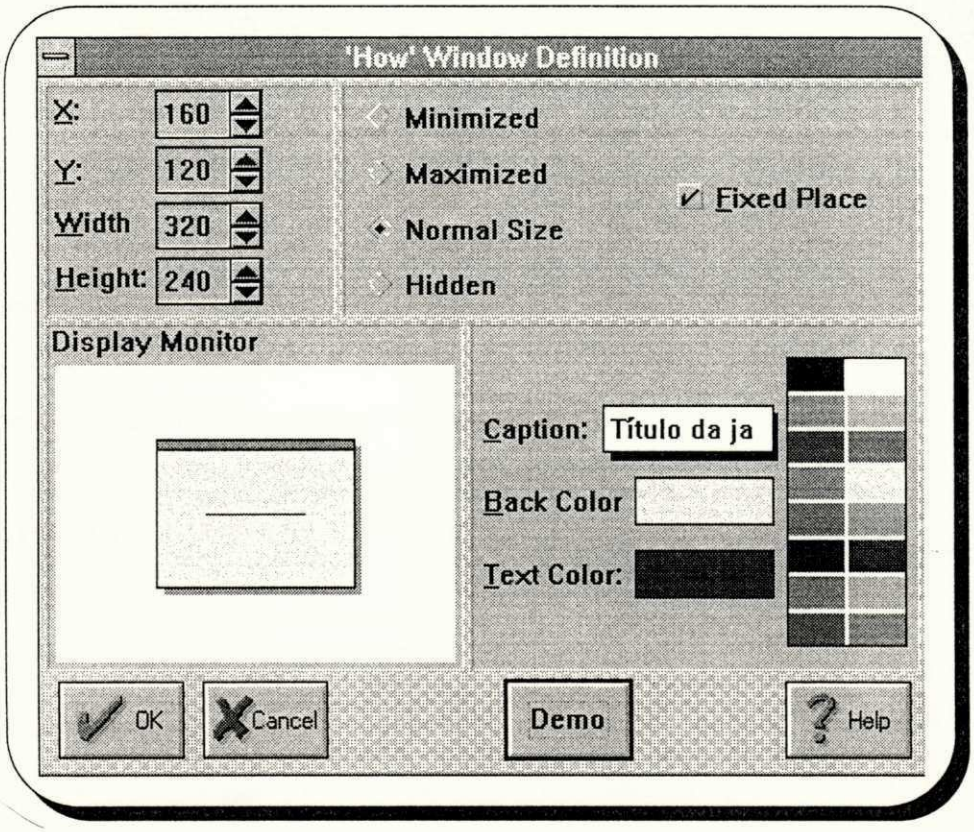

Figura 3.11 - Caixa de dialogo para definição da dimensão Como de um projeto

O botão Demo pode ser usado para definir a janela de forma interativa. Ao pressionar esse botão a tela fica escura e a janela é representada em seu tamanho natural. Ela pode ser então posicionada e ter suas dimensões alteradas como se fosse uma janela comum do Windows. Ao fechar essa janela é re-apresentada a caixa de diálogo de definição da dimensão Como com os novos valores refletindo as modificações efetuadas.

3.7.10 Definindo a dimensão "Quando"

A dimensão "Quando" de um projeto determina em que momento o mesmo deve ser apresentado. Existem três formas diferentes de apresentar um projeto segundo a dimensão "Quando": por escalonamento, por tempo de inoperação e imediatamente. Para definir a dimensão "Quando" de um projeto, basta acessar a opção *When* do menu *Project*.

Um projeto imediato significa que a partir no exato momento em que o usuário disparar o projeto na rede o mesmo será apresentado na máquina ou máquinas destino. Esse tipo de projeto é útil quando, por exemplo, um usuário deseja se comunicar com outro enviando uma simples

mensagem. Outro uso poderia ser do administrador tentando avisar os usuários de alguma ocorrência como por exemplo, desligamento do sistema. Nesse caso, a mensagem poderia ser enviada para todas as máquinas da rede.

Um projeto com a dimensão "Quando" definida como tempo de inoperação tem características bem interessantes. Ao definirmos a dimensão "Quando" desta forma estamos dizendo que o projeto só será apresentado após um determinado período de tempo em que o usuário destino não executa atividades relacionadas com o mouse ou o teclado. Ou seja, o projeto só será mostrado no momento em que o usuário ficar algum tempo sem fazer nada. Essa forma de apresentar um projeto é interessantíssima pois não atrapalha o trabalho do usuário no momento em que ele será apresentado. Essa é a maneira com que os programas do tipo *screen-saver* trabalham.

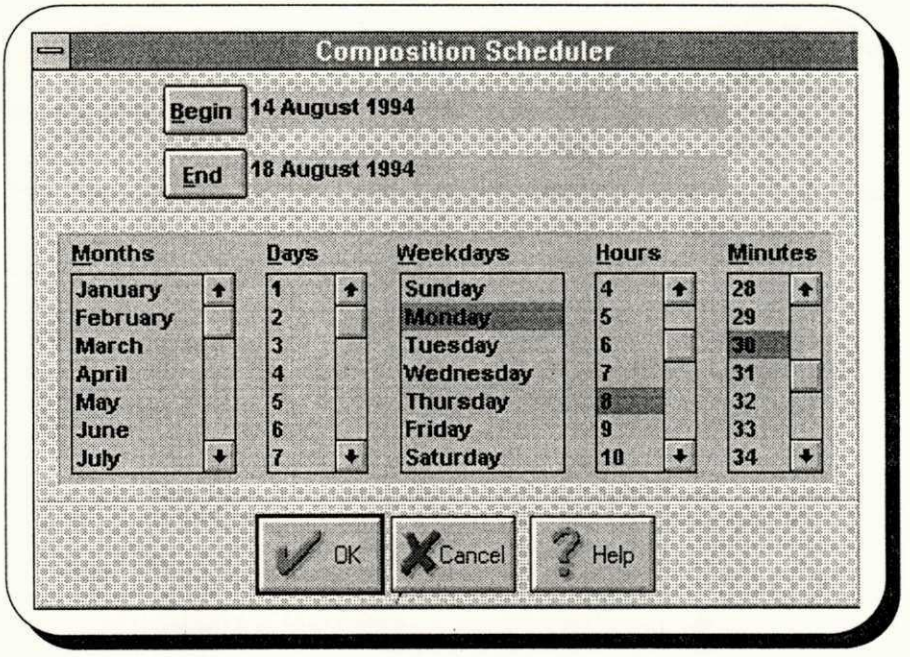

Figura 3.12 - Definindo a dimensão Quando de um projeto

Um projeto por escalonamento permite que se determine o exato momento ou momentos em que o mesmo será apresentado. Na caixa de diálogo para definição da dimensão "Quando" por escalonamento (figura 3.12) existem 5 listas de seleção múltipla para selecionar os meses, dias, dias da semana, horas e minutos respectivamente. No exemplo dado, o projeto seria apresentado todas as segundas-feiras às 8:30. As listas são de seleção múltipla, logo é possível

marcar mais de um elemento da mesma lista. Por exemplo, Segunda (*Monday*) e Quarta (*Wednesday*) na lista de dias da semana.

Os botões *Begin* e *End* abrem uma outra caixa de diálogo representando um calendário. Através deles é possível determinar a data de inicio e data final em que o projeto será apresentado. Ou seja, o projeto só terá validade durante o período estabelecido entre essas duas datas.

3.7.11 Definindo a dimensão "Para quem"

Como o próprio nome diz, é nesta dimensão do projeto que determinamos para quem se destina o projeto. A escolha é feita montando uma lista de pares ordenados da forma (nome do usuário, nome da máquina). Ou seja, se quisermos enviar um projeto para dois determinados usuários e para ser apresentado em uma determinada máquina, dois elementos devem ser incluídos na lista: (usuário1, máquina1) e (usuário2, máquina2).

Figura 3.13 - Caixa de diálogo para definição da dimensão Para quem de um projeto

Na dimensão Para quem existe o conceito de usuário e máquina **qualquer**. Ou seja, se quisermos que o usuário "Marcos" receba um determinado projeto em qualquer máquina da rede na qual o mesmo se logar, a dimensão "Para quem" precisa identificar apenas um elemento: (Marcos, Qualquer). Para construir um projeto que seja apresentado em qualquer máquina da rede e para qualquer pessoa a dimensão "Para quem" deve ser definida com o elemento: (Qualquer, Qualquer). A caixa de diálogo usada para definir a dimensão "Para quem" de um projeto é mostrada na figura 3.13.

3.8 Distribuindo um projeto

A segunda fase pela qual passa um projeto é a de distribuição. Nesta fase, dois módulos do NetComm são envolvidos: NCServer e NCClient. Eles são os responsáveis pela comunicação remota e pela troca de projetos e dados.

Tanto o NCServer quanto o NCClient devem estar sempre ativos. Para isso, eles são incluídos no grupo StartUp do Windows no momento da instalação do NetComm. Desta forma, sempre que uma sessão do Windows é iniciada eles são carregados automaticamente.

A arquitetura cliente-servidor do NetComm trás vantagens para o administrador da rede, já que ele pode escolher quem vai ter acesso ao NCServer e ao NCMaker e quem só terá acesso ao NCPlayer. Por exemplo, em uma determinada máquina pode existir apenas o NCClient, o NCLogin e o NCPlayer o que significaria que essa máquina estaria apta a apenas receber e apresentar projetos.

Os módulos NCServer e NCClient dispõem de uma interface simples com o usuário para permitir a gerência dos projetos bem como o espaço em disco ocupado pelos mesmos. Os dois módulos são descritos nas próximas duas seções.

3.8.1 O NCServer

Ao terminar a fase de construção de um projeto, ele deve ser enviado para as máquinas destino como determinado na dimensão Para quem do projeto. A partir desse momento

quem passa a tomar conta do projeto é o NCServer. Ele se encarrega de tirar uma cópia do arquivo de projeto e guardá-la em um diretório específico, sob seu controle.

Quando um projeto novo é recebido pelo NCServer ele se encarrega de avisar os clientes de sua existência. Os clientes então pedem ao NCServer que envie o projeto caso o mesmo se destine a eles.

Em princípio o usuário não precisaria tomar conhecimento da existência do NCServer e/ou do NCClient, tanto é que eles executam iconizados e poderiam até ficar escondidos, mas existem determinadas situações em que ele precisa interagir com os mesmos. Um exemplo é quando o usuário não quer mais que sua máquina seja servidora de um projeto. Nesse caso ele usa o NCServer para matá-lo. Ou seja, o NCServer não mais enviará dados para os clientes referentes ao projeto em questão. Outra função que pode ser ativada a partir do NCServer é a de auditoria. Pode-se definir um arquivo de log para que todas as ações tomadas pelo NCServer sejam registradas.

Como podemos ver na figura 3.14, a interface do NCServer é bastante simples. Na janela principal podem ser observadas informações de log, como por exemplo a data e hora em que um projeto ou dado foi solicitado ou enviado.

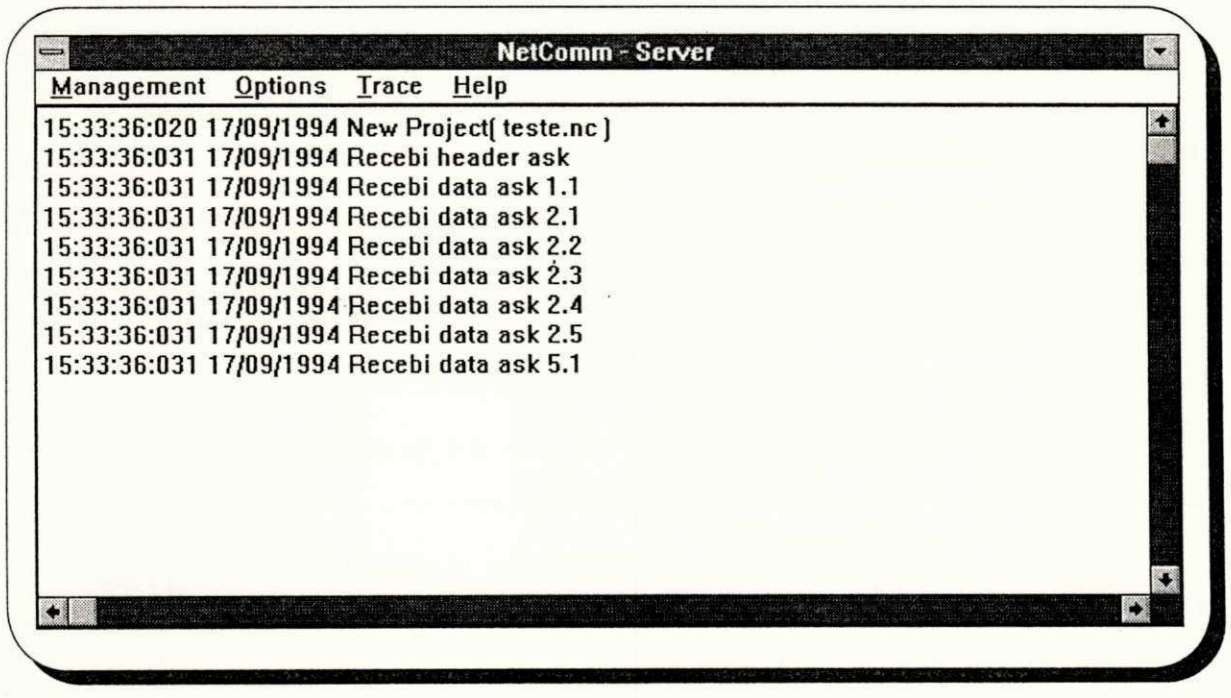

Figura 3.14 - Interface do NCServer

A partir do NCServer, o usuário pode descobrir quais são os Clientes e/ou Servidores ativos na rede e quem é o usuário que está logado nas máquinas. Isso é útil quando desejamos construir um projeto imediato e não temos certeza se o destinatário está logado nem em qual das máquinas da rede.

Outra função do NCServer diz respeito à gerência de projetos. O NCServer permite que o usuário mate um projeto antes que o tempo de vida do mesmo expire. O NCServer pode ser usado também quando desejamos transferir um projeto para que outra máquina faça o papel de servidora daquele projeto se por exemplo, a máquina onde ele foi construído estiver sobrecarregada.

3.8.2 O NCClient

A exemplo do NCServer, o NCClient dispõe de uma interface com o usuário muito simples que esconde a complexidade da tarefa a que o mesmo se destina. São duas as atividades básicas do NCClient: solicitar e receber dados e projetos e escalonar projetos para serem apresentados pelo NCPlayer. Também a exemplo do NCServer, o NCClient permite que as atividades relacionadas à troca de dados com o servidor sejam monitoradas e documentadas em um arquivo de log.

A interface do NCClient é praticamente idêntica à do NCServer e ele executa normalmente de forma minimizada. Ele tem em sua janela principal informações sobre as atividades monitoradas e sobre pedidos feitos pelo usuário (nome dos projetos a disposição do NCClient, por exemplo). Através de seus menus o usuário pode executar algumas tarefas de gerência, como determinar o percentual do espaço em disco que será destinado ao armazenamento de projetos ou desativar temporariamente ou definitivamente a apresentação de um determinado projeto. Caso se determine um percentual de uso do disco pelo NCClient, ele se encarregará de descartar os projetos que não forem para o usuário correntemente logado solicitando-os novamente quando se fizer necessário. A vantagem de não se descartar os projetos destinados a outros usuários que não o correntemente logado é a diminuição do tráfego na rede. O preço a ser pago é o maior consumo de espaço em disco.

O NCClient é ativado a partir do grupo StartUp[8] do Windows e sempre que é executado ele se comunica com os servidores requisitando os projetos que são destinados ao usuário correntemente logado. Os projetos recebidos podem ter dados associados (*bitmaps*, por exemplo) e é tarefa do NCClient buscar esses dados no servidor. Os dados só são solicitados no momento de apresentação do projeto se isso for necessário. Dados dinâmicos são solicitados sempre que o projeto for apresentado; por isso deve-se ter cuidado com esse tipo de dado se o período entre uma apresentação e outra for muito curto. Essa busca constante por atualizações dos dados pode gerar um tráfego intenso na rede tornando lento o acesso à mesma e prejudicando os demais usuários.

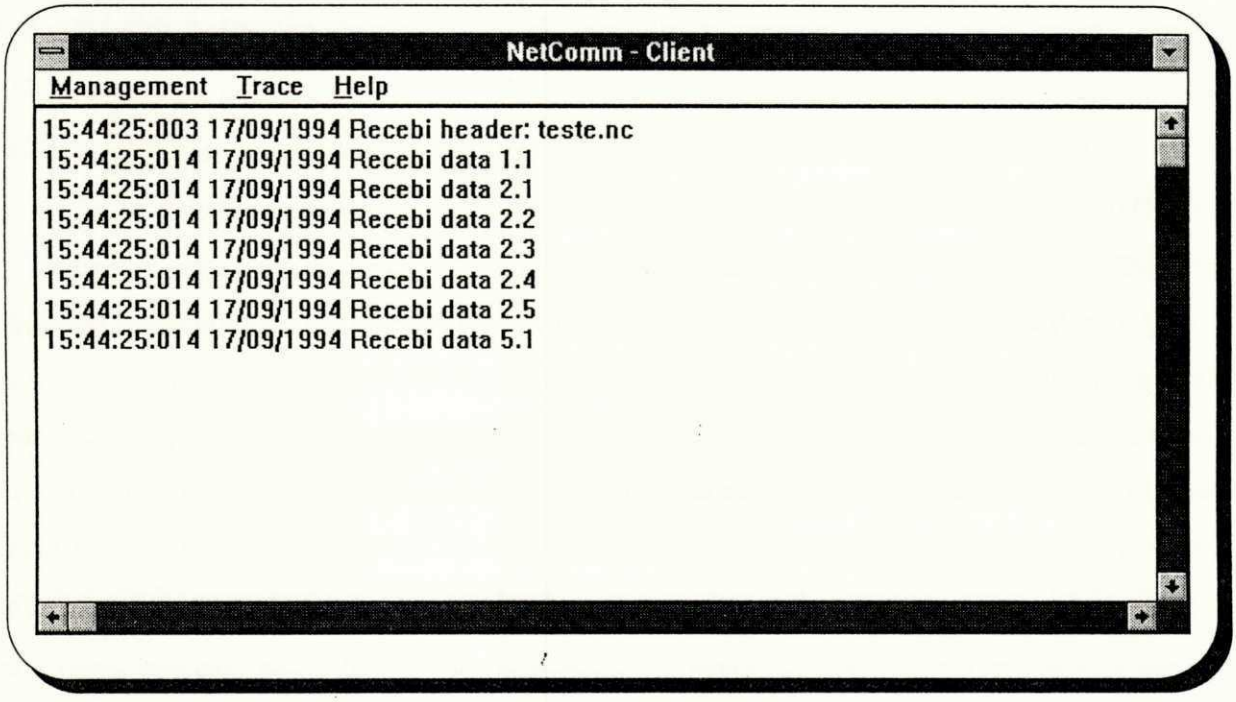

Figura 3.15 - Interface do NCClient

Em princípio, todos os projetos destinados ao usuário correntemente logado serão apresentados. Caso ele deseje interromper temporariamente a apresentação de um projeto (digamos que o usuário tenha cinco projetos por tempo de inoperação distintos) ele pode desativá-lo através do menu Management do NCClient. Essa interrupção só será válida durante a sessão corrente do Windows pois se o mesmo for re-inicializado o NCClient não tomará

[8] As aplicações pertencentes a esse grupo são executadas automaticamente quando o Windows é inicializado.

conhecimento da desativação. Caso o usuário queira interromper a apresentação do projeto indefinidamente, ele deve matar o projeto. Na figura 3.15 podemos observar a interface do NCClient.

3.9 Apresentando um projeto - NCPlayer

O NCPlayer é o módulo responsável por apresentar os projetos para o usuário na máquina destino. O NCPlayer é o único dos módulos do NetComm que não tem interface com o usuário. Ou melhor dizendo, a interface com o usuário é a janela que foi definida na parte Como do projeto que está sendo apresentado. A janela do NCPlayer aparecera como foi definida pelo projeto. Se a janela tiver sido definida como escondida, o usuário não tomará conhecimento visual do momento da execução do NCPlayer.

O NCPlayer é ativado sempre pelo NCMaker (através do botão Demo) ou pelo NCClient no momento em que o projeto deve ser apresentado (de acordo com o que foi definido na dimensão Quando) e ele encerra sua execução quando o projeto termina de ser mostrado. Caso o projeto contenha um laço (*loop*) global (o que implica em executar indefinidamente) o usuário pode interromper sua execução simplesmenta clicando o botão do *mouse* quando o cursor estiver sobre a janela. Muito cuidado portanto deve ser tomado ao construir projetos com laço global e janela escondida. O projeto pode ficar executando indefinidamente sem que o usuário tome conhecimento e só poderá ser interrompido usando o utilitário *Task List*[9] do Windows.

3.10 Login/Logout do NetComm

A forma de autenticação de usuários em uma rede varia dependendo de sua arquitetura. Segundo essa ótica, dividimos as redes em duas classes. Na primeira classe o usuário se identifica para a rede, ou seja, o processo de login é na rede e a partir desse momento ele têm acesso aos recursos da mesma. Na segunda classe, o usuário se identifica para a aplicação

[9] O utilitário *Task List* apresenta as aplicações que estão ativas no ambiente Windows.

distribuída e passa a acessar os recursos da mesma. Simplificando, na primeira classe o usuário se loga na rede e na segunda classe o usuário se loga na aplicação. Como exemplos da primeira classe podemos citar as redes Windows for Workgroup e Netware. Como exemplo da segunda classe podemos citar as redes TCP/IP. O NetComm foi projetado inicialmente para redes TCP/IP, nas quais o usuário se loga na aplicação, portanto o NetComm tem seu próprio esquema de autenticação e cadastro de usuários que está a cargo do módulo NCLogin.

Sempre que o NCServer ou o NCClient é ativado e caso não haja nenhum usuário logado o módulo NCLogin é ativado. Isso é feito porque o usuário só passará a receber os projetos no momento que ele se identificar para a aplicação.

A interface do módulo NCLogin é bastante simples, uma janela com um campo para o nome do usuário e outro para sua senha como podemos ver na figura 3.16. Caso o nome do usuário digitado não esteja cadastrado, o NCLogin pergunta se um novo usuário deseja ser cadastrado e em caso positivo pede-se a confirmação da senha. Essa forma de cadastro de usuários é idêntica à utilizada pelo *Windows for Workgroups* [BOYCE92] e é evidentemente pouco segura. Preocupações com segurança não é o tema principal deste trabalho, portanto decidimos não nos deter nessa parte e adotamos o mesmo processo do *Windows for Workgroups*.

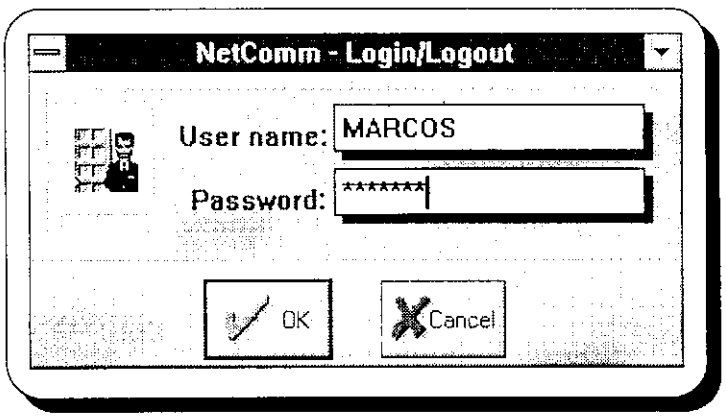

Figura 3.16 - Janela principal do NCLogin

O NCLogin armazena os dados sobre os usuários (nome e senha) em um arquivo criptografado no diretório de instalação do mesmo. O NetComm não usa um serviço de nomes distribuido, portanto o usuário precisa se cadastrar em todas as máquinas da rede para acessá-lo. Existe um usuário que é cadastrado no momento da instalação do NetComm. Esse usuário, cujo

nome é NCADMIN tem poderes especiais relativos à distribuição dos projetos que serão vistos no capítulo 4 na seção 4.5 referente à segurança. A senha desse usuário também é solicitada no momento da instalação, podendo ser deixada em branco caso não se deseje restringir o acesso aos recursos do NetComm.

Capítulo 4 - Implementação do NetComm

O NetComm é um *software* bastante complexo. A idéia aparentemente simples esconde grandes dificuldades na sua implementação. Em primeiro lugar o NetComm é um *software* desenvolvido para a interface gráfica Windows. Quem já teve oportunidade de programar sob esse ambiente conhece as dificuldades impostas pelo mesmo devido à herança do DOS. Em segundo lugar, o NetComm é um *software* distribuido e consequentemente, repleto de problemas distribuidos. A própria depuração de um *software* distribuido é muito mais complicada devido a problemas de sincronismo e *time-out*. Em terceiro lugar, o NetComm usa recursos de multimídia e o grande volume de dados envolvidos no armazenamento de arquivos gráficos ou de voz requerem um código otimizado ao máximo. Não basta funcionar, tem que ser eficiente.

Neste capítulo falaremos de como o NetComm foi implementado, descrevendo cada um dos seus módulos sob a perspectiva do desenvolvedor. Gostaríamos de salientar que por se tratar de um software complexo, a fase de codificação do NetComm foi levada à cabo por um grupo de pessoas[10] (incluindo-me) por mim lideradas sem as quais não teria sido possível concluir este trabalho no tempo previsto. As fases pelas quais passa um projeto (criação, distribuição e apresentação) serão utilizadas como forma de organização do capítulo e uma seção será dedicada a problemas de segurança. As estruturas de dados, classes, protocolos envolvidos serão nosso assunto no transcorrer das próximas seções. Iniciaremos o capítulo 4 apresentando uma visão geral dos módulos do NetComm.

4.1 Uma Visão Geral do NetComm

O NetComm está baseado na arquitetura Cliente-servidor e é dividido em cinco módulos básicos. Na figura abaixo, podemos ver a forma como esses módulos se relacionam entre si sob uma perspectiva de comunicação inter-processo. Ao avançarmos na leitura do capítulo, esses módulos serão explodidos.

---

[10] O grupo contou com a colaboração de Adriano Sérgio e de Alessandro Jatobá além de mim.

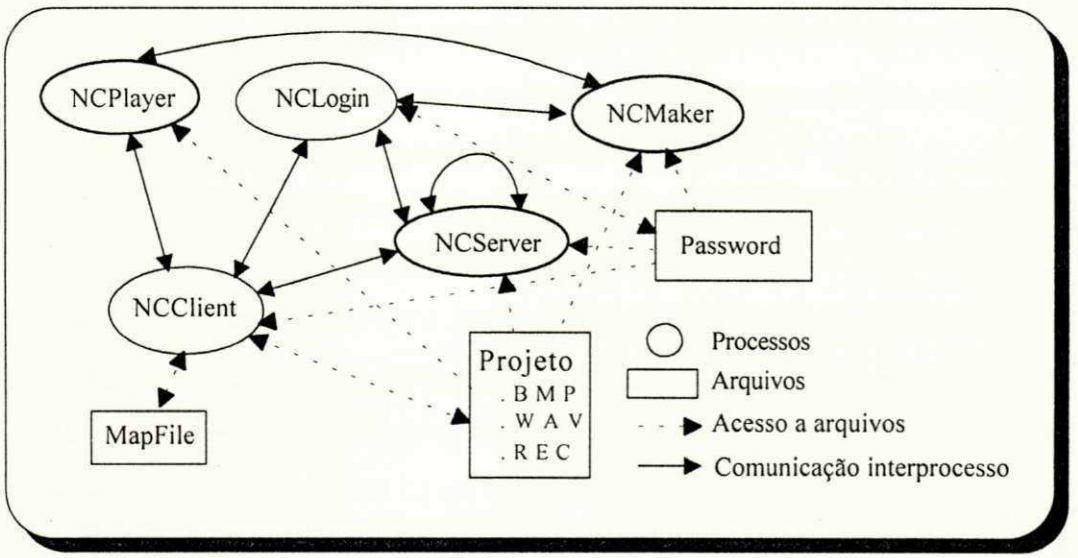

Figura 4.1 - Processos e arquivos do NetComm

4.2 Criação de um projeto (NCMaker)

O NCMaker é o módulo envolvido na fase de criação de um projeto. O principal objeto manipulado pelo NCMaker é o projeto que por sua vez é composto por diversas dimensões diferentes: O Quê, Como, Quando, Para quem, Por quem e Máquina servidora. Ao projetarmos o NetComm usamos uma metodologia orientada a objeto e passamos essa mesma divisão lógica para as classes de objetos.

4.2.1 Classes de objetos que compõe um projeto

Na figura 4.2 podemos observar quais as classes que compõem um projeto e qual a hierarquia envolvida entre elas (composição ou herança). Quando um objeto contém uma instância de outro, se diz que há uma composição e quando um objeto tem características de outro se diz que há uma herança [BORLAND94]. Por exemplo, a classe C_Visual herda características da classe C_Dado, já a classe C_Projeto contém uma instância da classe C_OQue.

Ao projetarmos o NetComm tomamos cuidado na reutilização de código. C++, que foi a linguagem utilizada para desenvolver o NetComm, é uma linguagem orientada a objeto que incorpora diversas características para facilitar a reusabilidade de código, entre elas, *templates* [SAKS93], métodos virtuais e herança. *Templates* e métodos virtuais foram largamente

utilizados na construção das classes de objetos do NetComm. Por exemplo, muitos dos objetos usam listas encadeadas para armazenar dados, foi então desenvolvida uma lista encadeada usando *templates* para ser usada por quem dela precisasse.

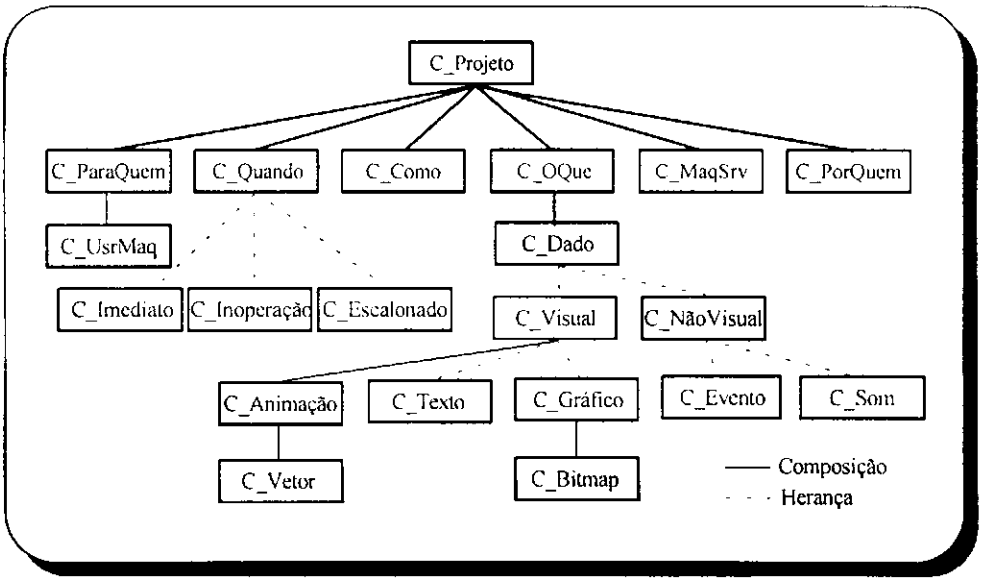

Figura 4.2 - Classes de objetos do NetComm

A classe C_Projeto contém dentro de sí uma instância de objeto das classes que representam as dimensões de um projeto: C_ParaQuem, C_Quando, C_Como, C_OQue, C_MaqSrv e C_PorQuem. Ou seja, um projeto é composto por instâncias desses objetos e mais alguns atributos específicos como por exemplo, nome do projeto.

A classe C_ParaQuem contém uma lista de objetos C_UsrMaq, que são pares ordenados contendo o nome do usuário e o nome da máquina a quem se destina o projeto. A classe C_Quando se sub-divide nas classes C_Imediato, C_Inoperação e C_Escalonado. Ou seja, dentro da classe C_Projeto existe um apontador para a classe C_Quando, porém é feito um *cast* dependendo do Quando selecionado pelo usuário no momento da construção de um projeto. A classe C_Como é muito simples, contendo apenas os atributos para definição da janela de apresentação do projeto. As classes C_PorQuem e C_MaqSrv também são muito simples e seus atributos são preenchidos automaticamente pelo NCMaker.

A classe C_OQue é um pouco mais complicada que as demais. Ela contém uma lista de objetos da classe C_Dado. Essa lista é basicamente uma representação de dados para a partitura

onde são mostrados os objetos no NCMaker. A classe C_Dado é uma classe que contém apenas métodos virtuais e deriva em duas outras classes: C_Visual e C_NãoVisual. A classe C_Visual contém um objeto da classe C_Animação que contém uma lista de objetos da classe C_Vetor. Ou seja, quando um objeto visual é animado, ele contém vetores de animação que são guardados em uma lista. A classe C_Visual deriva nas sub-classes C_Gráfico e C_Texto e a classe C_NãoVisual nas sub-classes C_Som e C_Evento. A classe C_NãoVisual não possui nenhum atributo especial e a classe C_Gráfico difere um pouco da C_Texto no sentido de conter um vetor de cinco objetos da classe C_Bitmap que serão usados para compor a animação de transformação. O limite de cinco *bitmaps* na composição da animação de transformação se deve unicamente à tentativa de reduzir o tempo de transmissão de dados pela rede. Poderíamos deixar que o usuário adicionasse um número ilimitado de *bitmaps* à animação de transformação e deixar a cargo de seu bom senso a escolha do limite.

Essa visão do projeto orientada a objeto trouxe enormes benefícios em termos de codificação e redução do número de linhas de código. Na próxima seção descrevemos como essa visão serviu no caso específico de acesso ao *Clipboard* [MSOFT92c][MSOFT92f] e acesso a arquivos.

4.2.2 Classes C_Arquivo e C_Clipboard

Existem três classes implementadas no NetComm que não foram mostradas na figura 4.2: C_Dispositivo, C_Arquivo e C_Clipboard. A classe C_Dispositivo é uma classe que só tem métodos virtuais [BORLAND93] e deriva em duas outras classes: C_Arquivo e C_Clipboard que efetivamente implementam os métodos *Open, Read* e *Write* definidos na classe pai. Todas as classes mostradas na figura 4.2 são derivadas da classe C_Dispositivo e herdam esses métodos. Desta forma, gravar um projeto em disco se tornou algo muito simples.

A classe C_Projeto implementa os métodos *Read* e *Write* simplesmente gravando um pequeno cabeçalho contendo por exemplo, um número mágico para a versão do arquivo e chamando os métodos *Read* e *Write* das classes C_ParaQuem, C_Quando, C_Como, C_OQue, C_PorQuem e C_MaqSrv. Cada uma dessas classes também se comporta da mesma forma, gravando o que precisa e passando a bola para as classes abaixo. Por exemplo, a classe C_ParaQuem grava seus dados próprios, como número de elementos da lista de pares (usuário,

máquina) e varre a lista passando o controle para cada um dos elementos da classe C_UsrMaq. A classe C_UsrMaq, que implementa seus próprios métodos Read e Write, finalmente grava seus atributos no arquivo. Em todo o processo, alguem de fora se encarregou de criar um objeto do tipo C_Arquivo e associá-lo ao arquivo de projeto em questão.

O *Clipboard* é uma região de memória compartilhada e em princípio não há diferença entre escrever em um arquivo e escrever em uma região de memória. Então porque não aproveitar o mesmo código escrito para gravar um projeto ou partes dele em arquivo e usá-lo para armazenar e ler dados no *Clipboard*? A classe C_Clipboard, faz exatamente isso. Ela implementa os métodos *Read* e *Write* de forma a fazer apenas uma cópia de regiões de memória. Desta maneira, todo o processo de gravação será o mesmo, com a única diferença de que no início devemos criar um objeto da classe C_Clipboard ao invés de C_Arquivo. Todo o código é aproveitado sem mudar uma só linha.

4.3 Distribuição de um projeto (NCServer e NCClient)

A fase de distribuição de um projeto está a cargo do NCServer e do NCClient. O NCServer é responsável por receber os projetos criados pelo NCMaker e distribuí-los aos clientes. O NCClient recebe os projetos enviados pelo NCServer e ativa o NCPlayer sempre que um projeto precisar ser apresentado.

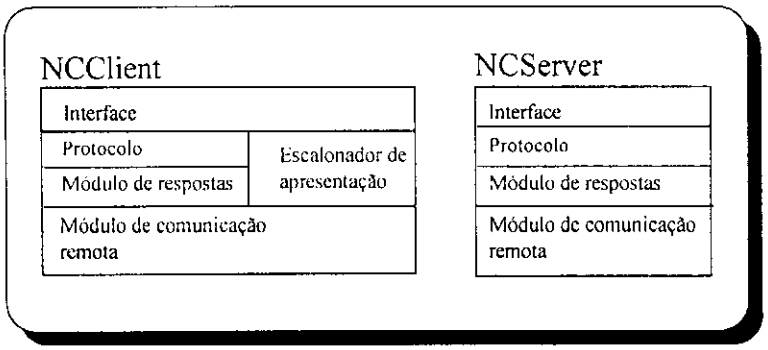

Figura 4.3 - Sub-módulos do NCServer e do NCClient

O NCClient está dividido em 5 sub-módulos básicos (figura 4.3). Em primeiro lugar está a interface com o usuário. Depois, a implementação do protocolo para troca de projetos e o

módulo de respostas a pedidos do NCServer. Na camada mais baixa está o sub-módulo de comunicação remota, responsável por detalhes de baixo nível de comunicação. A divisão funcional do NCServer é quase idêntica à do NCClient. O NCClient dispõe de um sub-módulo responsável pelo escalonamento dos projetos a serem apresentados (Escalonador de apresentação). Esse escalonador foi implementado como uma DLL (*Dynamic Link Library*) em virtude do Windows exigir que funções que façam uso de *hooks* (ganchos) sejam implementadas em DLLs [RITCHTER92]. O Escalonador usa um *hook*[11] para capturar os eventos de mouse e teclado para descobrir o tempo que o usuário passou inativo no caso de projetos por tempo de inatividade.

Ao enviar um projeto através da opção Send do menu do NCMaker, o usuário está na realidade passando o nome do projeto que está sendo editado para o NCServer através de uma mensagem. O NCServer se encarrega de tirar uma cópia do arquivo de projeto armazenando-o em um diretório sob seu controle. Imediatamente após ter sido efetuada a cópia do arquivo, o NCServer informa os clientes que um novo projeto está disponível na máquina. Todo o processo de troca de projetos e dados entre o cliente e o servidor é descrito na seção 4.3.2, antes porém é apresentada a nomenclatura utilizada na definição do protocolo.

4.3.1 Nomenclatura utilizada

A fim de entender melhor a próxima seção, onde apresentamos o protocolo para troca de projetos, adotamos a seguinte nomenclatura:

- *Handle* de projeto =
  - Identificador da máquina
  - Data
  - Hora
  - Contador numérico

- *Handle* de dado =

[11] Recurso oferecido pelo Windows o qual permite que eventos sejam capturados antes de serem repassados para a aplicação a qual se destinam.

- Contador numérico válido dentro do projeto

- Número do dado =
  - Objetos gráficos podem ter até 5 arquivos de dados associados. Portanto, este número identifica qual dos arquivos acessar. No caso dos outros objetos esse número será sempre 1.

- Dado =
  - *Handle* do dado + Dado propriamente dito

- Usuário (U) =
  - Par (usuário, máquina)

- Corpo =
  - S Dado - O corpo de um projeto representa o conjunto de todos os dados que compõem a dimensão "O quê". Ou seja, é um conjunto de arquivos gráficos do tipo bitmap, sons, textos ou eventos.

- Projeto =
  - Cabeçalho + Corpo

- Cabeçalho contém
  - *Handle* do projeto: identificador do projeto (único na rede onde o projeto foi criado)
  - "O quê"
  - "Como"
  - "Quando"
  - "Para quem"
  - "Por quem"
  - "Máquina servidora"

- "O Quê" =
  - Lista contendo os *handles* dos dados e referências aos arquivos onde os mesmos estão armazenados.

- Estados de um projeto:
  - Ativado: quando o projeto está pronto para rodar. O projeto é ativado sempre que um usuário se loga. Nesse caso todos os projetos destinados a ele são ativados. Se o usuário desejar, pode ativar um projeto previamente desativado por ele. Somente os projetos neste estado são passíveis de ser apresentados pelo NCPlayer.
  - Desativado: quando o projeto está "dormindo", ocorre quando o usuário ao qual ele se destina não está conectado. Este é o estado *default* de um projeto. Isto é, sempre que um novo projeto for criado, seu estado inicial será DESATIVADO. Um projeto é desativado sempre que o usuário encerra sua seção através do módulo de login ou por intervenção do usuário no NCClient.

4.3.2 Protocolo NetComm para Troca de Projetos

Para possibilitar a troca de projetos entre máquinas diferentes em um ambiente distribuído, foi criado um protocolo de comunicação que denominamos Protocolo de Troca de Projetos (PTP). O PTP define como os processos NCClient e NCServer se comunicam: que mensagens são trocadas, parâmetros que compõem as mensagens, comportamento dos processos em relação às mensagens, tipo das mensagens, erros. A tabela 4.1 a seguir mostra as mensagens que compõem o PTP. Os campos da tabela são os seguintes:

- Nome da mensagem
- Parâmetros da mensagem
- Breve descrição
- Sentido (Cliente-Servidor, Servidor-Cliente ou Servidor-Servidor)
- Tipo da mensagem (*Broadcast* ou *Unicast*)

Nome da Mensagem	Parâmetros da Mensagem	Breve Descrição	Sentido	Tipo de Mensagem
WhoHasHead	U	Quem tem cabeçalhos de projeto para o usuário U	C - S	Broadcast
IHaveHead	Handle de P	Eu tenho um cabeçalho de projeto	S - C	Unicast
GetHead	Handle de P	Dê-me o cabeçalho de projeto P	C - S	Unicast
PutHead	Header de P	Aí vai o cabeçalho do projeto	S - C	Unicast
NewProj	Handle de P	Projeto novo foi criado	S - C	Broadcast
GetNewHead	Handle de P, U	Pegue o cabeçalho do projeto novo se for para U	S - C	Unicast
GetNewData	Handle de P, Handle, Versão, Número de D	Dê-me a atualização do dado D do projeto P se a versão estiver desatualizada.	C - S	Unicast
GetData	Handle de P, Handle de D, Número de D	Dê-me o dado D do projeto P independentemente de versão	C - S	Unicast
VersionOK	Handle de D, Versão de D, Número de D	Versão de D está atualizada	S - C	Unicast
PutData	Handle,Versão, Número, Tamanho de D, D	Tome o dado D pedido	S - C	Unicast
ProjDied	Handle de P	Projeto P morreu	S - C	Broadcast
ProjUserDied	Handle de P, U	Projeto P morreu para usuário U	C - S	Unicast
ProjTransfered	Handle de P, Novo Servidor	O projeto P foi transferido para outro servidor	S - C	Broadcast
TransfProj	P	Transfira o projeto para sua máquina e assuma o papel de servidor	S - S	Unicast
GetTransfProj	Handle de P	Recupere um projeto transferido	S - S	Broadcast
GetTransfPrjOK	Handle de P	Responda a GetTransProj para saber se recuperação Ok	S - S	Unicast
WhoIsAlive	---	Quem está rodando?	S - S S - C	Broadcast
IamAlive	Id. Máquina	Resposta a WhoIsAlive	S - S C - S	Unicast

WhatProjs	Id. da máquina	Quais os projetos em M?	S - S	Unicast
MyProjsAre	Lista de Handles de Projeto	Meus projetos são os seguintes: resposta a WhatProjs.	S-S	Unicast
ServerInit	Id. Máquina	Estou entrando no ar	S - C	Broadcast
HaveHead	U	Você tem header de projeto para U?	C - S	Unicast
IHaveNoHead	- - -	Eu não tenho header de projeto	S - C	Unicast

Tabela 4.1 - Mensagens do Protocolo para Troca de Projetos (PTP)

Como se pode observar na tabela, existem 7 mensagens do tipo *Broadcast*. O uso de *broadcast* limita a escalabilidade [SANTOS93] do NetComm, tendo em vista que restringe seu uso a redes locais. Porém, a proposta é justamente essa: um *software* para grupos de trabalho conectados a uma rede local.

O *broadcast* é usado, de maneira disciplinada, somente para localizar ou para informar a disponibilidade/indisponibilidade de recursos na rede. Com essa perspectiva, demos características de uma arquitetura ponto-a-ponto à arquitetura cliente-servidor do NetComm. O uso de características ponto a ponto em uma arquitetura cliente-servidor traz muitos benefícios em contraposição à falta de escalabilidade. Como exemplo imediato está a não necessidade de configuração de características do servidor no cliente (endereço, por exemplo; o cliente se encarrega de encontrar o servidor ou servidores apropriados fazendo um *broadcast*). Outra vantagem é a possibilidade de mudança do servidor sem necessidade de reconfiguração dos clientes.

A forma como o protocolo foi projetado permite que existam múltiplos servidores e múltiplos clientes. O protocolo foi pensado de forma a facilitar a gerência dos projetos na rede. Há, por exemplo, a possibilidade de se transferir projetos para que outra máquina da rede faça o papel de servidora. Isso é útil quando queremos construir projetos em várias máquinas da rede mas nem todas ficam ligadas as 24 horas do dia. Vale salientar que ao se transferir um projeto para outra máquina, é feita uma cópia dos dados e que o atributo de dinamismo dos objetos perde o sentido, pois não é feita atualização de dados entre servidores. Levando em conta esse detalhe devemos afirmar que os projetos não são totalmente independentes de máquina. Sob esse aspecto

de gerência dos projetos existem também mensagens para verificar quais os servidores menos sobrecarregados de projetos ou quais servidores/clientes estão ativos.

4.3.3 Quadros de troca de mensagem

Para entender melhor o funcionamento de cada mensagem do protocolo e como o NCServer e o NCClient se comportam ao receber determinadas mensagens elaboramos quadros de troca de mensagens que apresentaremos nesta seção. Situações especiais como a queda de um servidor serão tratadas mais detalhadamente em seções específicas.

4.3.3.1 Quadro 1 - NCClient é inicializado.

Quando o NCClient é inicializado, ele envia uma mensagem *broadcast* WhoHasHead para os servidores (NCServer) perguntando quais deles têm projetos para o usuário correntemente logado e fica aguardando por respostas durante um determinado período. Os servidores respondem ao NCClient com mensagens IHaveHead e passam como parâmetro o handle do projeto que eles têm disponível. O NCClient, após receber as respostas dos servidores abre uma conexão com cada um deles pedindo o cabeçalho dos projetos com a mensagem GetHead.

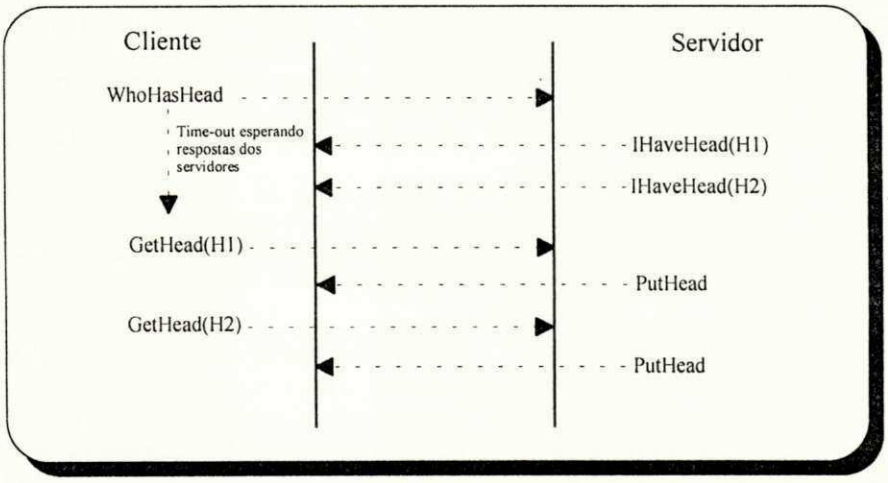

Figura 4.4 - Troca de mensagens na inicialização do NCClient

Ao receber uma mensagem GetHead(U) o NCServer verifica localmente se tem projetos destinados ao par (usuário, máquina) U e em caso positivo envia uma mensagem PutHead(*Header* de P) para quem enviou a mensagem (NCClient). Caso o NCServer não tenha nenhum

projeto com a parte Para Quem contendo o par (usuário, máquina) U, ele envia uma mensagem negativa de forma a evitar que o cliente fique aguardando uma resposta.

4.3.3.2 Quadro 2 - NCServer é inicializado

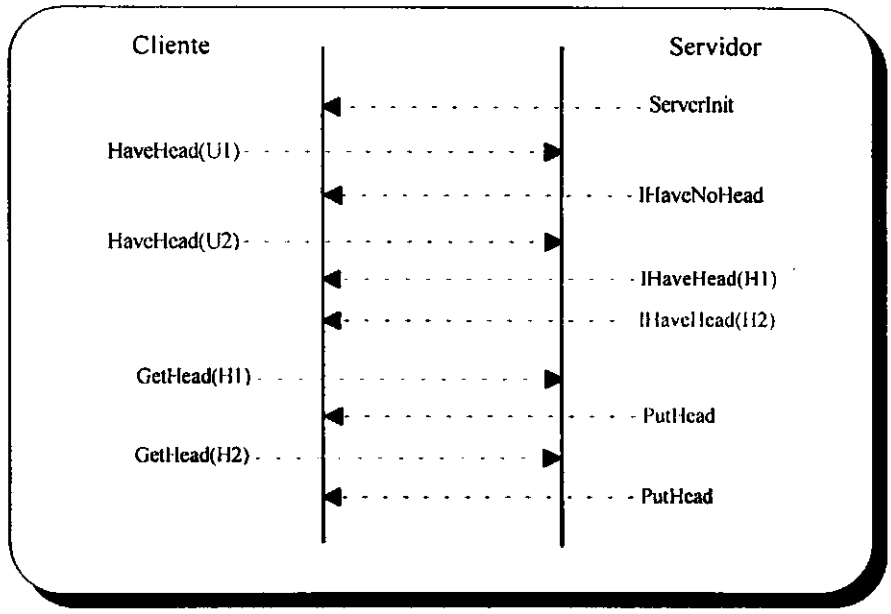

Figura 4.5 - Troca de mensagens na inicialização do NCServer

O processo de inicialização do NCServer é mostrado na figura 4.5. O NCServer ao entrar no ar envia uma mensagem *broadcast* ServerInit para os clientes. Os clientes então perguntam se o NCServer tem algum projeto para o usuário correntemente logado. O NCServer responde às perguntas dos clientes ou com uma mensagem IHaveNoHead ou com a mensagem IHaveHead. Os clientes então solicitam os cabeçalhos dos projetos cujos *handles* foram enviados pelo NCServer no caso da resposta ter sido IHaveHead.

4.3.3.3 Quadro 3 - Um novo projeto é criado

Quando um novo projeto é criado, o NCServer envia uma mensagem *broadcast* NewProj para os clientes. Os clientes ao receberem a mensagem NewProj, pedem o cabeçalho do projeto novo ao NCServer perguntando se ele se destina ao usuário correntemente logado; isso é feito com a mensagem GetNewHead. Caso o projeto se destine ao usuário que foi passado como parâmetro na mensagem GetNewHead, o NCServer o envia com uma mensagem PutHead. Caso contrário uma mensagem de erro é enviada ao NCClient.

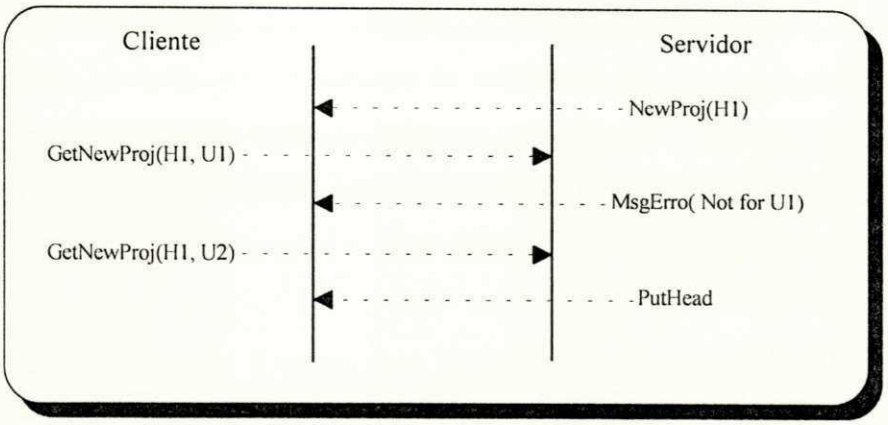

Figura 4.6 - Troca de mensagens quando um novo projeto é criado

4.3.3.4 Quadro 4 - Um projeto é escalonado para ser apresentado

Sempre que o NCClient é inicializado ele busca todos os cabeçalhos de projeto que são destinados ao usuário correntemente logado. Um projeto porém, é composto pelo cabeçalho mais os dados. Esses dados são arquivos gráficos do tipo bitmap (.bmp), arquivos de som (.wav), arquivos de texto ou de eventos (.rec) e são buscados no momento que o projeto vai ser apresentado. Desta forma só geramos tráfego na rede no momento em que o projeto precisa ser apresentado, ou seja, quando precisamos dos dados. A figura 4.7 mostra como o NCClient e o NCServer se comportam no momento de apresentar um projeto.

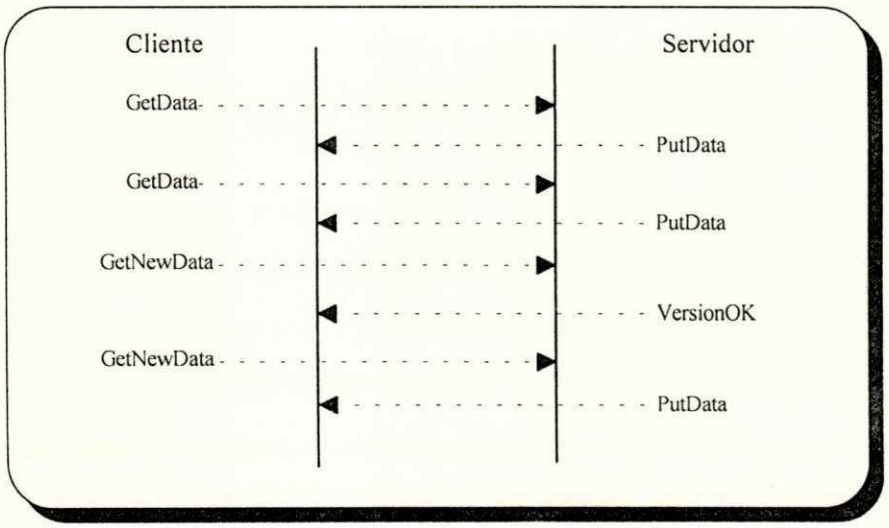

Figura 4.7 - Troca de mensagens no momento que um projeto vai ser apresentado

O NCClient primeiro verifica se o corpo (dados) do projeto está presente. Nesse caso os dados não dinâmicos não precisam ser buscados no servidor. Caso o corpo não esteja presente o NCClient solicita os dados não dinâmicos ao NCServer com a mensagem GetData a qual é respondida com a mensagem PutData. Independentemente do corpo do projeto estar ou não presente é preciso pedir atualizações dos dados dinâmicos do projeto através das mensagens GetNewData. O NCServer responde a essa solicitação com uma mensagem VersionOK ou PutData.

A mensagem GetData é usada pelo NCClient quando ele deseja buscar um dado do projeto no servidor. A mensagem GetNewData é semelhante a GetData, porém a mesma é enviada só para pedir atualizações de dados dinâmicos. Sempre que um projeto é escalonado, o NCClient pede a atualização dos dados dinâmicos do mesmo. O NCServer se encarrega de verificar se o dado do NCClient precisa ser atualizado comparando a versão do mesmo. Caso a versão do dado que o NCServer dispõe seja igual à que o NCClient pediu ele manda uma mensagem VersionOK como resposta. Em caso contrário, ou seja, as versões do dado do NCClient e do NCServer diferem é porque o mesmo precisa ser atualizado. Neste caso, o NCClient envia uma mensagem PutData para o NCClient. O NCClient passa a contar, a partir daquele momento com uma versão atualizada do dado.

4.3.3.5 Quadro 5 - Um projeto é transferido para outro servidor.

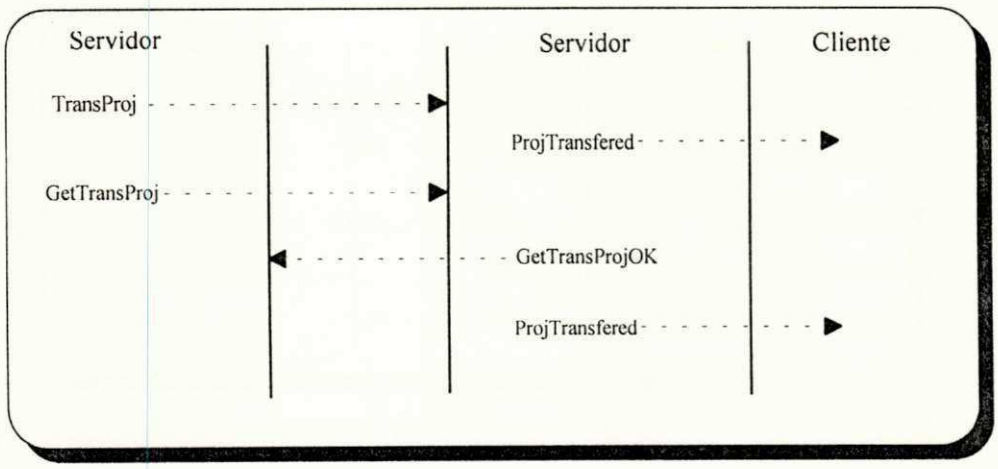

Figura 4.8 - Mensagens trocadas quando um projeto é transferido e depois recuperado

A transferência de projetos é util para liberar a carga de um determinado servidor. Por exemplo, suponha que um servidor está sobrecarregado de projetos e atendendo muitos pedidos de diferentes clientes. O usuário da máquina servidora ou o administrador da rede podem decidir distribuir melhor os projetos, tornando outra máquina da rede servidora também. A figura 4.8 mostra a troca de mensagens entre servidores e clientes envolvidas na transferência e posterior recuperação de um projeto.

Ao transferir um projeto de uma máquina para outra, o NCServer envia uma mensagem TransProj para outro NCServer. O segundo se encarrega de avisar os clientes da mudança de servidor enviando uma mensagem *broadcast* ProjTransfered. Caso o primeiro NCServer deseje recuperar a posse do projeto ele envia uma mensagem broadcast GetTransProj para os servidores. O NCServer que então estiver tomando conta do projeto envia uma mensagem GetTransProjOK e avisa os clientes da nova mudança de servidor. Sempre que um projeto é transferido de um servidor para outro, os dados são transferidos junto e os dados dinâmicos que faziam parte do projeto passam a ser estáticos. Isso ocorre porque as atualizações que eram feitas nos dados no servidor de origem não poderão mais ser feitas no novo servidor. Os arquivos são gerados com novos nomes em uma outra máquina. Ficaria portanto, muito difícil garantir o dinamismo dos dados nessa situação.

4.3.3.6 Quadro 6 - Morte de um projeto

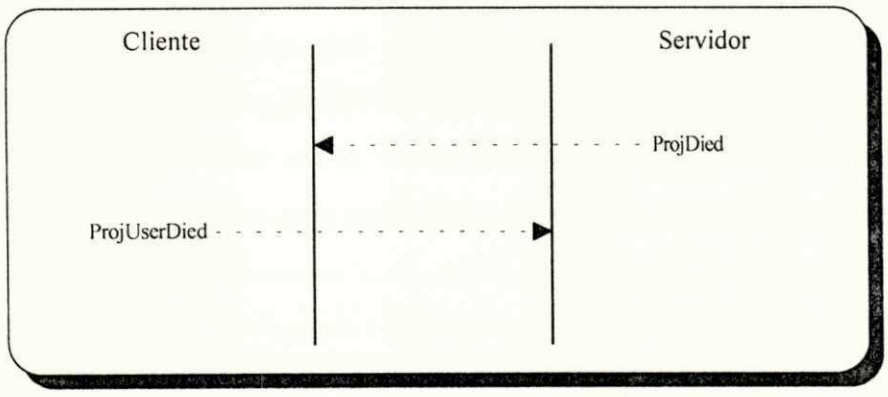

Figura 4.9 - Mensagens enviadas durante morte de um projeto

Existem duas situações de morte de um projeto: a primeira ocorre normalmente quando o tempo de vida do projeto expira e a segunda ocorre por intervenção do usuário. A intervenção do usuário em relação à morte de um projeto pode se dar tanto no NCServer quanto no NCClient.

Ao matar um projeto no NCServer estamos removendo o projeto do controle do mesmo e o projeto em questão passa a não mais existir para todos os usuários da rede. Ao matar um projeto no NCClient estaremos eliminando todos os arquivos relacionados ao projeto apenas na máquina cliente e apenas o usuário que matou o projeto não mais o receberá. Os demais usuários da rede continuarão a receber dados do mesmo normalmente. Quando o projeto é morto a partir do NCServer uma mensagem *broadcast* ProjDied é enviada para os clientes e eles se encarregam de apagar todos os dados referentes ao projeto. Quando o projeto é morto a partir do NCClient uma mensagem ProjUserDied é enviada para o NCServer.

4.3.4 Processo de inicialização do NCClient e NCServer

O calcanhar de Aquiles de qualquer software distribuido aparece no momento em que ocorrem situações de erro. A maioria das vezes não é possivel nem sequer determinar a situação de erro. Por exemplo, quando um servidor sai do ar, os clientes precisam saber que isso aconteceu. Como é que os clientes podem tomar conhecimento da queda de um servidor? Seria ótimo se antes de um processo servidor morrer ele pudesse avisar os clientes desse fato. Ocorre que quando falta energia não há como o servidor saber sequer que ele ía morrer. O esquema de tratamento de erros mais usados em ambientes distribuidos é baseado em fatias de tempo [SANTOS93]. Ou seja, no caso específico da queda de um servidor, se depois de um certo tempo o servidor não responder é porque provavelmente ele está com problemas. O problema é justamente o fato de que ele pode não estar com problemas. Simplesmente a máquina na qual o servidor estava executando podia estar momentaneamente sobrecarregada e o pedido do cliente não foi respondido até aquele momento mas será em um tempo X, só que esse tempo X é maior que o que cliente esperava. Um exemplo específico pode ocorre no momento da inicialização do NCClient.

Ao entrar no ar, o NCClient envia uma mensagem broadcast no sentido de localizar os servidores. Os servidores que estiverem ativos respondem e o NCClient armazena as respostas em uma lista durante um certo tempo. Acontece que se um servidor estivesse momentaneamente

sobrecarregado e tardasse a responder, o NCClient acharia que não há mais servidores ativos e simplesmente deixaria de receber uma resposta que seria enviada mais tarde. Esta seção trata de duas situações específicas de erro e a forma como elas foram resolvidas. As situações são a queda e o processo de reinicialização do NCClient e do NCServer. Informações mais detalhadas sobre tratamento de erros em ambientes distribuídos podem ser encontradas em [GEDEON93].

4.3.4.1 Queda e re-inicialização do NCClient

O NCClient é um processo que guarda estado, portanto sempre que ele for re-inicializado o estado anterior deverá ser restaurado. A forma que usamos para recuperar estado é simples: sempre que o NCClient é re-inicializado pedimos aos servidores que mandem os dados necessários para restaurar o estado anterior. Tudo o que for recebido é admitido como parte do estado anterior e é feita uma distinção entre os dados recebidos e os previamente existentes. Se algum dado que existia previamente não for recebido pelo NCClient, assumimos que o mesmo não tem mais utilidade e simplesmente o descartamos.

Sempre que o NCClient cai e é re-inicializado, ele deve tomar algumas atitudes. Em primeiro lugar, o NCClient verifica qual o usuário que está logado e envia uma mensagem WhoHasHead para os servidores, aguardando um determinado tempo pelas respostas dos mesmos. Cada uma dessas respostas é armazenada em uma lista onde são guardados os *handles* dos projetos recebidos. Para cada *handle* de projeto um diretório é criado (caso não existisse previamente). Esse diretório servirá para o armazenamento do cabeçalho do projeto e dos dados a ele associados. Após transcorrido o tempo de espera de respostas o NCClient varre a lista com os *handles* de projeto e pede, caso já não estejam presentes, os respectivos cabeçalhos um a um. O NCClient descarta os projetos que ele tinha disponível mas que não foram recebidos e ativa aqueles que foram recebidos.

4.3.4.2 Queda e re-inicialização do NCServer

O NCServer é um processo que não guarda estado, consequentemente seu processo de re-inicialização é bem mais simples. É trabalho do NCClient verificar se um determinado NCServer está inativo. Isso é feito ao tentar uma conexão e não ser atendido. Nesse caso o NCClient assume que o NCServer está inativo, desativa os projetos relacionados a ele e não mais

pede dados ou tenta estabelecer conexão com o mesmo até receber um comunicado de que ele está ativo novamente. O processo de re-inicialização do NCServer consiste exatamente em fazer esse comunicado e só. Quando o NCServer é re-inicializado, ele envia uma mensagem *broadcast* ServerInit para os clientes e fica aguardando seus pedidos. É trabalho dos clientes atualizarem seu estado indo buscar os cabeçalhos de projeto no NCServer que acabou de entrar no ar.

4.3.5 Mensagens para gerência de projetos

No protocolo PTP existem algumas mensagens (além daquelas para transferência de projetos) para permitir que o administrador da rede saiba avaliar a quantidade de projetos e determinar uma melhor distribuição entre os servidores. Essas mensagens de gerência também podem ser úteis aos usuários que desejam saber quem está conectado na rede e quais os clientes/servidores ativos. São, ao todo, quatro mensagens: WhoIsAlive, IamAlive, WhatProjs e MyProjsAre. A mensagem WhoIsAlive é enviada sempre que o usuário quiser saber quais são os clientes ou os servidores ativos na rede e quem está logado na máquina. A mensagem WhatProjs é usada para perguntar a um servidor quais os projetos que estão sob sua responsabilidade. As mensagens IamAlive e MyProjsAre são as respostas a WhoIsAlive e WhatProjs respectivamente.

4.4 Apresentação de um projeto (NCPlayer)

A fase de apresentação de um projeto está a cargo do NCPlayer com a ajuda do NCClient. Durante seu processo de inicialização, o NCClient constroi duas listas dos projetos sob seu controle passíveis de serem apresentados: uma para projetos escalonados e outra para projetos por tempo de inoperação. O escalonador de apresentação (figura 4.3) varre essas listas de tempos em tempos verificando se um projeto precisa ou não ser apresentado. Em caso positivo, o NCClient pede todos os dados necessários à apresentação do projeto para o NCServer e ativa o NCPlayer.

O NCPlayer é, conceitualmente, o módulo mais simples do NetComm (excluindo o NCLogin). Ele não tem interface com o usuário. A interface do NCPlayer é na realidade a janela definida na dimensão Como do projeto a ser apresentado e toda a informação apresentada nessa

janela é a que foi definida na dimensão "O Quê". Ou seja, caso o usuário defina uma janela escondida, o NCPlayer executará sem que o destinatário do projeto tome conhecimento. A inexistência de interface é um dos fatores que contribuem para a menor complexidade do NCPlayer em relação aos demais módulos. Outro fator importante é que todos os objetos manipulados pelo NCPlayer são os mesmos usados pelo NCMaker, portanto o código é o mesmo tanto para um como para o outro. Por exemplo, a leitura de um projeto foi implementada inicialmente no NCMaker e posteriormente aproveitada pelo NCPlayer. Esse aproveitamento de objetos e código fez com que o tempo gasto na implementação do NCPlayer fosse muito reduzido em relação aos demais módulos.

A maior dificuldade na implementação do NCPlayer diz respeito à animação de objetos gráficos. O Windows possui uma API para tratamento de gráficos denominada GDI (Graphic Device Interface) [PETZOLD90][RITCHTER92][MSOFT92c]. Apesar de muito poderosa para a construção de gráficos em duas dimensões, a GDI não possui funções para o tratamento de gráficos animados nem para a construção de objetos em três dimensões. Essas limitações do Windows foram resolvidas na nova versão 4.0 (Chicago) e no Windows NT versão 3.5. Como não foram essas as plataformas de desenvolvimento do NetComm foi preciso improvisar.

O Windows não possui funções adequadas para a animação de gráficos e e as poucas que possui não têm a velocidade de processamento adequada. As funções da GDI usadas na manipulação de *bitmaps* são muito lentas para a construção de animações. Na implementação do NCPlayer foram tomados cuidados com relação à velocidade de apresentação de gráficos animados. Ao incluir um objeto desse tipo em um projeto o usuário pode optar por ganhar em velocidade e perder em graciosidade fazendo ou não uso do atributo *transparent*. Esse atributo deve ser desmarcado quando os *bitmaps* usados são de grande tamanho.

Para apresentar qualquer informação na tela, o Windows faz uso de contextos de dispositivos. A idéia é que esses contextos sejam independentes do dispositivo de saída utilizado. Ou seja, da mesma forma que se coloca um bitmap em um monitor de vídeo de 800 por 600 pixels se coloca em um de 1024 x 768 ou em uma impressora.

O NCPlayer usa dois contextos de dispositivo para fazer a animação de objetos na tela. Enquanto desenhamos em um deles estamos apresentando a informação do outro e é feita a troca entre os dispositivos de contexto para dar o efeito de animação. Essa troca entre dispositivos de

contexto garante que as animações serão suaves. Caso fosse usado apenas um dispositivo de contexto os objetos aparentariam piscar na tela pois os mesmos precisariam ser apagados e mostrados novamente às vistas do usuário. Usando dois contextos de dispositivo o apagamento e re-apresentação é feito em um contexto de dispositivo enquanto outro está sendo apresentado na tela. Como a troca de dispositivos é feita uma vez que toda a imagem a ser apresentada está construida, o usuário não notara o efeito de varredura entre os diferentes objetos nem o apagamento dos mesmos. Isso fará com que o mesmo veja a animação de forma contínua.

Como dissemos anteriormente, mostrar um bitmap em um contexto de dispositivo é um processo lento. O atributo *transparent*, quando usado em objetos gráficos faz com que a operação seja ainda mais lenta. Para dar a impressão de transparência a um *bitmap* é construida uma máscara de bits usando operações XOR e AND [EDDON93]. Essa máscara é um segundo *bitmap* que deve ser apresentado o que faz com que no melhor dos casos, a animação do objeto seja duas vezes mais lenta se o mesmo possuir o atributo *transparent*. A solução neste caso, é evitar o uso do atributo sempre que os bitmaps forem de um tamanho considerável.

4.5 Segurança no NetComm (permissões de usuários)

Segurança é um tópico que não pode ser esquecido em se tratando de software distribuido. O NetComm é um software que devido à sua versatilidade permite uma grande liberdade ao usuário e portanto, mecanismos de policiamento devem ser levados em consideração. Imagine se algum usuário envia para outro um projeto com um objeto do tipo evento que ativa o gerenciador de arquivos do Windows e apaga todos os arquivos a partir da raiz. Esse tipo de coisa é que queremos evitar. Para isso foi implementado um esquema de permissões para os usuários do NetComm.

A idéia inicial era pegar uma carona no esquema de segurança do sistema operacional de rede se fosse o caso de usarmos *Windows for Workgroups* ou Netware na implementação do NetComm. Neste caso, poderíamos usar os serviços existentes para autenticação de usuários e cadastro de grupos. Como resolvemos implementar o NetComm usando Windows Sockets em redes TCP/IP [HALL93], foi preciso elaborar um esquema próprio de permissões. A implementação usando Windows Sockets está descrita no capítulo 5.

O cadastro de usuários é feito a nível local, pois não temos a disposição um serviço de nomes distribuido. Da mesma forma, localmente, o usuário da máquina pode determinar níveis de acesso para os usuários que são de seu conhecimento. Ao cadastrar um usuário através do NCLogin, ele recebe um nível *default* de acesso aos recursos do NetComm. É possível, no entanto, criar um arquivo extra de permissões para determinar o nível de acesso que ele terá aos recursos da máquina local. São ao todo três níveis de permissões: nenhuma restrição (*default*), média restrição e máxima restrição. Sobrepondo-se a esses três níveis, existe um usuário especial de nome NCADMIN que tem poderes especiais relacionados à transferência de projetos.

O nível de permissão determina que tipo de projetos o usuário permite que outras pessoas enviem para sua máquina local. Caso o usuário que enviou o projeto não esteja cadastrado na máquina, será assumido o nível máximo de restrição para ele. Na tabela 4.2 abaixo podemos observar o que cada usuário pode fazer segundo o nível de restrição em que ele foi cadastrado. O primeiro campo da tabela indica o tipo de atividade e os demais são relacionado ao nível de restrição do usuário.

ATIVIDADE	NCADMIN	1	2	3
Transferir projetos de uma máquina para outra	S	N	N	N
Projetos que contenham objetos do tipo evento	S	S	N	N
Projetos com Quando por escalonamento	S	S	S	N
Projetos com Quando por tempo de inoperação	S	S	S	N
Projetos com Quando imediato	S	S	S	S

Tabela 4.2 - Permissões dos usuários segundo seu nível de restrição

Vale salientar que não existem restrições sobre o que um usuário pode fazer desde que a dimensão Para quem contenha ele próprio (Para quem = Por quem). Com relação à segurança, o NetComm possui também mecanismos de auditoria que podem ser ativados a partir do NCClient ou do NCServer. Segurança não é o tema principal desta dissertação, por isso resolvemos implementar algo mais simples porém factivel.

Capítulo 5 - Detalhes de implementação

5.1 Modularização

O projeto do NetComm foi dividido de acordo com seus módulos funcionais NCMaker, NCPlayer, NCServer, NCClient e NCLogin em três níveis ou camadas. Para entender melhor, definimos as camadas hierarquicamente como níveis de uma árvore de diretórios. Na primeira camada, ou primeiro nodo, incluimos as bibliotecas gerais do NetComm. Nela estão funções utilizadas por todos os módulos do NetComm. A segunda camada na hierarquia refina dois nodos: *Network* e Editor de Projetos. No nodo *Network*, estão os módulos NCLogin, NCServer e NCClient. Já no nodo Editor de projetos estão os módulos NCMaker e NCPlayer - vale salientar que cada um dos nodos possui suas próprias bibliotecas comuns. Abaixo, na camada três, foi preciso subdividir os módulos mais complexos do NetComm como o NCMaker, o NCServer e o NCClient. Na figura 5.1 abaixo podemos ver os módulos do NetComm hierarquicamente.

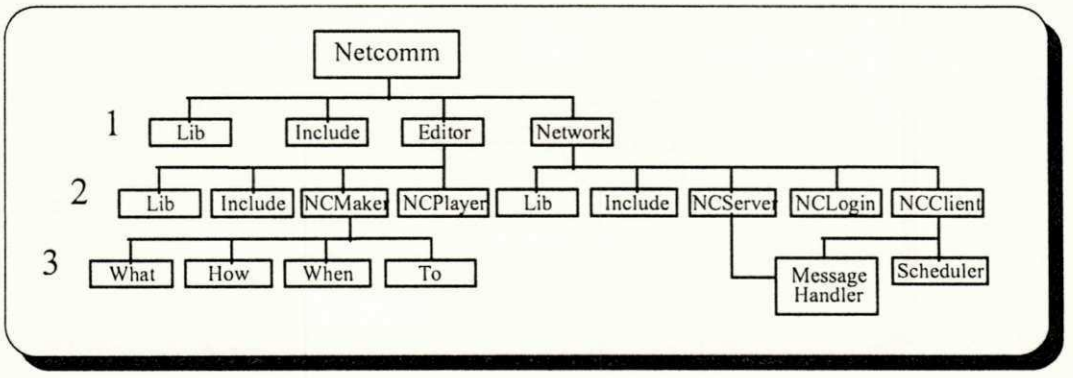

Figura 5.1 - Modularização hierarquica do NetComm

O módulo NCMaker é o maior dos módulos, tendo em vista a complexidade de sua interface e a grande interação com o usuário. Os módulos responsáveis pela comunicação inter-processo remota são menores em volume de código porém não em complexidade. Os detalhes de como foi implementada a comunicação inter-processo no NetComm são assunto da próxima seção.

5.2 Comunicação inter-processo

Os processos do NetComm se comunicam entre si de duas formas distintas: através de troca de mensagens ou através de passagem de parâmetros por linha de comando. Quando a comunicação é remota, como por exemplo no caso do NCClient e do NCServer, a troca de mensagens é feita usando a API (*Application Programming Interface*) de rede que será apresentada na seção 5.3. No caso da comunicação inter-processo ser local, como por exemplo entre o NCMaker e o NCLogin, a troca de mensagens é feita usando o suporte do próprio Windows para isso. A comunicação entre o NCMaker ou NCClient e o NCPlayer é feita através de parâmetros em linha de comando. Existe uma outra forma de comunicação inter-processo usada pelo NetComm que é através de arquivos. Isso ocorre por exemplo, entre o NCPlayer e o NCMaker. Ambos usam o mesmo arquivo de definição de projetos.

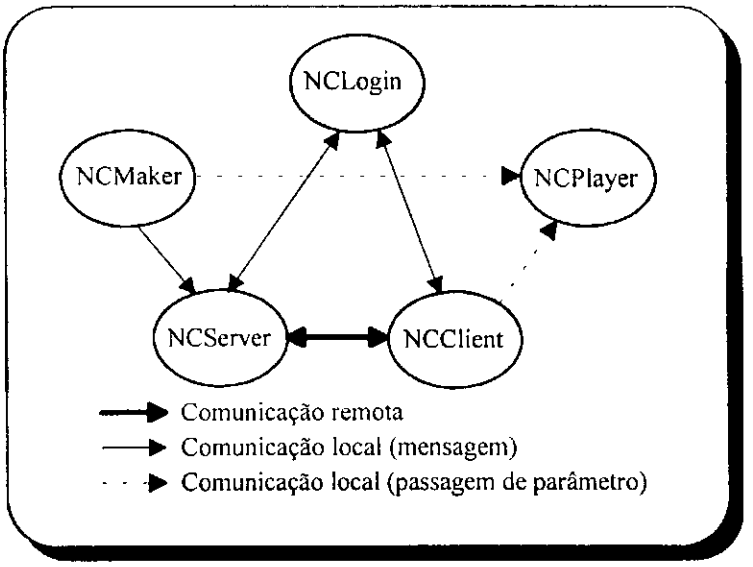

Figura 5.2 - Comunicação inter-processo no NetComm

O Windows dispõe de um eficiente sistema de troca de mensagens entre processos distintos. Como podemos observar na figura 5.3, no Windows existem duas filas de mensagens distintas: a fila do Windows e a fila da aplicação. A fila do Windows recebe eventos de *hardware*, tais como o movimento do mouse ou o pressionar de uma tecla. A fila da aplicação recebe mensagens de outras aplicações ou geradas pelo Windows.

Um exemplo de mensagem que é recebida através da fila do Windows é a WM_KEYDOWN[12], já uma mensagem gerada pelo Windows ou outra aplicação poderia ser WM_CHAR. A mensagem WM_KEYDOWN ocorre quando uma tecla é pressionada, já a mensagem WM_CHAR[13] é gerada pelo Windows quando ocorrem duas mensagens seguidas: WM_KEYDOWN e WM_KEYUP[14]. Ou seja a mensagem WM_CHAR, que é recebida pela fila da aplicação é gerada por duas outras que são recebidas pela fila do Windows. Vale ressaltar que qualquer aplicação Windows tem acesso a essas duas filas e que do ponto de vista do programador ela é apenas uma.

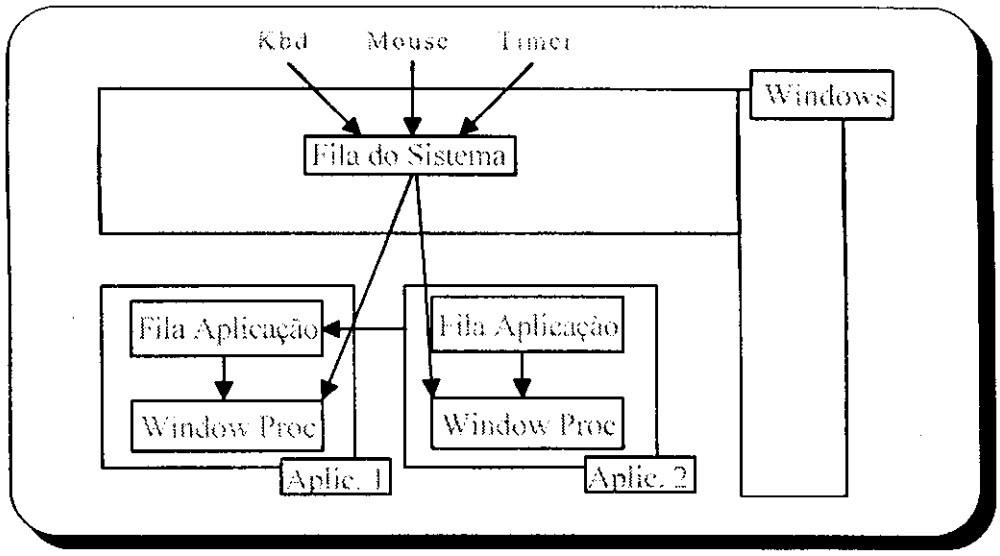

Figura 5.3 - Filas de mensagens do Windows

Toda mensagem no ambiente Windows dispõe de dois parâmetros denominados de wParam e lParam (um tipo Word e um tipo Long respectivamente). A semântica associada a esses dois parâmetros varia de mensagem para mensagem de forma que o lParam, por exemplo, pode tanto significar o apontador para uma estrutura bem como as coordenadas (x,y) do mouse dentro da janela. Para enviar uma mensagem para a fila da aplicação basta chamar a função PostMessage da API do Windows. Essa função recebe quatro parâmetros: um identificador para a janela à qual se deseja enviar a mensagem, um valor inteiro sem sinal para identificar a mensagem e os já conhecidos wParam e lParam como parâmetros da mensagem.

---

[12] Mensagem do Windows usada para indicar que uma tecla foi pressionada.
[13] Mensagem do Windows usada para indicar que uma tecla foi pressionada e liberada.
[14] Mensagem do Windows usada para indicar que uma tecla foi liberada

Como podemos observar, a forma de identificar o destinatário da mensagem não é a aplicação e sim a janela. Isso nos traz problemas pelo fato de que normalmente, uma aplicação tem mais de uma janela, ou pior, pode haver mais de uma instância da mesma aplicação executando ao mesmo tempo. Para resolver o segundo problema limitamos a execução dos processos em que a comunicação local é feita usando troca de mensagens a uma única instância. Isso não é uma limitação, mas pelo contrário, é algo desejado. Não queremos duas instâncias do NCServer executando, por exemplo. O Primeiro problema é resolvido como consequência do segundo. A partir do momento em que temos uma única instância do processo podemos identificar, já que o projetamos, a janela principal do mesmo para a qual queremos enviar as mensagens.

Agora que sabemos como os processos do NetComm se comunicam localmente usando mensagens, vamos descrever como essa mesma comunicação é feita para processos remotos (NCClient e NCServer). A API usada para comunicação remota é o assunto da próxima seção.

5.3 API de Rede

Quando falamos em computação distribuida, estamos falando ao mesmo tempo em heterogeneidade. Ou seja, são diferentes computadores, de diversos fabricantes, usando arquiteturas de rede distintas e interfaces de programação distintas. Seria ótimo se a palavra heterogeneidade fosse banida da área de informática, mas para nossa infelicidade ela existe.

A forma de se programar aplicações distribuidas hoje, que é o que nos interessa, não está padronizada. Alguém poderia se arriscar a dizer que RPC (*Remote Procedure Call*) [BLOOMER91][SANTOS93] é um padrão, mas surge a pergunta: RPC de quem, da ONC (*Open Network Computing*), da Microsoft? Objetos distribuidos [OMG91] poderiam ser uma opção, mas novamente vem a pergunta: CORBA [OMG91] ou OLE 2.0 [BRSCHMIDT93] da Microsoft?

Ao nos dirigirmos ao ambiente Windows, que novamente é o que nos interessa, existe um padrão *de facto* em se tratando de arquitetura de redes que é Netware da Novell, responsável por mais de 70% do mercado de redes para PCs. O problema é que ainda existem os outros 30% para atrapalhar nossa vida. São diferentes pilhas de protocolos para cada uma delas como por

exemplo: IPX/SPX, NetBeui, TCP/IP [DAVIS93]. Logo, se queremos uma aplicação que funcione em todos esses ambientes, precisaremos de uma API de Rede única para todos eles. O problema é que essa API única não está disponível. Não é tema deste trabalho padronizar uma interface única para a construção de aplicações distribuidas. Uma tentativa de definir uma API única de acesso a serviços de rede para o ambiente Windows pode ser vista em [DAVIS93]. O NetComm usa uma API de acesso a serviços de rede denominada Windows Socket [HALL93] que será explicada a seguir.

5.3.1 Windows Sockets

Windows Sockets é a especificação de uma API de baixo nível para acesso a protocolos de comunicação em ambiente Windows desenvolvida pela Microsoft como parte de algo denominado WOSA (*Windows Open System Architecture*) . WOSA é formado por um conjunto enorme de especificações que variam desde APIs para acesso a serviços de Mail (MAPI) à padronização de acesso a Bancos de dados (ODBC).

Windows socket é usado para acessar pilhas de protocolo TCP/IP mas nada impede que essa mesma API seja usada para acessar outras pilhas de protocolos como IPX/SPX, desde que haja interesse da Novell ou de terceiros em disponibilizá-la.

Windows Socket é uma API baseada na especificação de Sockets de Berkeley [STEVENS90] com algumas extensões necessárias ao ambiente Windows. Windows é um sistema cuja programação é orientada a eventos e tudo nele é informado através de mensagens. As extensões se fazem necessárias para permitir que mensagens sejam associadas a eventos de rede e que elas sejam repassadas através da fila de mensagens da aplicação (figura 5.3).

O Windows é um sistema operacional não pré-emptivo, isso quer dizer que é responsabilidade da aplicação que executa sob esse sistema retornar o controle para que outra aplicação possa executar [RITCHTER92]. As funções originais de Sockets de Berkeley bloqueiam a aplicação enquanto uma operação está sendo efetuada. Por exemplo, ao fazer a leitura de dados de um determinado socket a aplicação fica bloqueada até que haja dados disponíveis no socket. Fazer esse tipo de operação no Windows seria desastroso, já que bloquear uma aplicação no Windows significa bloquear todas as outras aplicações. Por esse motivo não existem funções na API de Windows Socket que bloqueiem a aplicação, mas para garantir

compatibilidade com Berkeley o bloqueio é simulado na implementação de Windows Socket. A implementação de Windows Socket se encarrega de, ao receber um pedido de operação com bloqueio, entrar em um laço de espera no qual executa a operação pedida de forma não bloqueada, só retornando o controle para a aplicação chamadora no momento que a mesma se concluir. Nesse laço de espera a implementação de Windows Socket fica responsável por entregar o controle a outras aplicações para que elas possam executar. Ou seja, todas as funções dentro da API de Windows Socket são "não bloqueadas", mas pode ser simulado um bloqueio quando desejado.

As extensões de Windows Socket em relação à Sockets de Berkeley são relacionadas ao assincronismo inerente do Windows. O controle de uma aplicação Windows é centralizado em funções associadas a janelas (*Window Procedures*) e que recebem e tratam todos os eventos destinados à mesma. Na figura 5.4 podemos ver como isso se processa. Uma determinada mensagem chega em qualquer uma das filas e ela é despachada para a janela que ativa sua função associada ou passa a mesma para que seja processada por uma função default chamada *Default Window Procedure*. Pois bem, Windows Socket permite que se associe um determinado Socket a uma janela, informando que tipo de evento de rede desejamos tratar e a que identificador de mensagem o mesmo está associado. Por exemplo, para um determinado socket queremos ser informados através da mensagem WM_NEWPACKET (mensagem criada pelo usuário), sempre que houver algum dado pronto para ser lido. Com esse tipo de filosofia de programação (orientada a eventos) não é necessário uma função de leitura com bloqueio, pois só tentaremos ler algum dado do socket sempre que houver disponibilidade do mesmo, ou melhor dizendo, sempre que recebermos uma mensagem WM_NEWPACKET na fila.

O paradigma de troca de mensagens para o desenvolvimento de aplicações distribuidas não é o que poderíamos chamar de "estado-da-arte" em se tratando de abstração de detalhes de comunicação. O uso de RPCs ou de objetos distribuidos é uma maneira muito melhor se comparada com troca de mensagens usando sockets. O principal fato que nos levou a adotar esse tipo de abstração para o desenvolvimento do NetComm foi a farta bibliografia sobre o assunto e farta disponibilidade desse tipo de API para ambiente Windows. Além de diversos *freewares* de excelente qualidade a Microsoft, por exemplo coloca sua implementação de Windows Socket (Wolverine) gratuitamente à disposição de quem tiver uma cópia do *Windows for Workgroups.*

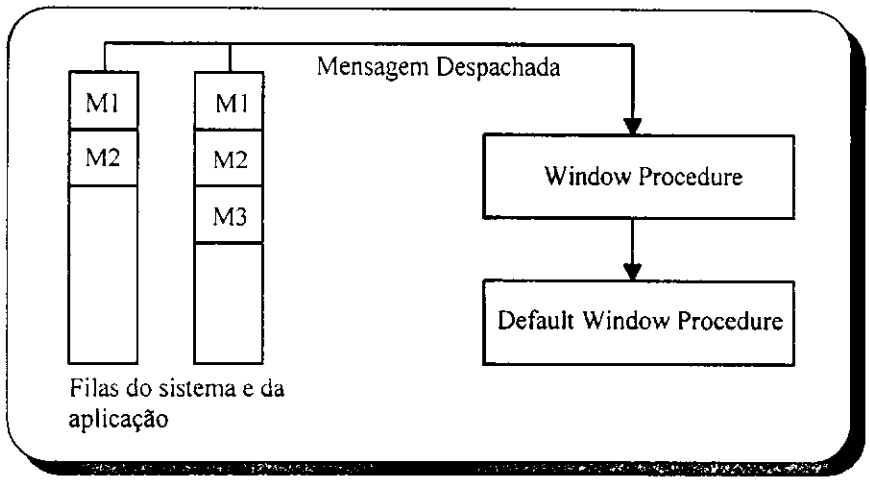

Figura 5.4 - Processamento de mensagens no Windows

Para garantir a portabilidade do NetComm para outras plataformas e tecnologias de rede, desenvolvemos duas classes de objetos denominadas C_Socket e C_Network que dividem o problema em camadas. A classe C_Socket representa um nivel bem baixo (camada 1) que acessa diretamente a API de Windows Socket (camada 0). Já a classe C_Network (camada 2), que incorpora recursos da classe C_Socket, nos garante um poder de abstração maior no desenvolvimento de aplicações distribuidas.

A camada 0 é fornecida com a pilha de protocolos que no nosso caso é uma implementação de Windows Socket. Essa implementação é fornecida como uma DLL (Dynamic Link Library) que pode ser usada por qualquer aplicação Windows. Uma DLL, é uma biblioteca de acesso compartilhado [RITCHTER92]; muitas aplicações podem acessá-la ao mesmo tempo, mas apenas uma instância da mesma reside na memória. Outra característica de uma DLL é que a mesma é link-editada com o código da aplicação dinamicamente e não estaticamente logo após o processo de compilação como é feito normalmente.

A camada 1 representada pela classe C_Socket, é uma forma de garantir independência da camada 0 para as camadas mais altas. Ou seja, caso troquemos a camada 0, coisa que implica em uma provável mudança de API, basta mudar nossa camada 1 mantendo a mesma API para as camadas superiores. A camada 0 é uma DLL, portanto nada impede que tenhamos diferentes implementações da camada 1 para cada arquitetura de rede disponível e que ela seja selecionada em tempo de execução. Ou seja, na hora que o NetComm for ativado ele se encarrega de reconhecer qual o suporte de rede disponível e então carregar a DLL correspondente. A classe

C_Socket só está implementada atualmente para Windows Socket. Essa forma de padronizar o acesso a serviços de rede foi utilizada em [DAVIS93].

A camada 2, representada pela classe C_Network, garante um nível mais alto de abstração para o desenvolvimento de aplicações distribuidas, fazendo uso do paradigma cliente-servidor. Ela esconde certos detalhes de baixo nivel da camada 1, porém nada impede que o desenvolvedor faça uso de serviços da camada 1 se assim desejar. A camada 3, ou camada de aplicação, representa a aplicação do desenvolvedor. A seguir descreveremos melhor a implementação das classes C_Socket e C_Network e mostraremos uma pequena aplicação como exemplo. A figura 5.5 abaixo mostra as camadas da API de rede por nós definida.

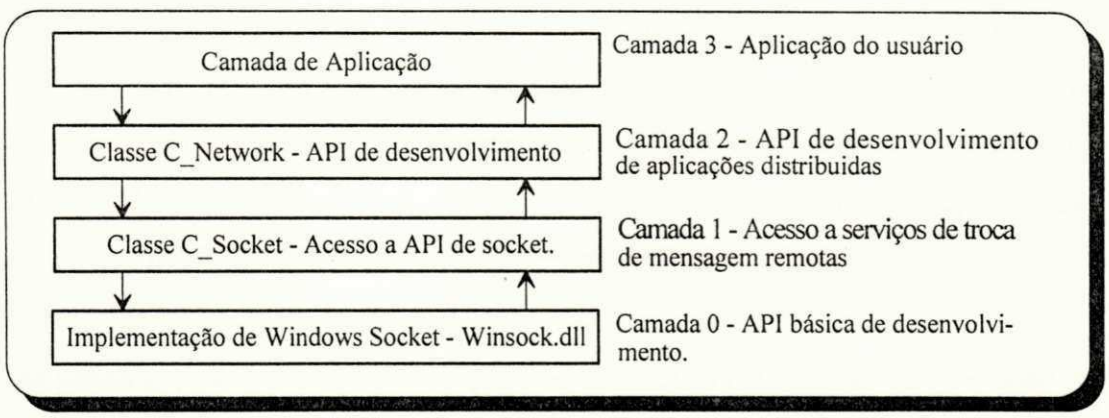

Figura 5.5 - Classes e camadas de acesso a serviços de rede

5.3.2 Classe C_Socket

A classe C_Socket é essencialmente, o mapeamento de uma API baseada em funções para uma API baseada em métodos de objetos. Os métodos foram sendo introduzidos nessa classe sob demanda de desenvolvimento, sendo que a paridade método→função não é um a um. Ou seja, nem todas as funções da API de Windows Socket foram implementadas, bem como alguns métodos da classe C_Socket não têm equivalente, ou mapeiam varias funções, na API de Windows Socket.

A Classe C_Socket herda dois métodos de uma outra classe chamada C_Dispositivo. Essa classe C_Dispositivo é composta apenas por dois métodos virtuais: *Read* e *Write*. A vantagem desse enfoque, é que podemos ter várias classes herdando esses métodos tais como: classe para

acesso a arquivos, memória, *clipboard*, entre outras. Isso significa que a forma de escrever e/ou ler em um arquivo, na memória, no *clipboard* ou em um socket é idêntica.

C_Socket já cuida de garantir uma certa independência em relação à camada 0. A forma de endereçar máquinas, por exemplo, é através do nome da mesma, e não através de estruturas preenchidas pelo usuário como é feito normalmente em Windows Socket.

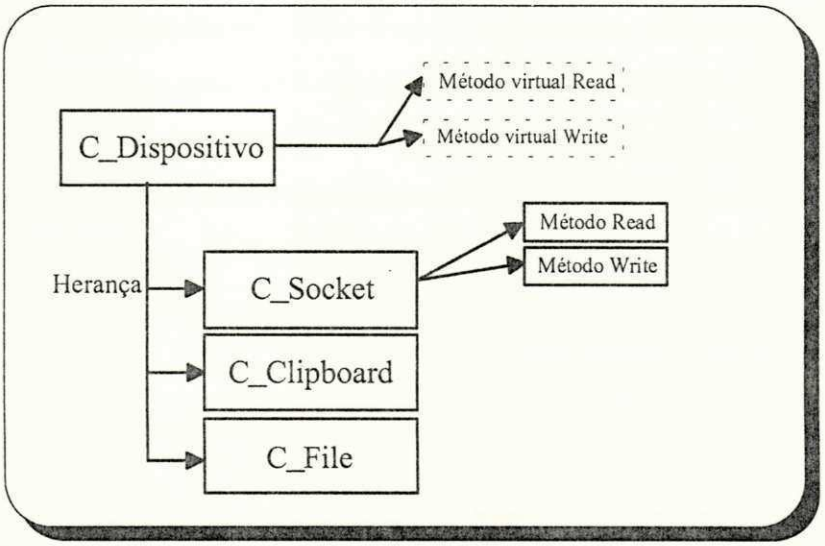

Figura 5.6 - Métodos virtuais na classe C_Socket

A seguir descrevemos os métodos públicos que fazem parte da classe C_Socket:

5.3.3 Classe C_Network

A classe C_Network, desenvolvida como parte deste trabalho é bem mais interessante que a classe C_Socket. Ela se encarrega de garantir a independência da API de acesso a serviços de transporte em um ambiente distribuido bem como manipular os problemas inerentes a um ambiente orientado a eventos como é o caso do Windows.

Ao instanciarmos um objeto da classe C_Network, precisamos determinar o tipo de interação que teremos com ele. Existem quatro métodos básicos na API da classe C_Network para determinar que tipo de objeto estamos construindo: ***SetServerSide***, ***SetClientSide***, ***SetBroadServerSide*** e ***SetBroadClientSide***.

O comportamento de um objeto da classe C_Network difere caso o mesmo seja um servidor ou um cliente. Por isso nos encarregamos de determinar o tipo do objeto logo após sua

criação. Quando queremos marcar um objeto para se comportar como servidor usamos o método *SetServerSide* e *SetClientSide* caso desejemos que o mesmo se comporte como um cliente. A classe C_Network permite que se escolha o protocolo de transporte a ser usado: orientado a conexão ou datagrama. No caso de nossa implementação foram usados os protocolos TCP e UDP [STEVENS90] respectivamente. Caso desejemos criar um objeto que receba e/ou transmita dados para todas as máquinas da rede local (*broadcast*), devemos usar os métodos *SetBroadServerSide* e *SetBroadClientSide*. Nesse caso, datagramas serão utilizados sempre. Isso acontece pelo fato de não ser possível usar um protocolo orientado à conexão quando usamos o *broadcast*; a API de Windows Socket não permite.

Ao definirmos um objeto com o método *SetServerSide* estamos dizendo para o mesmo se amarrar em uma porta conhecida e ficar aguardando pedidos de conexão. O conceito de porta serve para identificar o processo que deve ser notificado quando da transmissão de dados. Ou seja, não basta identificar apenas a máquina para a qual queremos enviar dados; precisamos saber para qual processo da máquina em questão os dados devem ser repassados. Maiores informações sobre o conceito de porta podem ser vistas em [HALL93], em [DAVIS93] e em [STEVENS90]. No caso de definirmos um objeto com o método *SetClientSide*, estamos dizendo para o mesmo se conectar com outra máquina, ou a própria máquina local, na porta pré-determinada pelo servidor.

No momento em que um objeto definido para se comportar como servidor recebe um pedido de conexão, ele cria um outro objeto da classe C_Network. Esse novo objeto é criado pela própria implementação da classe C_Network e se amarra a uma porta qualquer (desconhecida). O objeto criado será responsável pela troca de dados com o objeto que fez o pedido de conexão. O objeto servidor fica então liberado para receber outros pedidos de conexão (figura 5.7). Tal procedimento é importante para o servidor poder atender vários pedidos simultaneamente caracterizando-se como um servidor concorrente. Ou seja, para cada pedido de conexão é criado um novo objeto que será responsável pela troca de informação referente à mesma.

A classe C_Network dispõe de mecanismos internos para suportar servidores concorrentes. Servidores concorrentes têm a vantagem de permitir que múltiplos clientes se conectem a ele. Em um sistema operacional que dispõe de *threads* [CUSTER92][KING94][STEVENS90] e de escalonamento pré-emptivo, como é o caso de algumas versões do Unix ou o NT, é mais simples programar esse tipo de servidor. Ou seja, basta

criar um *thread* ou mesmo um processo distinto, para cada pedido de cliente. No caso do Windows isso se torna um pouco mais complicado. Não existe o conceito de *thread* (pelo menos não na API Win16) e é muito custoso criar outro processo no Windows. A solução então, auxiliado pela programação orientada a objeto, foi criar uma janela para cada instância de objeto da classe C_Network.

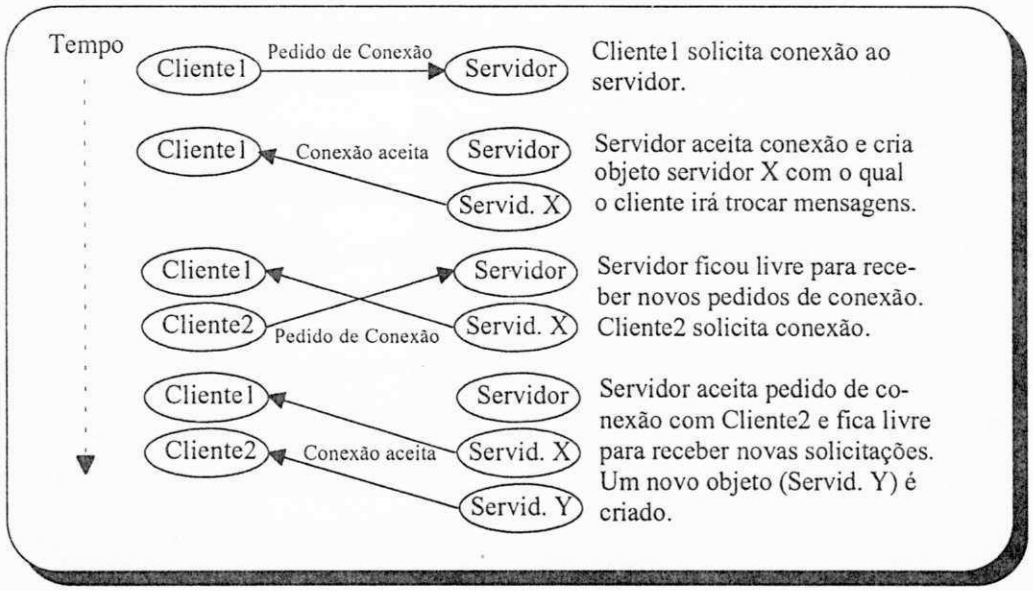

Figura 5.7 - Comportamento dos objetos client e servidor

No ambiente Windows, cada janela é responsável por tratar seus próprios eventos, logo é possível simular a existência de *threads* criando uma janela para cada *thread* desejado. As janelas devem obviamente, ser escondidas, ou seja, não podem aparecer para o usuário. Existem diversas limitações em se usar esse tipo de enfoque, porém foi o única solução encontrada.

A primeira limitação diz respeito ao número de janelas que é possivel criar dentro do ambiente Windows que é limitado a, no máximo, 255 por aplicação. Com isso sabemos que poderemos criar no máximo 255 *threads* por instância de aplicação. Descontando as janelas abertas normalmente por qualquer aplicação Windows e contando que mesmo um botão é considerado uma janela e entra nessa conta, chegamos a um número de aproximadamente 220 *threads* o que é bastante razoável.

A segunda limitação diz respeito a forma do Windows tratar suas janelas. Toda janela criada no ambiente Windows é instância de uma determinada classe. Existem classes pré-definidas como botão, lista de seleção, entre outras, e o programador pode criar suas próprias

classes. Ao criar uma classe o programador precisa definir o comportamento das instâncias da mesma. Por exemplo, quando um botão receber um clique de *mouse*, o comportamento dele é mudar sua aparência para parecer pressionado. De outra forma, ao criar uma classe de janelas, é preciso programar a resposta aos eventos passíveis de ocorrerem para as instâncias da mesma.

O Windows segue uma filosofia distonante em relação a, por exemplo, MOTIF nesse sentido. No MOTIF, para cada janela criada é definida uma função de tratamento de eventos. Ou seja, o controle fica distribuido e isso leva a grande duplicação de código. No WINDOWS, o controle é centralizado na definição da classe. Ou seja, para cada classe existe um procedimento associado de tratamento de eventos e todas as instâncias daquela classe usarão esse procedimento. Existem vantagens e desvantagens no uso de um ou outro enfoque; uma melhor discussão sobre esse assunto pode ser encontrada em [SILVA93].

No caso do Windows, que é o que nos interessa, temos o controle de todos os eventos centralizado em um único procedimento que é chamado de *Windows Procedure*. Esse procedimento é normalmente um enorme comando de decisão múltipla (figura 5.8), onde cada elemento representa o tratamento desejado para a mensagem em questão. Por exemplo, para a mensagem WM_DESTROY (figura 5.8), enviada para uma janela sempre que é solicitada a destruição da mesma, o tratamento é chamar a função PostQuitMessage. Os eventos não tratados na *Window Procedure* do usuário são repassados para uma função *default* existente no Windows chamada *DefWindowProc*.

Para definir uma classe de janelas no ambiente Windows é preciso preencher uma estrutura chamada WNDCLASS [MSOFT92d]. Essa estrutura possui vários campos como: nome da classe, ícone associado à janela, entre outros, e um dos campos dessa estrutura é justamente um apontador para a *Window Procedure*.

No ambiente Windows, cada mensagem é despachada para que seja tratada pela janela a qual a mesma foi endereçada. O tratamento é feito pela *Window Procedure*, porém ela é única para todas as janelas da classe, logo é preciso um mecanismo para identificar para qual das janelas a mensagem se destina. A forma de identificar para qual janela se destina a mensagem é através de um *handle* passado como parâmetro para a Window Procedure (o parâmetro é do tipo HWND como podemos observar na figura 5.8). Além do *handle* da janela, a *Window Procedure* recebe o identificador da mensagem (uMsg) e dois parâmetros associados à mesma (wParam e

lParam, um inteiro sem sinal e um inteiro longo respectivamente). A semântica associada a esses dois parâmetros é dependente do tipo da mensagem.

```
LRESULT CALLBACK
WindowProcedure( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg ){
   case WM_CREATE:
      // Comandos executados antes da criacao da janela
      break;
   case WM_COMMAND:
      switch( wParam ){
      case IDM_FILENEW:
         FileNew();
         break;
      case IDM_EXIT:
         PostQuitMessage( 0 );
         break;
      }
      break;
   case WM_DESTROY:
      PostQuitMessage( 0 );
      break;
   default:
      return( DefWindowProc( hWnd, uMsg, wParam, lParam ) );
   }
   return( 0 );
```

Figura 5.8 - Exemplo de *Window Procedure* na linguagem C++

As extensões introduzidas na API de Windows Socket nos permitem associar uma mensagem a eventos de rede de um determinado socket e a uma janela específica através da função *WSAAsyncSelect*. Existem diversos eventos de rede pré-determinados mas os que nos interessam são três: FD_READ, FD_CLOSE e FD_ACCEPT. O evento FD_READ ocorre sempre que algum dado está disponível para ser lido no socket. O evento FD_CLOSE ocorre quando uma conexão é quebrada por um dos dois lados envolvidos na comunicação e FD_ACCEPT ocorre quando há um pedido de conexão pendente.

Na implementação de sockets de Berkeley, ao se tentar ler um dado em um determinado socket, o processo fica bloqueado até que algo seja recebido. Ao associarmos o evento FD_READ a uma mensagem, estamos dizendo que a mesma será recebida por uma *Window Procedure* e que o tratamento para esse evento se dará dentro dela. O mais interessante é que o evento informa a disponibilidade de dados (no caso de FD_READ), portanto após o recebimento da mensagem teremos a certeza de que poderemos ler dados do socket sem correr o risco de

ficarmos bloqueados. Ou seja, em *Windows Socket* só devemos ler dados de um socket quando temos a absoluta certeza de que os mesmos existem (por isto que existe a função WSAAsyncSelect) [DAVIS93].

```
/***
   Window Procedure for network events.
***/
LRESULT CALLBACK
NetWndProcDef( HWND hWnd, UINT uMessage, WPARAM wParam, LPARAM lParam )
{
   C_Network    *pcnNetwork;

   // Sets the correct pointer to the object
   if( sizeof( C_Network* ) == sizeof( LONG ) ){
      pcnNetwork = ( C_Network* ) GetWindowLong( hWnd, 0 );
   } else {
      pcnNetwork = ( C_Network* ) GetWindowWord( hWnd, 0 );
   }
   switch( uMessage ){
   case NM_SERVERMSG:
      switch( pcnNetwork->tsSide ){
      case S_SRVKNOWN:
         pcnNetwork->HandleSrvMsg( lParam );
         break;
      case S_SRVUNKNOWN:
         pcnNetwork->HandleUnkSrvMsg( lParam );
         break;
      }
      return( 0 );
   case NM_SERVERBROADMSG:
      pcnNetwork->HandleBroadSrvMsg( lParam );
      return( 0 );
   case NM_CLIENTMSG:
      pcnNetwork->HandleClientMsg( lParam );
      return( 0 );
   case NM_CLIENTBROADMSG:
      pcnNetwork->HandleBroadClientMsg( lParam );
      return( 0 );
   case WM_TIMER:
      if( pcnNetwork->InTimerWait() == FALSE ){
         printf( "Erro: Nao pode acontecer***!!!\n" );
         break;
      }
      pcnNetwork->SetInMsg( FALSE );
      pcnNetwork->HandleTimerMsg();
      break;
   }
```

Figura 5.9 - Window Procedure dos objetos da classe C_Network

A Classe C_Network faz uso dessa particularidade de Windows Socket. Cada instância de objeto da classe C_Network possui um handle de janela e para cada um desses objetos uma janela é criada. Ao instanciarmos o primeiro objeto é feito o registro da classe das janelas

associando as mesmas à *Window Procedure* **NetWndProcDef**. Ao definir o tipo de objeto desejado (servidor, cliente, servidor ou cliente de *broadcasts*) também é feita a associação dos eventos de rede que desejamos tratar com mensagens (NM_SERVERMSG, NM_CLIENTMSG entre outras) que serão repassadas através da fila de mensagens. A função NetWndProcDef foi por nós desenvolvida e é extremamente simples como podemos ver na figura 5.9.

O primeiro problema com que a função NetWndProcDef se depara é descobrir com qual instância de objeto estamos tratando. Como falamos anteriormente, uma Window Procedure recebe como parâmetro o *handle* da janela à qual está associada a mensagem em questão. O problema é que o *handle* da janela não é de grande valia para nós, o que precisamos é de um apontador para a instância do objeto que está recebendo os eventos.

Felizmente, o Windows permite que ao criar uma janela sejam alocados alguns bytes adicionais que podem ser acessados posteriormente atraves do *handle* da janela usando as funções *GetWindowWord* ou *GetWindowLong*[15]. Sempre que criamos um objeto da classe C_Network é criada uma janela e nos bytes adicionais que alocamos para a mesma colocamos o apontador para a própria instância que está sendo criada. Isso é feito usando o apontador especial ***this*** do C++. Logo, para saber qual a instância do objeto corrente (para qual os eventos são destinados) basta chamar a função *GetWindowLong* ou *GetWindowWord*. O uso das duas funções se deve ao fato de que no Windows, o tamanho de um apontador depende do modelo de memória em que o código fonte é compilado [RITCHTER92].

Logo após descobrirmos em que instância estamos, tentamos descobrir qual o tipo da instância; se é servidora, cliente, servidora de *broadcasts* ou cliente de *broadcasts* como foi explicado no início desta seção. Após descobrirmos o tipo de objeto em que estamos, chamamos a função de tratamento de eventos correspondente para o mesmo. Como essas funções de tratamento de eventos já são métodos da classe C_Network não precisamos mais nos preocupar em descobrir em que instância estamos. Existe uma função de tratamento de eventos para cada tipo de objeto da classe C_Network e é nelas que verificamos qual o evento de rede que ocorreu: FD_READ, FD_CLOSE ou FD_ACCEPT.

O método construtor da classe C_Network recebe como parâmetro o apontador para uma função de tratamento de eventos do usuário. Essa função do usuário é chamada após o

---

[15] Funções da API do Windows para acessar bytes extras alocados para cada instância de janela.

processamento *default* feito pela classe nas suas próprias funções de tratamento de eventos. É usando essas funções que o programador dá a semântica desejada aos eventos de rede.

Nem todos os eventos de rede são passados para a função de tratamento do usuário. O evento FD_ACCEPT, por exemplo, é tratado pela própria classe; ele é usado para implementar servidores concorrentes e só é usado quando o objeto é do tipo servidor.

Para implementar servidores concorrentes, a classe C_Network implementa uma lista de objetos C_Network dentro de sí própria. Quando uma instância de objeto da classe C_Network do tipo servidor é criada, ela fica amarrada a uma porta determinada esperando por pedidos de conexão (só eventos FD_ACCEPT são tratados). Ao receber um desses pedidos, um novo objeto C_Network é criado e colocado na lista do objeto do tipo servidor. Esse objeto se amarra em uma porta alocada pela própria implementação de Windows Socket e fica responsável pela comunicação com o cliente que pediu a conexão, liberando o objeto do tipo servidor para tratar novas conexões. A esse objeto criado pela propria implementação da classe C_Network denominamos de servidor desconhecido. O servidor desconhecido é que, na realidade, repassa os eventos de rede para a função de tratamento de eventos do usuário. O processamento dos eventos é completamente assíncrono e independente de objeto para objeto, já que cada um deles possui sua própria janela para recepção dos eventos.

O uso da classe C_Network foi indispensável no desenvolvimento do NetComm. A seguir apresentamos uma aplicação cliente-servidor exemplo usando a classe C_Network.

5.3.4 Exemplo de aplicação usando a classe C_Network

Como exemplo de aplicação usando a classe C_Network, apresentamos o Ecoa. O Ecoa é uma aplicação cliente-servidor, e o que ela faz é simplesmente ecoar de volta tudo o que é recebido. Ecoa é dividido em Ecoa Cliente e Ecoa Servidor. O servidor fica aguardando mensagens do cliente e todo dado recebido é enviado de volta como um eco.

5.3.4.1 Ecoa Cliente

O módulo Ecoa Cliente se divide em três funções. A primeira, *WinMain*, é o ponto de entrada de todo programa Windows e é equivalente à função *main* de um programa escrito em C. A segunda função, *Ecoa*, é responsável por fazer a conexão com o servidor e enviar a mensagem

a ser ecoada. A terceira função, *EcoaUserProc*, é responsável por tratar os eventos de rede associados ao objeto C_Network criado pela função Ecoa. Ela fica esperando a resposta do servidor e caso ela chegue a imprime. Caso contrário (*timeout*) é impressa a mensagem de erro "Tempo esgotado esperando resposta do servidor".

```
//
// Module: ECOACLNT.CPP
// Description:
//
//          Ecoa client module
//
// Programmer: Marcos Sebastian Alsina
//
// Last update: 06/08/94
//


#if !defined(__WINDOWS_H)
#include <windows.h>
#endif                  // __WINDOWS_H
#include <netcl.h>

// Global variables (only for this file)
static C_Network   *_pcnClient;         // client object

// Global variables (non-statics)
extern HINSTANCE          _hInstance;          // defined in windows.h

// Functions
#pragma war -arg   // desativa warnings
int PASCAL
WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int iCmdShow )
{
        MSG     msg;
        char    *szParam1;
        char    *szParam2;

        _hInstance = hInstance;
        _InitEasyWin();
        // Opens the socket DLL
        if( OpenSocketDLL() != 0 ){
                MessageBox( hWnd, "Problemas abrindo DLL!\n", "ERRO", MB_OK );
                exit( -1 );
        }
        if( !GetParam( lpszCmdLine, szParam1, szParam2 ) ){
                printf( "Sintaxe: ecoaclnt NomeMaquina Mensagem\n" );
                exit( -1 );
        }
        if( Ecoa( szParam1, szParam2 ) != NET_OK ){
                printf( "Erro ao criar cliente\n" );
```

```
                CloseSocketDLL();
                exit( -1 );
        }

        // Loop de mensagens
        while( GetMessage( &msg, NULL, 0, 0 ) ){
                TranslateMessage( &msg );
                DispatchMessage( &msg );
        }
        CloseSocketDLL();
        if( _pcnClient != NULL ){
                delete _pcnClient;
        }
        return( msg.wParam );
}


/***
        Faz conexão com servidor "szMachineName" e envia
        mensagem para ser ecoada.
***/
int
Ecoa( char *szMachineName )
{
        int     iErr;

        if( ( _pcnClient = new C_Network( EcoaUserProc ) ) != NULL ){
                if( !_pcnClient->InMsg() ){
                        _pcnClient->SetInMsg( TRUE );
                        if( ( iErr = _pcnClient->SetClientSide( szMachineName, "tcp", SERVERPORT )) ==
NET_OK ){
                                _pcnClient->SetTimerId();
                                _pcnClient->Write( szMessage, strlen( szMessage ) + 1 );
                                return( NET_OK );
                        }
                }
                delete _pcnClient;
                return( iErr );
        }
        return( E_NETCREATEOBJ );
}


/***
        Funcao do usuario para tratamento das mensagens
        recebidas do servidor.
***/
void
EcoaUserProc( C_Network *pcnNetwork, UINT uMsg )
{
        char    szBuf[MAXPACKSIZE];
```

```
        switch( uMsg ){
        case FD_READ:
                if( pcnNetwork->Read( &szBuf, MAXPACKSIZE ) <= 0 ){
                        printf( "problemas lendo o eco do servidor\n" );
                } else {
                        printf( "Ecoa: %s\n", szBuf );
                }
                delete _pcnNetwork;
                PostQuitMessage( 0 );
                break;
        case WM_TIMER:
                printf( "Tempo esgotado esperando resposta do servidor\n" );
                delete _pcnNetwork;
                PostQuitMessage( 0 );
                break;
        }
}
```

A função GetParam não foi aqui apresentada. Ela se encarrega de extrair os parâmetros passados por linha de comando devolvendo os apontadores para os mesmos. O código em si é bastante simples: fazemos a conexão com o servidor e aguardamos a resposta na função definida do usuário *EcoaUserProc*.

5.3.4.2 Ecoa Servidor

O módulo Ecoa Servidor possui também três funções básicas: WinMain que foi explicada na seção 5.3.4.1, EcoaSrv e EcoaSrvUserProc. As funções são análogas às do módulo Ecoa Cliente. A diferença é que a função EcoaSrv cria um objeto C_Network do tipo servidor e que a função EcoaSrvUserProc, responde às mensagens do cliente enviando os mesmos dados recebidos (eco). Caso seja recebida a mensagem especial "MORRA" o servidor destroi os objetos e abandona o processamento.

```
//
// Module: ECOASERV.CPP
// Description:
//
//          Ecoa server module
//
// Programmer: Marcos Sebastian Alsina
//
// Last update: 08/08/94
//
```

```
#if !defined(__WINDOWS_H)
#include <windows.h>
#endif                  // __WINDOWS_H
#include <netcl.h>

// Global variables (only for this file)
static C_Network   *_pcnServer;        // client object
// Global variables (non-statics)
extern HINSTANCE          _hInstance;          // defined in windows.h

// Functions
#pragma war -arg   // desativa warnings
int PASCAL
WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int iCmdShow )
{
        MSG     msg;

        _hInstance = hInstance;
        _InitEasyWin();
        // Opens the socket DLL
        if( OpenSocketDLL() != 0 ){
                MessageBox( hWnd, "Problemas abrindo DLL!\n", "ERRO", MB_OK );
                exit( -1 );
        }
        if( EcoaSrv() != NET_OK ){
                printf( "Erro ao criar servidor\n" );
                CloseSocketDLL();
                exit( -1 );
        }

        // Loop de mensagens
        while( GetMessage( &msg, NULL, 0, 0 ) ){
                TranslateMessage( &msg );
                DispatchMessage( &msg );
        }
        CloseSocketDLL();
        if( _pcnServer != NULL ){
                delete _pcnServer;
        }
        return( msg.wParam );
}


/***
        Cria servidor de ecos
***/
int
EcoaSrv( void )
{
        int         iErr;

        if( ( _pcnServer = new C_Network( EcoaSrvUserProc ) ) != NULL ){
```

```
                if( ( iErr = _pcnServer->SetServerSide( "tcp", SERVERPORT )) == NET_OK ){
                        return( NET_OK );
                }
                delete _pcnClient;
                return( iErr );
        }
        return( E_NETCREATEOBJ );
}


/***
        Funcao do usuario para tratamento das mensagens
        recebidas do servidor.
***/
void
EcoaSrvUserProc( C_Network *pcnNetwork, UINT uMsg )
{
        char    szBuf[MAXPACKSIZE];

        switch( uMsg ){
        case FD_READ:
                if( pcnNetwork->Read( szBuf, MAXPACKSIZE ) <= 0 ){
                        printf( "problemas lendo mensagem do cliente.\n" );
                } else {
                        if( strcpy( szBuf, "MORRA" ){
                                PostQuitMessage( 0 );
                                delete pcnNetwork;
                        } else {
                                printf( "Ecoa: %s\n", szBuf );
                                // respondendo com a mesma mensagem
                                pcnNetwork->Write( szBuf, MAXPACKSIZE );
                        }
                }
                break;
        }
}
```

5.4 Recuperação de estado do NCClient

O processo de recuperação de estado de um processo distribuido é quase sempre difícil e penoso para o programador. São muitas as situações de erro que podem ocorrer durante o processo de recuperação de estado. Nesta seção apresentamos o processo de recuperação de estado do NCClient. O NCServer é um processo que não guarda estado, portanto sua inicialização é bem mais simples.

O NCClient mantém uma série de arquivos e diretórios que precisam ser atualizados no caso de queda. Ao ser ativado, o NCClient pergunta aos servidores, quem tem e quais são os

projetos para o usuário correntemente logado. Isso é feito através da mensagem WhoHasHead, que é do tipo *broadcast*. Os servidores ao receberem tal mensagem verificam se eles dispõem de algum projeto destinado a esse usuário e em caso positivo respondem ao NCClient.

O NCClient armazena durante um certo período de tempo, todas as respostas dos servidores em uma lista encadeada, contendo basicamente o endereço do servidor e o handle do projeto. Após passado o tempo de espera de respostas, o NCClient se encarrega de varrer a lista e pedir o cabeçalho do projeto correspondente ao handle. O handle do projeto é único na rede, e se manterá o mesmo em caso de queda do servidor já que o mesmo é armazenado em arquivo pelo servidor.

Para cada cabeçalho de projeto recebido é criado um diretório de armazenamento e o nome desse cabeçalho é armazenado em um arquivo de mapeamento contendo o *handle* do projeto como chave e o nome do diretório. A diferença entre o que foi recebido e o que havia anteriormente é removida. Ou seja, se um determinado diretório para armazenamento de projetos existia, tinha seu nome mapeado no arquivo de mapeamento e esse cabeçalho não foi novamente recebido é porque provavelmente o mesmo deixou de existir ou porque o servidor não está ativo. Em qualquer um dos casos, o diretório é removido bem como o registro no arquivo de mapeamento.

Cuidados especiais devem ser tomados em relação à quedas do servidor ou do próprio cliente durante operações lentas como a transferência de dados ou projetos. Para resolver o problema foi criado um mecanismo de transação [MULLENDER89] no qual a operação só é dada como válida após sua conclusão. Qualquer arquivo de dados recebido pelo NCClient é inicialmente gravado com um nome temporário e após a conclusão da operação de transferência o mesmo é movido (re-nomeado) para seu respectivo diretório de armazenamento. Ao entrar no ar o NCClient remove os arquivos temporários que por ventura existam e uma nova solicitação de dados é feita.

Capítulo 6 - Considerações finais

6.1 Conclusão

O NetComm é um software que dá um novo enfoque à maneira de gerenciar redes de computadores e à maneira com que os usuários de uma rede acessam e trocam a informação em uma organização. Suas características de software multimídia distribuído garantem um grande número de evoluções possíveis e abrem perspectivas de novos trabalhos.

O NetComm permite que um grande número de aplicações possam ser construídas em torno de seu núcleo. Ele poderia ser, por exemplo, adaptado como software para a área de educação permitindo que alunos e/ou professores usassem seu editor de animações para construir simulações ou apresentações.

Propusemos desenvolver um software para comunicação entre usuários de uma rede local e gerência da mesma. Este objetivo foi alcançado incorporando ao NetComm, recursos avançados como interface gráfica, multimídia e eventos distribuídos e abrimos caminho para que um grande número de novas aplicações pudesse ser construído em torno do mesmo. O estudo de eventos distribuídos mostrou que os mesmos são uma ferramenta poderosa, mas que precisa ser levado em consideração um grande número de problemas.

6.2 Trabalhos futuros

O NetComm é um *software* que permite um grande número de evoluções. A forma com que o NetComm foi projetado permite que novas dimensões sejam incorporadas, bem como a expansão das já definidas neste trabalho. A seguir mostraremos algumas direções futuras em relação ao NetComm.

- Incorporação de imagens de vídeo aos projetos
- Sistema de correio eletrônico multimídia.
- Uso de mensagens interativas.

- Vídeo-conferência.
- Possibilidade de troca de projetos entre diferentes redes locais.
- Construção de *gateways* para sistemas de correio eletrônico.
- Incorporação de algoritmos de compressão de dados.
- Adaptação do protocolo a novos transportes em redes locais de alta velocidade.
- Construção de um simulador para validação do protocolo.
- Usar uma implementação de eventos distribuídos que trate eventos de *software* e/ou eventos interativos.
- Tornar os projetos e o NetComm independentes de plataforma.

Como primeira evolução do NetComm pensamos na incorporação de imagens de vídeo aos projetos. Isso pode ser feito simplesmente incluindo um novo tipo de objeto na dimensão "O que" do projeto. As demais dimensões não seriam alteradas.

Sistemas de correio eletrônico talvez se constituam na aplicação mais comum para grupos de trabalho conectados a redes locais. Uma limitação do NetComm é não permitir a gravação de projetos para posterior leitura. Fazendo isso, e incorporando o conceito de caixa postal, teremos um correio eletrônico. A dimensão "Como", poderia incorporar essa nova forma de apresentar a informação. A dimensão "Quando" do projeto poderia indicar o momento de avisar o destinatário, da chegada de uma nova mensagem em sua caixa postal. Desta forma, poderíamos enviar uma mensagem da qual o usuário só tomaria conhecimento no momento que desejássemos. As possibilidades nesse caso são enormes. Por exemplo, poderíamos construir um projeto com dados dinâmicos e definir a dimensão "Quando" de forma que o projeto fosse reenviado toda segunda-feira. Neste caso, o destinatário teria uma mensagem nova em sua caixa postal todas as segundas-feiras com os dados dinâmicos atualizados. De graça, ganhamos um correio multimídia, já que o NetComm permite que se incluam sons e gráficos em um projeto além de, futuramente, imagens de vídeo.

Uma outra limitação com o NetComm é que a comunicação não é interativa. Ou seja, um projeto é enviado e apresentado em outra máquina, porém não há uma resposta imediata do outro lado a menos que o destinatário construa um projeto como resposta. Seria interessante que pudéssemos ter um sistema como o *Talk*, em que a comunicação é interativa. O *Talk* é um

software que permite que um usuário se comunique com outro em redes TCP/IP abrindo um canal de comunicação no qual tudo o que um dos usuários escreve é imediatamente apresentado no terminal do outro. Essa nova forma de comunicação poderia ser feita incluindo um novo atributo de interatividade à dimensão Como de um projeto. Com isso, os objetos poderiam ser colocados dinamicamente em um projeto tanto do lado construtor como do lado receptor. Outro enfoque a ser considerado em relação à interatividade dos projetos, poderia ser a inclusão de objetos interativos tais como botões ou listas de seleção. Desta forma, o usuário receptor de um projeto poderia alterar a seqüencia de apresentação do mesmo. Objetos interativos garantem um poder ainda maior ao NetComm que poderia ser usado como, por exemplo, um sistema multimídia de ajuda como os encontrados em estações de trem, bancos, grandes supermercados e *shopping centers*.

Uma consequência imediata da possibilidade de enviar mensagens interativas é a construção de um sistema de vídeo-conferência. A partir do momento que pudermos incluir imagens de vídeo em projetos NetComm e que dispusermos de mensagens interativas teremos um sistema de vídeo-conferência. A partir do momento em que abríssemos um canal *full-duplex* de comunicação poderíamos ir acrescentando imagens de vídeo ou mesmo sons ao projeto e receber a resposta imediatamente.

O uso de broadcast no protocolo restringe o uso do NetComm a redes locais. Em uma futura evolução o protocolo poderia ser adaptado para permitir que projetos pudessem ser trocados entre redes distintas, inclusive de longa distância com canais de alta velocidade. Para isso, *gateways* poderiam ser utilizados de forma a propagar o *broadcast* entre distintas redes.

Uma consequência imediata da adaptação do NetComm para serviços de correio eletrônico seria a construção de gateways para os principais sistemas desse tipo, como X.400, VIM , MHS[16] entre outros.

O NetComm é um software que trabalha com grandes volumes de dados. Os arquivos de gráficos, sons e vídeo são grandes consumidores de recursos. Algoritmos de compressão específicos para esse tipo de dados se fazem necessários para consumir menos recursos da máquina, seja espaço em disco ou tempo de transmissão pela rede. Uma evolução para o NetComm seria desenvolver novos algoritmos de compressão de dados específicos para o tipo de

---

[16] Protocolos para acesso a serviços de correio eletrônico.

informação manipulada e incorporá-los ao mesmo. Os algoritmos para compressão de imagens gráficas e de vídeo já são bem conhecidos e muitos de domínio público (TIFF, JPEG, MPEG entre outros). Os algoritmos para compressão de arquivos de som como o *Wave* porém, não são ainda suficientemente eficientes e poderiam ser tema de estudo de novos trabalhos de dissertação.

A evolução das arquiteturas de redes de alta velocidade como FDDI (*Fiber Distributed Data Interface*) fizeram surgir novos protocolos de transporte específicos para essas redes. Esses protocolos têm recursos que facilitam a programação de aplicações multimídia distribuídas como por exemplo, *multicast*. O *multicast* é uma forma de enviar dados para um determinado conjunto de máquinas. Diferentemente do *broadcast* e do *unicast* que enviam dados para todas ou uma máquina respectivamente. O protocolo de troca de projetos do NetComm poderia ser adaptado para incorporar essa característica em uma futura versão do software para redes de alta velocidade. O protocolo poderia substituir o uso do *broadcast* pelo *multicast* ganhando com isso um fator de escalabilidade maior.

O protocolo para troca de projetos (PTP) precisa ser validado em situações de carga extrema. Um simulador poderia ser construído para tal tarefa. Com ele poderíamos observar o comportamento do protocolo em situações que não nos foram possíveis de testar (uma rede com 1000 máquinas, por exemplo).

A implementação de eventos distribuídos apresentada trabalha apenas com eventos de *hardware* e mais especificamente de *mouse* e teclado. Uma nova implementação poderia tratar eventos de *software* bem como implementar o conceito de eventos interativos apresentado no capítulo 2.

Como grande e último desafio para a evolução do NetComm citamos a possibilidade de usar o *software* em ambientes heterogêneos. Para isso seria preciso usar uma representação de dados independente de máquina e construir rotinas de suporte ao tipo de dados manipulados pelo NetComm (.BMP, .WAV) às novas plataformas. A interface do NetComm teria que também ser repensada para se adaptar às GUIs dessas novas plataformas.

Bibliografia

[ADAMS93] Adams, L. *C for Windows Animation Programming*, McGraw-Hill, 1993

[BLOOMER91] Bloomer, J. *Power Programming with RPC*, O'Reilly &Associates, Inc. 1991

[BORLAN93] Borland International, Inc. *Borland C++ 4.0: Programmer's Guide*, Borland International, Inc. 1993.

[BOYCE92] Boyce, J. *Inside Windows for Workgroups*, New Riders Publishing, 1992.

[BRSCHMIDT93] Brockschmidt, K. *Programming for Windows with Object Linking and Embedding 2.0*, Microsoft Press, 1993.

[BUGG93] Bugg, K. E. & Tackett J. *Implementing and Using Windows Help*, The C Users Journal, Sep. 1993, pp. 63-72.

[CLARK92] Clark, J. D. *Windows Programmer's Guide To OLE/DDE*, Prentice Hall, 1992.

[CUSTER92] Custer, H. *Inside Windows NT*, Microsoft Press, 1992.

[DAVIS93] Davis, R. *Windows Network Programming: How to Survive in a World of Windows, DOS, and Networks*, Addison-Wesley Publishing Company, 1993.

[EDDON93] Eddon, G. R. *Fundamental Techniques for Sprite Animation in Windows-based Applications*, Microsoft System Journal, Dec. 1993, pp. 81-87.

[HALL93] Hall, M. & Towfiq, M. & Arnold, G. & Treadwell, D. & Sanders, H. *Windows Sockets: An Open Interface for Programming under Microsoft Windows, Version 1.1*, Internet Document, 1993.

[JOSLIN93] Joslin, P. *Using the Windows DIB Color Table*, The C Users Journal, Sep. 1993, pp. 27-48.

[KING94] King, A. *Memory Management, the Win32 subsystem, and Internal Synchronization in Chicago*, Microsot System Journal, May 1994, pp. 57-61.

[MSOFT92a] Microsoft Corporation. *The Windows Interface: An application Design Guide*, Microsoft Press, 1992.

[MSOFT92b] Microsoft Corporation. *Microsoft Windows 3.1 Programmer's Reference, Vol. 1*, Microsoft Press, 1992.

[MSOFT92c] Microsoft Corporation. *Microsoft Windows 3.1 Programmer's Reference, Vol. 2*, Microsoft Press, 1992.

[MSOFT92d] Microsoft Corporation. *Microsoft Windows 3.1 Programmer's Reference, Vol. 3*, Microsoft Press, 1992.

[MSOFT92e] Microsoft Corporation. *Microsoft Windows 3.1 Programmer's Reference, Vol. 4*, Microsoft Press, 1992.

[MSOFT92f] Microsoft Corporation. *Microsoft Windows 3.1: Guide to Programming*, Microsoft Press, 1992.

[MULLENDER89] Mullender, S. *Distributed Systems*, ACM Press, Addison-Wesley, 1989.

[OMG91] Object Management Group & X/Open. *The Common Object Request Broker: Architecture and Specification*, OMG Document Number: 91.12.1, Revision 1.1, 1991

[PETZOLD90] Petzold, C. *Programming Windows: The Microsoft Guide to writing applications for Windows 3*, Microsoft Press, 1990.

[PIETREK93] Pietrek, M. *Windows Internals: The Implementation of the Windows Operating Environment*, Addison-Wesley, 1993.

[RITCHTER92] Richter, J. M. *Windows 3.1: A Developer's Guide, 2nd Edition*, M&T Books, 1992.

[RYMER93] Rymer, J. R. *Expersoft XShell: a speedy ORB with large ambitions*, Distributed Computing Monitor, Nov. 1993, vol. 8, n. 11, pp. 15-27.

[SAKS93] Saks, D. *Stepping Up to C++: Recent Language Extensions to C++*, The C Users Journal, Jun. 1993, pp. 117-124.

[SANTOS93] Santos, G. J. *Considerações para o Desenvolvimento de Aplicações Distribuídas em Ambientes Heterogêneos*, Dissertação de Mestrado, UFPB, 1993.

[SILVA93] Silva, L. G. *UNIFIC: Uma API para Unificação da Programação entre diferentes GUIs*, Dissertação de Mestrado, UFPB, 1993.

[STALLINGS93] Stallings, W. *SNMP, SNMPv2, and CMIP the practical guide to network management standards*, Addison-Wesley Publishing Group, 1993.

[STEVENS90] Stevens, W. R. *Unix Network Programming*, Prentice Hall, 1990.

[TANENBAUM92] Tanenbaum, A. S. *Modern Operating Systems*, Prentice Hall, 1992

Glossário

- **Ambientes gráficos** - O mesmo que interfaces gráficas
- **API (*Application Programming Interface*)** - Padronização de um conjunto de funções usadas pelo programador para acessar determinados serviços.
- **Aplicação em *background*** - Aplicação executada na retaguarda enquanto o usuário trabalha em outras tarefas.
- **Aplicação remota** - Aplicação executada em máquina remota.
- ***Bitmap*** - Padrão para o armazenamento de imagens.
- ***Broadcast*** - Mecanismo usado para transferir dados para todas as máquinas de uma rede local indistintamente.
- **Caixa de ferramentas** - Nome dado a um tipo gráfico de controle formado por um conjunto de botões ou ícones.
- **Caixa de diálogo** - Tipo de janela de interface com o usuário usada normalmente para entrada de dados.
- ***Calendar.exe*** - Aplicação de agendamento de compromissos que acompanha o Windows.
- **CD-ROM** - Dispositivo óptico de armazenamento de informações.
- ***Chicago*** - Nome usado para indicar a nova versão 32 bits do Windows.
- **Classes (de objetos)** - Paradigma de programação usado para definir características e comportamentos comuns a grupos de objetos.
- **Cliente** - Nome dado ao processo que solicita serviços de outro em um ambiente distribuído.
- **Cliente-servidor** - Modelo de programação de aplicações distribuídas no qual um processo (Servidor) oferece serviços para outro (Cliente).
- ***Clipboard*** - Área de compartilhamento de dados entre aplicações no ambiente Windows.
- **Composição (de objetos)** - Diz-se que existe uma composição quando um objeto contêm outro.

- **Computação distribuída** - Nome dado à área da computação que trata de assuntos relacionados ao processamento distribuído da informação usando redes de transmissão de dados para tal.
- **Conexão** - Neste contexto uma conexão é o resultado de uma solicitação de comunicação entre processos distintos usando protocolos orientados a conexão como TCP, por exemplo.
- **CORBA (*Common Object Request Broker Architecture*)** - Definição de uma arquitetura de objetos distribuídos.
- **Correio eletrônico** - Aplicação que permite a troca de mensagens entre usuários de uma rede.
- **DDE *(Dynamic Data Exchange)*** - Protocolo usado para compartilhamento de dados entre aplicações. Originalmente desenvolvido para ambiente Windows.
- **DefWindowProc** - Nome dado à função *default* para tratamento de mensagens no ambiente Windows.
- **DIB (*Device Independent Bitmap*)** - *Bitmap* independente de dispositivo de saída.
- **DLL (*Dynamic Link Library*)** - Biblioteca compartilhada ou de carga dinâmica na nomenclatura Windows.
- **DOS** - Sistema operacional mono-usuário para PCs.
- **Driver** - Nome dado a programas que controlam dispositivos de *hardware*.
- **Eventos** - Neste contexto, evento é qualquer coisa que ocorre seja por intervenção do usuário, do *hardware*, de dispositivos externos, do gerenciador de janelas, do sistema operacional ou das aplicações que executam sob o mesmo e que a aplicação toma conhecimento através de uma fila de mensagens.
- Eventos remotos - Eventos que ocorrem em máquinas remotas.
- **FDDI (*Fiber Distributed Data Interface*)** - Rede de alta velocidade usando fibra ótica como meio físico
- ***Freeware*** - Software de distribuição gratuita.
- ***Gateway*** - Tradutor de protocolos normalmente implementado por *software*.
- **GDI (*Graphic Device Interface*)** - API do Windows para acesso a serviços gráficos.

- **Gerência de redes** - Área da computação que cuida de aspectos de administração de redes de computadores, aplicações e máquinas remotas.
- **Gravador de macros** - Aplicação que armazena e reproduz um determinado conjunto de eventos associados à mouse e teclado.
- ***Grid*** - Grade lógica usada para determinar a precisão com que o usuário acessa objetos gráficos.
- ***Groupware*** - Software para grupos de trabalho conectados a redes de computadores.
- Grupo StartUp (do Windows) - Grupo especial de ícones. Todas as aplicações nesse grupo são executadas automaticamente no momento da inicialização do Windows.
- ***Handle*** - Nome genérico dado a um manuseador ou identificador de objetos.
- **Herança (de objetos)** - Diz-se que existe uma herança quando um objeto herda características de outro.
- **Hypertexto** - Técnica de organização da informação em uma forma hierárquica na qual a mesma pode ser acessada através de referências.
- **Ícone** - Objeto gráfico cujo desenho indica sua funcionalidade.
- **Interface gráfica** - Forma gráfica de apresentar a informação ao usuário e de interação com o mesmo.
- **Internet** - Maior rede de computadores do mundo.
- **IPX/SPX** - Pilha de protocolos usado pelo sistema operacional Netware.
- **JPEG** - Padrão para o armazenamento de imagens em forma compactada.
- **Login** - Processo de identificação de um usuário para permitir seu acesso a recursos compartilhados.
- **Lotus Organizer** - Aplicação tipo PIM da Lotus.
- **MAPI** - Definição de uma API para acesso a serviços de correio eletrônico da Microsoft.
- **Mensagem** - Estrutura de dados usada para representar eventos ou mecanismo para troca de informação entre aplicações.
- **Método destrutor** - Método executado antes da destruição de um objeto.
- **Método construtor** - Método executado antes da criação de um objeto.

- **Método** - Terminologia usada para indicar funções ou procedimentos que determinam o comportamento de um objeto.
- **Métodos virtuais** - Métodos não implementados dentro de uma determinada classe e sim dentro das classes que herdam características da mesma.
- **MHS** - Protocolo para acesso a serviços de correio eletrônico usado pela Novell.
- **Motif** - Exemplo de interface gráfica
- ***Mouse*** - Dispositivo de interação com o usuário como o teclado.
- **MPEG** - Padrão para o armazenamento de imagens gráficas animadas.
- ***Multicast*** - Semelhante ao *broadcast*, com a diferença que as mensagens são endereçadas a um conjunto de máquinas e não a todas.
- **Multimídia** - Expressão utilizada para indicar a interação usando diversos tipos de mídia tais como: sons, gráficos, textos.
- **NetBeui** - Protocolo de comunicação usado pela rede Lan Manager.
- **Netware** - Sistema operacional de rede da Novell.
- Objetos distribuídos - Paradigma de programação distribuída usando objetos.
- **ODBC *(Open Database Connectivity)*** - Padrão desenvolvido pela Microsoft para o acesso à Bancos de Dados.
- **OLE *(Object Linking and Embedding)*** - Protocolo para o compartilhamento de dados entre aplicações desenvolvido pela Microsoft.
- **ONC *(Open Network Computing)*** - Termo usado pela SUN para as ferramentas componentes de sua arquitetura distribuída.
- ***Open Look*** - Exemplo de interface gráfica.
- **PIM (*Personal Information Manager*)** - Nome dado a programas de gerenciamento pessoal.
- **Ponto-a-ponto** - Especialização da arquitetura cliente-servidor na qual todo mundo pode ser cliente e servidor ao mesmo tempo.
- **Porta** - Forma de identificar um serviço remoto. O endereço identifica a máquina, enquanto a porta identifica o processo.
- ***PostMessage*** - Função da API do Windows usada para enviar mensagens para a fila da aplicação.

- **Pré-emptivo** - Um sistema operacional é pré-emptivo quando ele se encarrega de mudar o contexto entre as aplicações, determinando quem deve executar e quando.
- **Processo** - Um processo representa uma unidade de execução sob um sistema operacional.
- **PTP (Protocolo para troca de projetos)** - Protocolo usado pelo NetComm.
- ***Recorder*.exe** - Aplicação para gravação/reprodução de macros que acompanha o Windows.
- **RPC (*Remote Procedure Call*)** - Paradigma de programação de aplicações distribuídas usando o mecanismo de chamada a funções remotas.
- ***Screen-saver*** - Nome dado a programas que usam imagens animadas para evitar que o fósforo dos monitores de vídeo queimem em pontos específicos. Eles são ativados sempre em períodos de inatividade do usuário.
- **Servidor** - Processo que oferece serviços para outros
- **Servidor concorrente** - Servidor que atende múltiplos pedidos ao mesmo tempo.
- **SNMP (*Simple Network Management Protocol*)** - Protocolo para gerência de redes.
- ***Socket*** - Descritor usado para acessar diretamente serviços de transporte.
- ***Sound Blaster*** - Dispositivo multimídia para PCs.
- ***Spin-box*** - Tipo de controle gráfico que permite edição direta ou através de botões que incrementam ou decrementam o valor desejado.
- ***Task List*** - Utilitário do Windows que mostra as aplicações em execução.
- **TCP** - Protocolo de transporte orientado a conexão.
- **TCP/IP** - Nome dado à pilha de protocolos que formam a arquitetura de redes TCP/IP.
- ***Templates*** - Extensão da linguagem C++ que permite que uma mesma classe trabalhe com diferentes tipos de dados sem que seja necessária sua recodificação.
- ***Thread*** - Também chamado de *Light Weight Process* (processo leve). É um processo normal com a diferença que a área de dados é comum ao pai e ao filho (*thread*). Cada *thread* possui sua própria pilha de execução.
- **TIFF** - Padrão para o armazenamento de imagens com alto nível de compressão.

- **Transação** - Uma transação é um grupo indivisível de funções. A transação só é válida se todos os passos necessários para sua conclusão tiverem sido efetuados. Caso contrário, os passos são desfeitos.
- **UCP** - Unidade Central de Processamento
- **UDP** ***(User Datagram Protocol)*** - Protocolo de transporte não orientado a conexão.
- **Unix** - Sistema operacional multiusuário
- **Vídeo-conferência** - forma de comunicação em que vários usuários se comunicam entre si usando imagens de vídeo e som em tempo real.
- **VIM** (***Vendor Independent Messaging***) - Protocolo para acesso a serviços de correio eletrônico usado pela Lotus.
- **VOC** - Padrão para o armazenamento de sons.
- **WAV** - Padrão para o armazenamento de sons.
- **Win16** - API de 16 bits do Windows.
- **Win32** - API de 32 bits do Windows
- ***Window Procedure*** - Função responsável pelo tratamento de eventos relacionados a uma determinada classe de janelas do Windows.
- ***Windows for Workgroups*** - Versão do Windows para grupos de trabalho formando uma rede ponto-a-ponto.
- ***Windows*** - Mescla de interface gráfica e sistema operacional da Microsoft.
- ***Windows NT*** - Sistema operacional de 32 bits da Microsoft.
- ***Windows Socket*** - API para acesso a serviços de transporte no Windows
- ***WinMain*** - Função que serve de ponto de entrada para os programas para Windows.
- **WNDCLASS** - Estrutura de dados que define uma classe de janelas no Windows
- **WOSA** (***Windows Open Services Architecture***) - Conjunto de padrões e APIs que determinam o acesso a serviços dos mais diversos.
- **X.400** - Protocolo para acesso a serviços de correio eletrônico.