# Aplicação em Tcl/Tk para Protocolo Multidestinatário


UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

ESCOLA DE ENGENHARIA

DEPARTAMENTO DE ELETRÔNICA

Autor:

Marcelo Dias de Amorim

Orientador:

Prof. Otto Carlos Muniz Bandeira Duarte

Examinador:

Prof. Aloysio de Castro Pinto Pedroza

Examinador:

Prof. Mauros Campello Queiroz



DEL

Março de 1996

# Aplicação em Tcl/Tk para Protocolo Multidestinatário

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

ESCOLA DE ENGENHARIA

DEPARTAMENTO DE ELETRÔNICA

Autor:

Marcelo Dias de Amorim

Orientador:

Prof. Otto Carlos Muniz Bandeira Duarte

Examinador:

Prof. Aloysio de Castro Pinto Pedroza

Examinador:

Prof. Mauros Campello Queiroz

DEL

Março de 1996

Dedico este trabalho a todos os meus familiares e amigos.

# Agradecimentos

Aos meus pais, Yêda e Paulo, irmãs e demais familiares.

Ao meu orientador, Professor Otto Carlos Muniz Bandeira Duarte.

Aos meus amigos: Alexandre, Jaca, Vaz, Ana Paula; Naldinho, Milton, Belem, Hermes, Sinval, Gerlan, Lúcia, Siome, Juju; Rubi, Bill, Delavega, Sérgio, Michel, Ronaldo, Spind, Ildemar, Marlos, Pedro, Tony, Evandro, Jonny, Nacao, Marcos, Patrícia, Célio, Faerman, Gilberto, Iniciações (Renata, Luish, Daniel, Sinohara, Ricardo, Gardel, Franchico e irmão); Funcionários da COPPE-Elétrica (Wilson, Fidel, Wanderlei, Daisy, Bia, Vilma); Professores Aloysio, Gelson, Mauros, Moreirão; Galera de Letras (Simone, Andréa, Carmem, Vanessa, Valéria);

E também ...

# Resumo

Este trabalho apresenta um estudo e utilização de um ambiente de programação destinado à implementação de aplicações. São detalhadas duas ferramentas: o Tcl e o Tk, que formam, juntas, uma linguagem para desenvolvimento de aplicações com interface gráfica com o usuário em ambientes *X windows*. Para se demonstrar as potencialidades destas, uma aplicação de teleconferência que se utiliza de um protocolo multiponto desenvolvido no GTA foi implementada. Todos os passos necessários são descritos, inclusive um estudo sobre as principais características do protocolo. Para finalizar, são apresentadas conclusões e sugestões a respeito das vantagens e desvantagens da utilização do Tcl/Tk na implementação de aplicações.

# Índice

# Lista de Figuras

# Lista de Tabelas

# Capítulo 1

# Introdução

Nos últimos anos, a velocidade de transmissão de dados nos meios de comunicação tem crescido substancialmente. Com o aprimoramento da tecnologia, o fator limitante deixou de ser o meio de transmissão e passou a ser o processamento da informação.

Uma grande parcela dos estudos tem se concentrado na pesquisa e desenvolvimento de protocolos de comunicação de alto desempenho, visando-se cada vez mais o ganho de rendimento.

Paralelamente, a exigência por serviços que requerem grande banda-passante também cresceu. Aplicações de transferência de grandes quantidades de dados e de dados em tempo real, como áudio e vídeo são exemplos clássicos.

O Grupo de Teleinformática e Automação (GTA) da COPPE/UFRJ tem, como uma de suas principais vertentes, um grande número de estudos envolvendo protocolos de comunicação de alto desempenho.

O presente trabalho consiste em um estudo aprofundado de uma forma de se criar aplicações que se sirvam dos protocolos desenvolvidos. Um levantamento detalhado de duas poderosas ferramentas para implementação de aplicações foi a motivação principal para o desenvolvimento deste projeto. Essas ferramentas são o Tcl e o

Tk, que, juntos, formam uma linguagem para desenvolvimento de aplicações em ambientes *X windows*.

Para que as potencialidades destas ferramentas fossem demonstradas, foi desenvolvida uma aplicação de conferência para funcionar junto a um protocolo multiponto-a-multiponto implementado no GTA. Este protocolo mereceu um estudo das suas principais características, já que compreender os serviços oferecidos pelo protocolo é uma das fases mais importantes para a implementação de uma aplicação.

O capítulo 2 é destinado ao estudo de um protocolo multiponto-a-multiponto desenvolvido no GTA e denominado MCF (protocolo **M**ultidestinatário **C**onfigurável com gerenciamento de acesso por **F**ichas de permissão). A intenção é se implementar uma aplicação em Tcl/Tk para funcionar junto a este protocolo.

O capítulo 3 trata do Tcl/Tk, desde suas funcionalidades básicas até procedimentos mais avançados, como por exemplo interfaceá-lo com a linguagem C. São apresentadas as vantagens de se utilizar o Tcl/Tk, bem como suas limitações.

No capítulo 4 é todo destinado à aplicação escolhida, uma conferência multiponto que envolve troca de texto e imagens. São detalhados os passos necessários para tal, desde o início da comunicação até a fase de término. Algumas limitações do Tcl/Tk observadas quando da implementação da aplicação são mostradas.

E, por fim, no capítulo 5 se encontram as conclusões a respeito do trabalho, quando se deve ou não utilizar o ambiente de programação estudado, e propostas para trabalhos futuros.

# Capítulo 2

# O Protocolo MCF

O protocolo MCF, desenvolvido pelo GTA, consiste em um protocolo multidestinatário estatisticamente confiável, configurável e com gerenciamento de acesso por fichas de permissão [8].

Ele se utiliza de um mecanismo de fichas para permitir que vários transmissores coexistam dentro de um mesmo grupo, característica dos protocolos multiponto-a-multiponto. São utilizadas **N** fichas para um total de **K** participantes da comunicação **(N<K)**. A diferença **(K-N)** é o número de "apenas ouvintes".

Resumidamente, serão apresentados os mecanismos de comunicação relacionados aos serviços de conexão, reconhecimento, controle de fluxo e controle de erros. Esta é uma etapa primordial para a definição de um novo protocolo.

## 2.1 Mecanismos de Comunicação

### 2.1.1 Gerenciamento de Conexões

Para o protocolo MCF foram estudados dois tipos de mecanismos de gerenciamento de conexões: por *handshake* e por *conexões implícitas* (baseadas em temporizadores). O mecanismo adotado foi o de *handshake* para estabelecimento e liberação de conexões ponto-a-ponto. A escolha deste mecanismo, no lugar de conexões implícitas, se deve ao fato destas últimas requererem atualizações constantes dos campos de tempo de vida das mensagens, além de introduzirem o inconveniente de se calcular os períodos de tempo para acionamento e estouro de temporizadores, tarefa esta não fácil, visto que são cálculos dependentes das condições de tráfego da rede. A vantagem da escolha do mecanismo de *handshake* é que ele possibilita a negociação de parâmetros da conexão (e.g., controle de erros e de fluxo), o que vem atender exatamente a que se propõe o protocolo MCF.

Para as conexões multidestinatárias, optou-se por uma espécie de estabelecimento implícito, porém sem temporizadores. A verdade é que, neste protocolo, não existe propriamente conexões para comunicações multidestinatárias. Uma estação que deseje participar de um grupo não precisa anunciar a nenhuma outra (nem mesmo ao servidor de fichas) esta decisão, bastando conhecer o endereço do grupo (que deve ser de domínio de todos) e configurar a si própria com os parâmetros que lhe interessam. Desde então se torna uma estação "ouvinte" e, quando desejar transmitir dados deve contatar o servidor de fichas (e somente ele), fazendo o pedido de uma ficha.

### 2.1.2 Reconhecimentos

Para o envio de reconhecimentos por parte do receptor, optou-se pela geração submetida à comandos explícitos do transmissor, ou seja, é necessário que este envie um pedido de reconhecimento diretamente ao receptor. Isso reduz a carga de processamento do receptor (que já é naturalmente maior que a do transmissor), mas ao mesmo tempo a aumenta porque é preciso se determinar os momentos de geração dos reconhecimentos. Contudo, se a escolha fosse por reconhecimentos controlados por temporizadores, o protocolo cresceria em complexidade.

### 2.1.3 Controle de Fluxo

São oferecidos dois mecanismos para controle de fluxo. O primeiro, por janelas de transmissão. O segundo, por controle de taxas de transmissão.

A escolha de ambos se deve ao fato deles dependerem muito do tipo de aplicação escolhida. Para transferência de voz, por exemplo, existe a necessidade de um controle rígido sobre a taxa de envio de pacotes, sendo portanto utilizado o mecanismo de controle de taxas de transmissão. Já para transferência de quadros de imagens (não vídeo), um controle por janelas de transmissão seria melhor aplicado. Portanto, fica a cargo da aplicação a escolha do mecanismo.

### 2.1.4 Controle de Erros

Foram estudados e implementados dois tipos de forma de recuperação de erros. A primeira é a recuperação por *go-back-N*. Esta não introduz grande processamento, mas força retransmissões de dados corretamente recebidos. A segunda forma é por retransmissão seletiva, que tem a vantagem da não necessidade de retransmissão de dados corretos, mas requer armazenamento destes em *buffers* do receptor e de resseqüenciamento das mensagens, aumentando a carga de processamento na recepção. A figura 2.1 ilustra o funcionamendo dos dois mecanismos, quando ocorre um erro no terceiro pacote enviado pelo transmissor.

## 2.2 PDUs

O protocolo MCF define seis tipos de PDUs. Seus campos são alinhados em 4 octetos, para aproveitar o potencial da máquinas de 32 *bits*, que é o caso das SPARC-Sun utilizadas no desenvolvimento do presente trabalho.

Todas as PDUs seguem um mesmo formato, apresentado na figura 2.2. O cabeçalho é comum a todos os seis tipos. Ele é utilizado para armazenar diver-

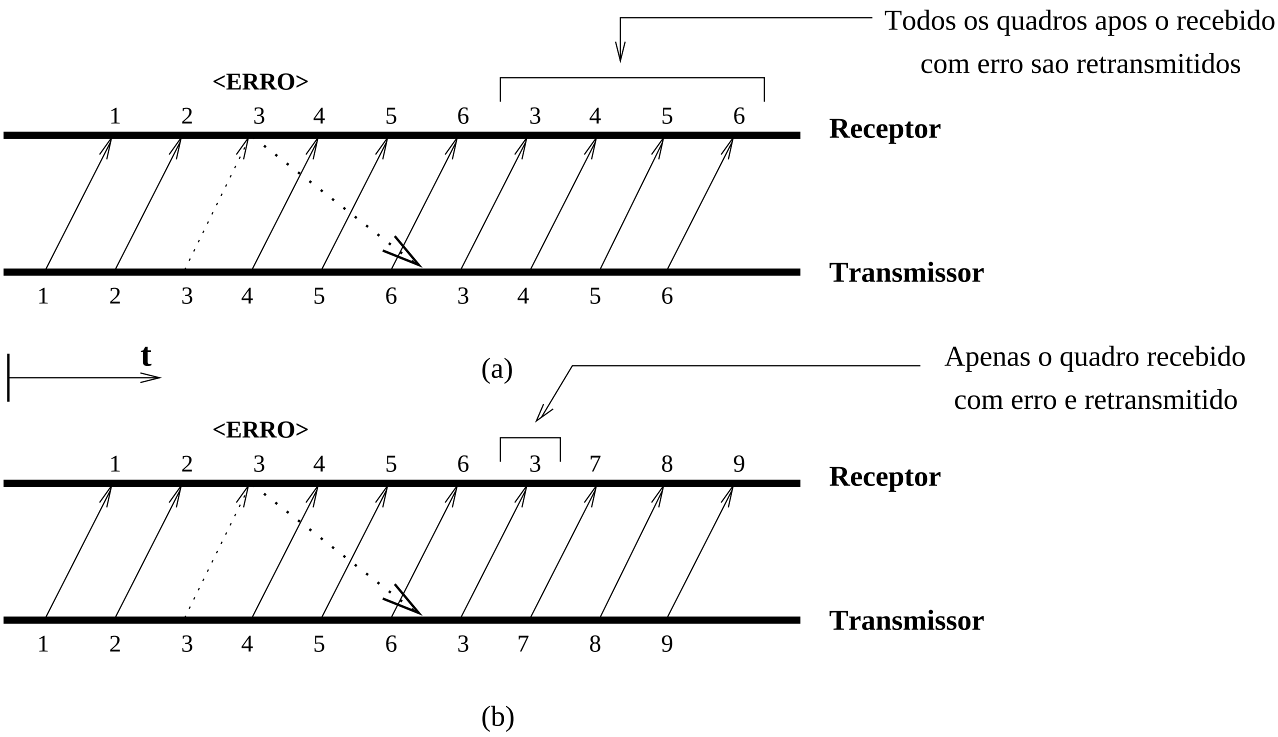


Figura 2.1: Recuperação por *go-back-N* e por retransmissão seletiva

sas informações, como:

- Tipo da PDU (tabela 2.2)
- Opções de configuração do protocolo (tipos de mecanismo de recuperação de erros, controle de fluxo, se a conexão é multidestinatária ou não, etc.)
- Associar uma PDU a uma conexão
- Número de seqüência (em PDUs DATA)
- Número de octetos do segmento de dados
- *Checksum* sobre cabeçalho

A cauda tem apenas um campo, que armazena a função de *checksum* aplicada aos octetos do segmento de dados.

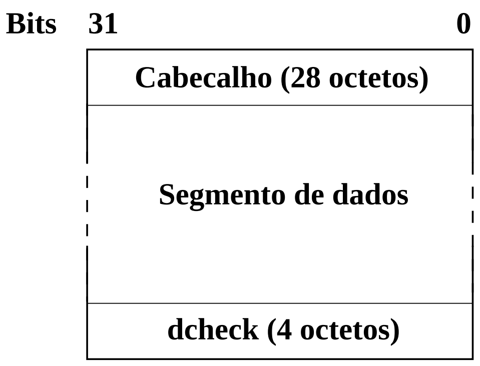


Figura 2.2: Formato de uma PDU

Tabela 2.1: Descrição das PDUs

| *Tipos de PDU* | *Descrição* |
|---|---|
| 0x00 DATA | PDU de dados do usuário |
| 0x01 CNTL | PDU de informações de controle |
| 0x02 FIRST | PDU de estabelecimento de conexões ponto-a-ponto |
| 0x04 TOKEN | PDU representando a ficha em conexões multidestinatárias |
| 0x06 PASS | PDU de pedido de ficha |
| 0x08 ERROR | PDU de indicação de erro de protocolo |

### 2.2.1 Descrição das PDUs

Abaixo segue uma descrição das PDUs utilizadas pelo protocolo.

| | |
|---|---|
| PDU FIRST | Utilizada para estabelecimento de conexões ponto-a-ponto |
| PDU DATA | Utilizada para transporte de dados do usuário |
| PDU CNTL | Usada para troca de informações de controle entre entidades pares, como número de seqüência, reconhecimentos, parâmetros de taxa de transmissão, entre outras |
| PDU TOKEN | Representa a ficha de permissão de transmissão para conexões multidestinatárias |
| PDU PASS | Utilizada pelos participantes de um grupo para solicitarem a ficha |
| PDU ERROR | Usada para fornecer diagnósticos para situações de erro do protocolo |

## 2.3 Primitivas de Serviço

Nesta seção serão descritas, através da tabela 2.3, todas as primitivas utilizadas pelo protocolo MCF. Como apresenta características de um protocolo de transporte, a nomenclatura adotada segue à das primitivas de serviço da camada Transporte do modelo OSI da ISO.

Tabela 2.2: Primitivas de Serviço

| **Primitivas** | **Descrição** | **Parâmetros** |
|---|---|---|
| T-CONNECT.Request | Pedido de estabelecimento de conexão ponto-a-ponto | Endereço destino<br>Endereço origem<br>Qualidade de serviço |
| T-CONNECT.Indication | Indicação de pedido de estabelecimento de conexão ponto-a-ponto | Endereço destino<br>Endereço origem<br>Qualidade de serviço |
| T-CONNECT.Response | Resposta ao pedido de estabelecimento de conexão ponto-a-ponto | Endereço respondedor<br>Resposta (aceitação ou recusa)<br>Qualidade de serviço |
| T-CONNECT.Confirm | Confirmação de pedido de estabelecimento de conexão ponto-a-ponto | Endereço respondedor<br>Resposta<br>Qualidade de serviço |
| T-DISCONNECT.Request | Pedido de liberação de conexão ponto-a-ponto | Tipo de desconexão (normal ou aborto) |
| T-DISCONNECT.Indication | Indicação de pedido de liberação de conexão ponto-a-ponto | Razão da desconexão |
| T-DISCONNECT.Response | Resposta ao pedido de liberação de conexão ponto-a-ponto | Não tem |
| T-DISCONNECT.Confirm | Confirmação do pedido de liberação de conexão ponto-a-ponto | Não tem |
| T-DATA.Request | Pedido de transmissão de dados do usuário | Dados do usuário |

| Primitivas | Descrição | Parâmetros |
|---|---|---|
| T-DATA.Indication | Indicação de recepção dos dados do usuário | Dados do usuário |
| T-MULTIPOINT.Request | Adesão a um grupo multiponto | Categoria (servidor ou não)<br>Qualidade de serviço |
| T-TOKEN.Request | Pedido de ficha para transmissão em grupos multiponto | Endereço destino<br>Endereço origem |
| T-TOKEN.Indication | Indicação de recepção ou devolução de ficha | Não tem |
| T-TOKEN.Response | Comando de devolução da ficha | Não tem |
| T-LEAVE.Request | Desligamento de um grupo multiponto | Não tem |

## 2.4 A Arquitetura de Implementação

O GTA vem há alguns anos desenvolvendo e aprimorando um sistema de alto desempenho para redes de computadores. A figura 2.3 mostra o esquema geral da arquitetura GTA baseada no Modelo de Referência OSI. Além das sete camadas definidas pelo modelo, são utilizados três módulos auxiliares que permitem um ganho significativo de rendimento: o Gerenciador de Memória, o Escalonador de Tarefas e o Gerenciador de Temporizações. Informações detalhadas sobre a arquitetura e os módulos podem ser encontradas em [3][6].

A introdução do protocolo MCF na arquitetura definida pelo GTA pode ser visualizada na figura 2.4. O módulo Globais é contituído pelos módulos Gerenciador de Memória, Escalonador de Tarefas e Gerenciador de Temporizações.

O programa principal é somente uma chamada a todos os módulos do sistema. Um pseudo-código é mostrado a seguir.

```
se Inicializar_Interface_MCF_MAC()
   se Inicializar_Mem_Tx()
        se Inicializar_Mem_Rx()
               se Inicializar_Filas_Internas()
                     Inicializar_Gerente_RMOSI();
                     Inicializar_Modulo_Temporizador();
                     Inicializar_Interface_USU_MCF();
                     Estacao_Local recebe Inicializar_Camada_MAC();
                     Inicializar_MCF();
                     Inicializar_Usuario();
                     repetir
                          Fim recebe Usuario();
                          enquanto Fila_Gerente_RMOSI_Nao_Vazia() fazer
                               Executar_Rotina_Fila_Gerente_RMOSI();
                          fim-enquanto
                     ate Fim = TRUE
                     Finalizar_Usuario();
                     Finalizar_Camada_MAC();
                     Finalizar_Interface_USU_MCF();
                     Finalizar_Gerente_RMOSI();
                     Finalizar_MCF();
                     Finalizar_Modulo_Temporizador();
                     Finalizar_Mem_Rx();
                     Finalizar_Mem_Tx();
                     Finalizar_Interface_MCF_MAC();
               senao
                     Finalizar_Mem_Rx();
                     Finalizar_Mem_Tx();
```

```
                    Finalizar_Interface_MCF_MAC();
                fim-se
                Finalizar_Mem_Tx();
                Finalizar_Interface_MCF_MAC();
            fim-se
    senao
            Finalizar_Interface_MCF_MAC();
    fim-se
fim-se
```

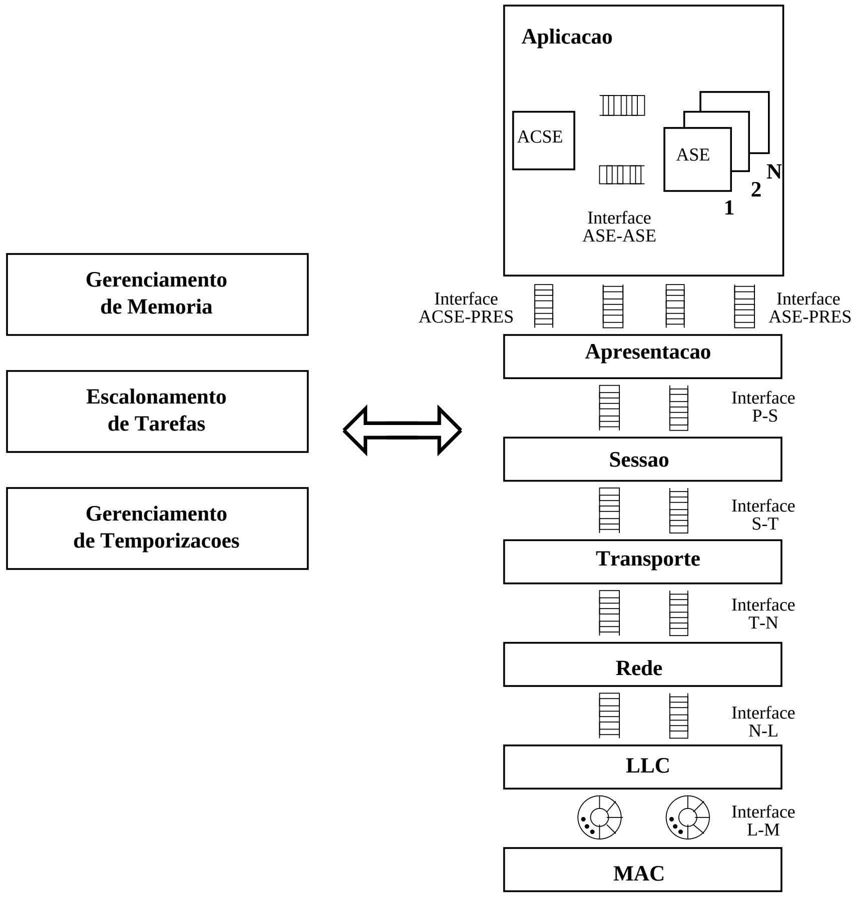


Figura 2.3: Arquitetura GTA

A rotina pode ser subdividida em três partes: **inicialização**, **execução** e **finalização**. A etapa de inicialização consiste no teste de memória necessário para alocações que porventura venham a acontecer, inicialização de filas internas à camada MCF, além de outras chamadas às funções de inicialização do sistema. A fase de finalização libera toda a memória alocada na primeira etapa. A fase de execução é tão somente um laço de repetição, que termina quando a função `Usuário()` retorna Verdadeiro. A função `Usuário()` é utilizada para acesso às funções da aplicação.

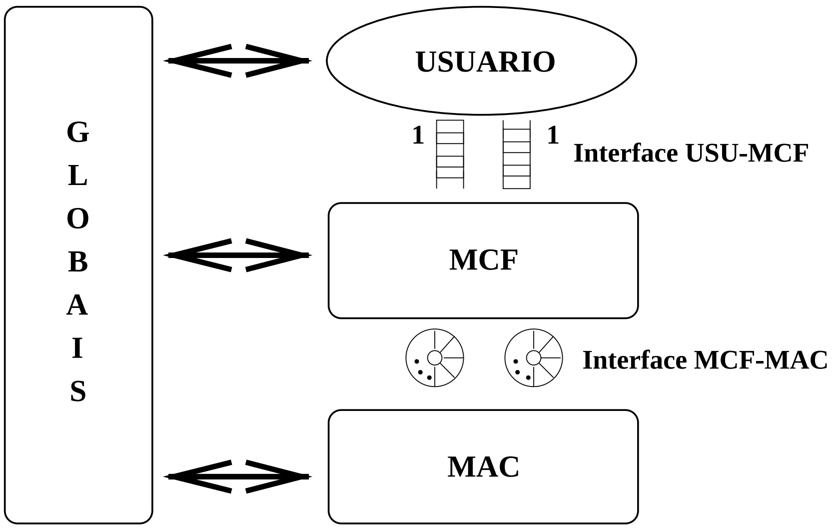


Figura 2.4: O protocolo MCF na arquitetura GTA

# Capítulo 3

# O Kit de Desenvolvimento Tcl/Tk

No início dos anos 80, uma equipe da Universidade de Berkeley desenvolvia ferramentas para o projeto integrados. Na maioria das vezes os pesquisadores implementavam códigos de programação pouco eficientes, e os trechos de código que poderiam aparentemente ser reutilizados não eram reaproveitados.

Desta situação surgiu uma linguagem de programação, de propósito geral, como uma biblioteca em C, e extensível, para que os programadores acrescentassem seus próprios códigos ao núcleo quando necessário. Esta linguagem se chama Tcl, de *Tool Command Language* [9][12].

O Tcl é, ao mesmo tempo, uma linguagem de comandos e um interpretador para os comandos. Como extensão para o Tcl, foi criado um conjunto de ferramentas denominado Tk ("*ToolKit*"), com a finalidade de se criar e manipular interfaces gráficas com o usuário em ambientes *X Windows*.

Sendo uma linguagem de cenários (do inglês *scripting language*), o Tcl é similar a outras linguagens *shell* do UNIX, como o Bourne Shell, C Shell, Korn Shell e Perl. Os programas *shell* permitem a execução de outros programas, além de fornecerem recursos para programação (variáveis, controle de fluxo, procedimentos) que permitem a criação de cenários (*scripts*) que montam programas existentes em uma nova ferramenta que satisfaça a determinadas necessidades.

A facilidade de integrar um interpretador Tcl a uma dada aplicação é o que o difere dos demais programas *shell*. O Tcl preenche o papel de uma linguagem de extensão usada para configurar aplicações. O programador estrutura a aplicação como uma sequência de operações primitivas montadas em um cenário de uma forma a melhor atender às necessidades do usuário. Além, um dado programa pode ser controlado por outro, levando à interação entre diferentes programas. Existem outras linguagens de extensão, tais como o Scheme, Elisp e Python. A escolha do Tcl pode ser baseada no fato deste seguir algumas características de programação da linguagem C e permitir que novas primitivas Tcl sejam escritas como procedimentos C e incorporadas ao núcleo.

A biblioteca Tcl, como característica das próprias bibliotecas, são rotinas compiladas e, no caso, implementa o interpretador de comandos e uma série de comandos do núcleo (manipulação de variáveis, controle de fluxo, acesso a arquivos, procedimentos).

Fora à biblioteca, o programador pode definir novos comandos, como mostrado no esquemático da figura 3.1. Estes comandos são associados a procedimentos em C providos pela aplicação. O resultado é que a aplicação fica composta por uma série de primitivas escritas em uma linguagem compilada e exportadas como comandos Tcl.

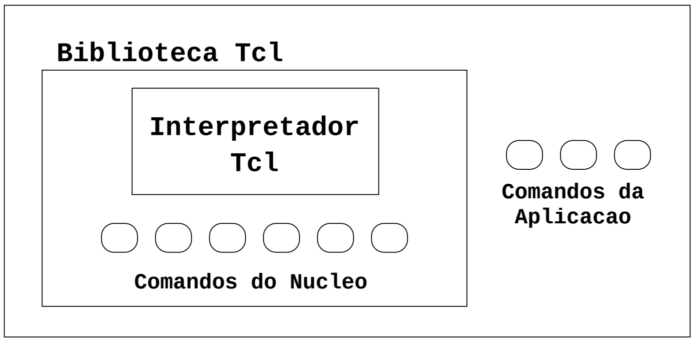


Figura 3.1: Biblioteca Tcl mais alguns comandos específicos da aplicação

Muitas extensões para o Tcl foram desenvolvidas. Há um interesse particular deste trabalho no Tk, a mais difundida. O Tk é um conjunto de ferramentas para ambientes *X windows* que define comandos que permitem a criação e manipulação de controles e interfaces com o usuário. As principais vantagens são a rapidez e facilidade na criação das interfaces.

## 3.1 Aspectos Gerais

### 3.1.1 Passando Comandos ao Interpretador Tcl

Para que comandos Tcl sejam passados ao interpretador, é utilizado um aplicativo, um *shell* denominado `tclsh`. Assim que o `tclsh` é invocado, surge um *prompt* onde comandos são esperados. Por exemplo, se for dado como entrada o comando `expr 2 + 2`, aparecerá o resultado `4` e um novo comando será esperado.

```
expr 2 + 2
⇒ 4
```

O exemplo acima resume algumas das principais características do Tcl. Os comandos são formados por palavras separadas por espaços. Neste exemplo existem quatro palavras: `expr`, `2`, `+` e `2`. A primeira é o nome do comando, que seleciona um procedimento em C que implementará a funcionalidade especificada. As demais são os argumentos passados ao procedimento. Estes argumentos são concatenados em uma única *string*, que no caso acima, será calculada como uma expressão aritmética. O resultado é retornado também como uma *string*. É importante ressaltar que os argumentos dos comandos são sempre passados como caracteres, mesmo quando se deseja passar valores numéricos. O interpretador se encarrega de fazer a conversão.

### 3.1.2 O Tk

O Tk é uma extensão do Tcl que fornece ao programador funcionalidades para se criar interfaces gráficas. O Tk tem seus próprios comandos, além daqueles definidos no núcleo do Tcl, como mostra a figura 3.2.

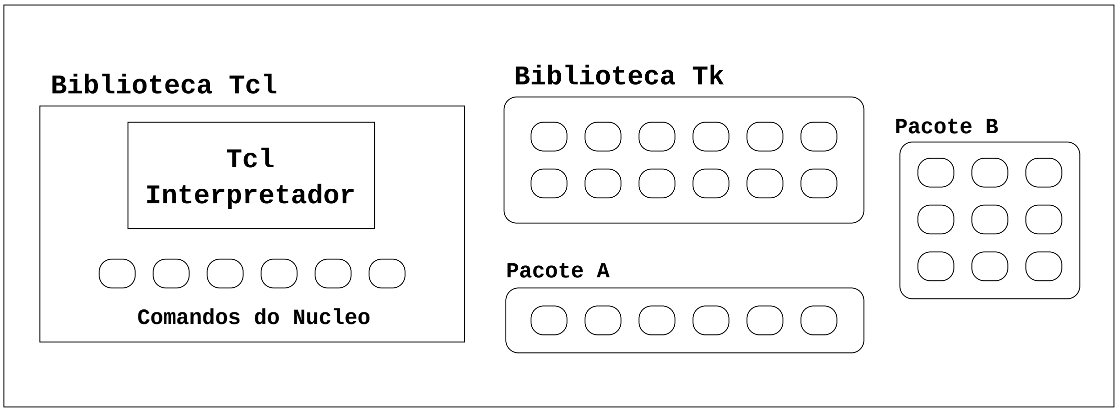


Figura 3.2: Uma aplicação Tcl pode incluir diferentes pacotes como biblioteca

O *shell* do Tk, semelhante ao `tclsh`, é chamado `wish`, e um exemplo de sua utilização é apresentado a seguir, um botão com a inscrição "*Alo, mundo!*". A figura 3.3 ilustra este exemplo.

```
button .b -text ``Alo, mundo!''  -command exit
pack .b
```

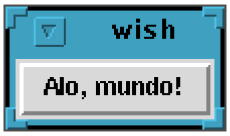


Figura 3.3: Uma *widget* do tipo botão

O primeiro comando, `button`, cria um botão com a escritura *"Alo, mundo!"* que, ao ser "pressionado", executa o comando de saída da aplicação, definido em `-command exit`.

O `pack` é outro comando, que faz com que o botão seja mostrado na tela. A criação de uma *widget*[1] (no caso, o botão) não implica na sua exibição automática.

[1]As *widgets* serão explicadas adiante, na seção reservada ao Tk

### 3.1.3 Os Arquivos *Script*

No lugar de se enviar comandos interativamente no *shell* (`wish` ou `tclsh`), é permitida a criação de um arquivo contendo a sequência de instruções desejadas. São os chamados arquivos de cenário. O exemplo anterior ficaria com a seguinte forma:

```
#!/usr/local/bin/wish -f
button .b -text ''Hello, world!''  -command exit
pack .b
```

O cenário pode ser executado diretamente do *prompt* do UNIX, bastando que seja dada permissão de execução para o arquivo. É necessário também que se coloque na primeira linha do arquivo o caminho para o executável `wish`, como feito acima.

### 3.1.4 Variáveis e Substituições

O Tcl permite que se armazene valores em variáveis para posterior utilização em comandos, como os mostrados a seguir, que exemplificam o armazenamento e substituição de variáveis.

```
set a 5
⇒ 5
expr $a * 2
⇒ 10
```

O símbolo `$` exige do Tcl uma *substituição de variável*. O interpretador, ao notificar a presença do `$`, substitui o caracter `a` por seu conteúdo, ou seja, `5`.

A *substituição de comandos* se utiliza de colchetes. O interpretador primeiramente avalia a expressão interior aos colchetes e, posteriormente, o comando externo.

No exemplo a seguir, é calculada primeiramente a expressão `$a * 2`. O resultado é atribuído à variável `b`.

```
set a 5
⇒ 5
set b [expr $a * 2]
⇒ 10
```

Existe um terceiro tipo de substituição: a de *barra invertida*. Ela é utilizada onde há necessidade de se usar caracteres especiais. Por exemplo:

```
set a GTA
⇒ GTA
set b \$a
⇒ $a
```

Se não tivesse sido feita uma substituição de barra invertida, o resultado seria diferente.

```
set a GTA
⇒ GTA
set b $a
⇒ GTA
```

### 3.1.5 Um Exemplo Maior

A seguir é apresentado um exemplo mais completo, que mostra algumas substituições juntamente com algumas estruturas de controle.

```
proc power {base p} {
    set result 1
    while {$p > 0} {
        set result [expr $result * $base]
        set p [expr $p - 1]
    }
    return $result
}
```

O cenário acima eleva um número `base` à potência `p`. `Proc` é o comando de criação de procedimentos, similar aos procedimentos do Pascal ou C. Se este cenário for executado e o procedimento receber como argumentos para `base` e `p` os valores 2 e 3, tem-se como resultado 8.

```
power 2 3
⇒ 8
```

## 3.2 A Linguagem Tcl

Não é objetivo deste documento mostrar todos os comandos Tcl. Por isso, será apenas apresentado o que a linguagem oferece como características peculiares, principalmente no que se refere às vantagens de utilizá-la na criação de aplicações. O principal ponto de aprendizado se concentra na sintaxe, que consiste em apenas algumas poucas regras.

### 3.2.1 Variáveis

As variáveis utilizadas pelo Tcl podem ser do tipo *simples* (vale lembrar que são sempre vistas como cadeia de caracteres) e do tipo *array associativo* (onde os nomes dos elementos podem ser palavras arbitrárias), ao contrário dos *arrays* de

Tabela 3.1: Exemplos com variáveis dos tipos *simples* e *array associativo*

| *simples* | *array associativo* |
|---|---|
| `set a 5`<br>⇒ `5` | `set NumDias(Jan) 31`<br>⇒ `31` |
| `set a {GTA COPPE}`<br>⇒ `GTA COPPE` | `set NumDias(Fev) {28 29}`<br>⇒ `28 29` |

Tabela 3.2: Expressões em Tcl

| *expressão* | *significado* |
|---|---|
| -a | negativo de a |
| !a | not lógico |
| a%b | resto da divisão de a por b |
| a==b | 1 se a é igual a b; 0, se não |
| a?b:c | se a for não-zero então b, senão c |

outras linguagens, onde os nomes dos elementos obedecem a uma determinada regra (geralmente números inteiros). Um ponto crucial na manipulação de variáveis é com relação à substituição, que gera uma certa dificuldade para os programadores iniciantes. Mas, se for corretamente compreendida, um grande passo terá sido dado na direção do domínio da linguagem.

A tabela 3.2.1 mostra exemplos utilizando os dois tipos de variáveis.

### 3.2.2 Expressões

Expressões combinam valores (ou operandos) com operadores para produzirem resultados. As expressões obedecem à sintaxe da linguagem C. Os operadores são basicamente os mesmos, como se pode ver na tabela 3.2.2, que mostra alguns deles.

### 3.2.3 Listas

Listas são usadas para manipulação de grupos de dados. As listas têm muitas aplicações na linguagem Tcl. Exemplos de listas são um grupo de pessoas, os arquivos de um diretório ou mesmos as opções de uma *widget*. Uma lista é um conjunto de elementos, sendo cada elemento uma palavra qualquer. O comando abaixo atribui à variável `x` uma lista de nomes.

```
set x {alpha beta gamma delta buzios}
⇒ alpha beta gamma delta buzios
```

É também possível o aninhamento de listas.

```
set a {1 2 {3 4 5} 6}
⇒ 1 2 {3 4 5} 6
```

Existem vários comandos para manipulação de listas, cada um com suas finalidades específicas (concatenação, número de elementos da lista, criação de "sub-listas" de acordo com certos parâmetros, etc).

### 3.2.4 Controle de Fluxo

Os comandos de controle de fluxo são muito similares aos da linguagem C. São eles: `if`, `while`, `for`, `foreach`, `switch` e `eval`.

O comando `eval` recebe um bloco de palavras e o executa como um comando.

```
set cmd ''set a 0''
⇒ set a 0

eval $cmd
⇒ 0
```

O comando `foreach` é um laço de repetição, onde a seqüência é repetida de acordo com os elementos de uma lista.

```
set a {a b c d}
set b 0
foreach i $a {
   if {$i == c} break
   incr b 2
}

set b
⇒ 4
```

Os demais comandos (`if`, `while` e `for`) funcionam como nas demais linguagens, tais como C ou Pascal.

### 3.2.5 Procedimentos

Os motivos principais da utilização de procedimentos são agrupar funcionalidades mais usadas e modularizar a programação. Um exemplo de procedimento (comando `proc`) já foi visto no ítem 2.4 deste texto. Outro é apresentado abaixo, que calcula o fatorial de um número.

```
proc fatorial x {
     if {$x <= 1} {
              return 1
     }
     expr $x * [fac[expr $x-1]]
}

fatorial 4
⇒ 24
```

Tabela 3.3: Alguns comandos de manipulação de palavras

| |
|---|
| `format formatstring ?value value ...?`<br>Retorna uma *string* igual a `formatstring`, exceto em `%formato`, que é substituído por `value`.<br>ex: `set msg [format ''%s vale %d'' $letra $numero]` |
| `string compare string1 string2`<br>Retorna -1, 0 ou 1 se `string1` é lexicograficamente menor, igual ou maior que `string2`. |
| `string range string first last`<br>Retorna uma substring de `string`, começando em `first` e terminando em `last`, inclusive. |

### 3.2.6 Manipulação de Palavras

A manipulação de palavras é um dos pontos fortes da linguagem Tcl. E a explicação se baseia no fato de o Tcl ser uma linguagem por si só toda baseada em palavras. A tabela 3.2.6 mostra alguns dos comandos.

Esse é um ponto forte para o desenvolvimento de aplicações. Se os mesmos comandos fossem implementados em C, muitas linhas de código seriam necessárias.

Como exemplo de manipulação de palavras tem-se o código abaixo, que cria uma lista de nome `novo` com todos os termos de `lista` que tenham as três primeiras letras `Tcl`.

```
set novo {}
foreach el $lista {
   if [string match Tcl* $el] {
        lappend novo $el
   }
}
```

### 3.2.7 Arquivos

O Tcl oferece muitas facilidades para acesso a arquivos: leitura e escrita, recuperação de informações do sistema de arquivos, como tamanho e data de último acesso, além de outras. E além, a manipulação dos nomes dos arquivos é feita de uma forma muito simples. Os exemplos seguintes mostram alguns comandos de manipulação de arquivos.

```
file extension src/main.c
⇒ .c

file rootname src/main.c
⇒ src/main

file dirname /a/b/c
⇒ /a/b

glob *.c *.h
⇒ main.c hash.c hash.h

glob {{src,backup}/*.[ch]}
⇒ src/main.c src/hash.c src/hash.h backup/main.c
```

Além dos comandos para manipulação de nomes de arquivos e diretórios, existem aqueles para abertura e fechamento de arquivos e para leitura e escrita.

### 3.2.8 Processos

O Tcl também tem muitas funcionalidades relativas a processos. Os ítens abaixo enumeram o que se pode fazer com relação a eles.

- Criação de novos processos com o comando `exec`
- Criação de novos processos com o comando `open` e comunicação com estes através de comandos de I/O

- Acesso a identificadores de processos com o comando `pid`
- Leitura e escrita de variáveis de ambiente com o comando `env`
- Terminação do processo corrente com o comando `exit`

Além do que foi visto até agora, é possível também tratar dos erros e excessões, além do estado interno do interpretador Tcl.

Estes dois últimos ítens apresentaram comandos muito úteis para implementação de aplicações, que fazem com que o Tcl seja uma recomendado para tal.

A próxima seção terá como assunto o Tk – como criar interfaces gráficas em ambientes *X windows*.

## 3.3 O Toolkit Tk

O Tk é uma extensão da linguagem Tcl. Permite que o programador desenvolva interfaces gráficas ao usuário através de programas de cenário. Para tal, oferece comandos adicionais para criação e manipulação de alguns elementos de controle, os chamados *widgets*. Assim como o Tcl, o Tk pode ser incluído em programas C como biblioteca.

### 3.3.1 Widgets

O Tk utiliza o ambiente X para implementar uma série de controles que seguem o estilo do Motif[2]. Esses controles são chamados *widgets*. Botões (*Buttons*), entradas (*entries*), menus e barras de rolamento (*scrollbars*) são exemplos de *widgets* e serão explicados em seções seguintes.

[2]Motif é um padrão definido pela Open Software Foundation que especifica algumas características de ambiente em X Windows

Cada *widget* faz parte de uma classe, cada uma com características próprias que determinam seu comportamento e aparência. Por exemplo, uma *widget* da classe dos botões pode exibir um texto ou um *bitmap*. As classes têm opções de configuração específicas (cor de fundo, se é possível a exibição de uma imagem, profundidade da tela, etc).

A figura 3.4 mostra, com um exemplo, a hierarquia seguida pelas *widgets*.

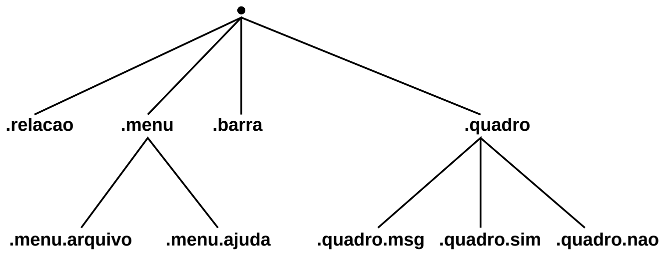


Figura 3.4: Hierarquia seguida pelas *widgets*

### 3.3.2 Classes de *Widgets*

#### Quadros (*frames*)

São as mais simples *widgets*. São retângulos coloridos e podem ter ou não visual tridimensional. São usados basicamente para agrupar outras *widgets*. O cenário a seguir mostra como os quadros são criados. Ele é acompanhado da figura 3.5, que exibe cinco quadros com diferentes opções de configuração de relevo.

```
foreach  relevo {raised sunken flat groove ridge} {
         frame .$relevo -width 15m -height 10m -relief $relevo  \
         -borderwidth 4
         pack .$relevo -side left -padx 2m -pady 2m
}
.flat configure -background black
```

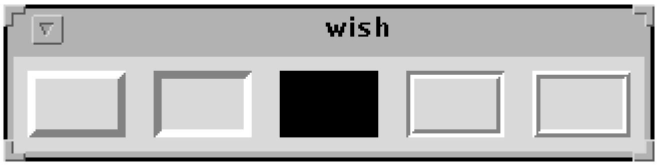


Figura 3.5: Uma seqüência de cinco quadros com diferentes opções de *relief*

**Quadros externos (*toplevels*)**

São como os quadros, ocupando janelas externas, ao contrário dos outros, que ocupam janelas internas. Têm utilidade em caixas de diálogos e outras aplicações particulares.

**Rótulos (*labels*)**

Também são como os quadros, só que são capazes de exibir em seus interiores um texto ou um *bitmap*. A figura 3.6 mostra dois rótulos, um com um *bitmap* e outro com um texto (os dois rótulos estão dentro da mesma janela).

```
label .bitmap -bitmap @$tk_library/demos/images/flagdown
label .rotulo -text "Sem mensagens"
pack .bitmap .rotulo -side top
```


Figura 3.6: Dois rótulos: um exibindo bitmap e outro um texto

### Botões (*Buttons*), botões de checagem (*checkbuttons*) e botões de opção (*radiobuttons*)

Os três têm características semelhantes. Os botões, quando pressionados, executam algum comando definido. Os botões de checagem são usados para escolhas binárias, como setar ou resetar alguma opção. Os botões de opção são como os botões de checagem, setando uma opção dentre várias, mantendo as demais resetadas. Botões de checagem e de opção não invocam comandos.

A figura 3.7 ilustra os três tipos de botão.

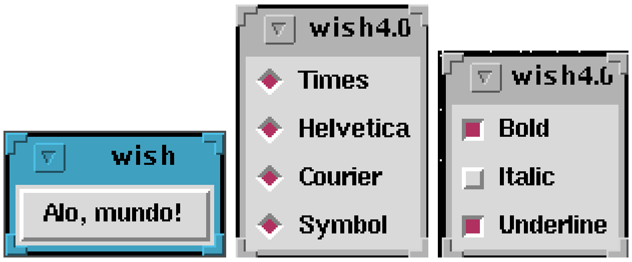


Figura 3.7: Botão, botão de checagem e botão de opção

### Mensagens (*messages*)

São semelhantes aos rótulos, porém permitem múltiplas linhas de texto. As mensagens maiores que a dimensão horizontal definida são automaticamente particionadas em diferentes linhas, como se pode observar na figura 3.8.

```
message .msg -width 8c -justify left -relief raised -bd 2 \
        -font -Adobe-Helvetica-Medium-R-Normal--*-180-* \
        -text "Mensagens maiores que o tamanho maximo definido \
        com a opcao -width sao automaticamente quebradas em  \
        mais de uma linha.  Entendeu?"
pack .msg
```

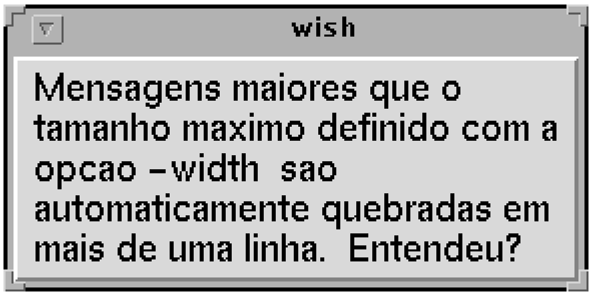


Figura 3.8: Uma widget do tipo mensagem

**Listas (*listboxes*)**

Uma lista é uma *widget* que exibe uma coleção de palavras, permitindo que o usuário selecione uma ou mais delas. O exemplo da figura 3.9 exibe uma lista contendo os nomes das cores existentes no sistema.

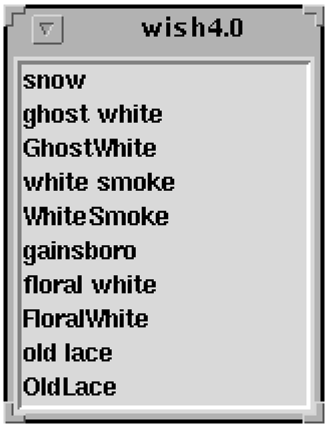


Figura 3.9: Uma widget do tipo lista exibindo as cores do sistema

**Barras de rolamento (*scrollbars*)**

São controles para visualização de outras *widgets*, quando estas ocupam um espaço de tela maior que o definido. São associadas a outras *widgets*. O exemplo da figura 3.10 exibe uma lista associada a uma barra de rolamento.


Figura 3.10: Uma widget do tipo lista com barra de rolamento

**Escalas (*scales*)**

São um tipo de *widget* e exibem uma escala de valores inteiros, permitindo que o usuário os altere apenas movendo uma barra. A figura 3.11 ilustra três escalas.

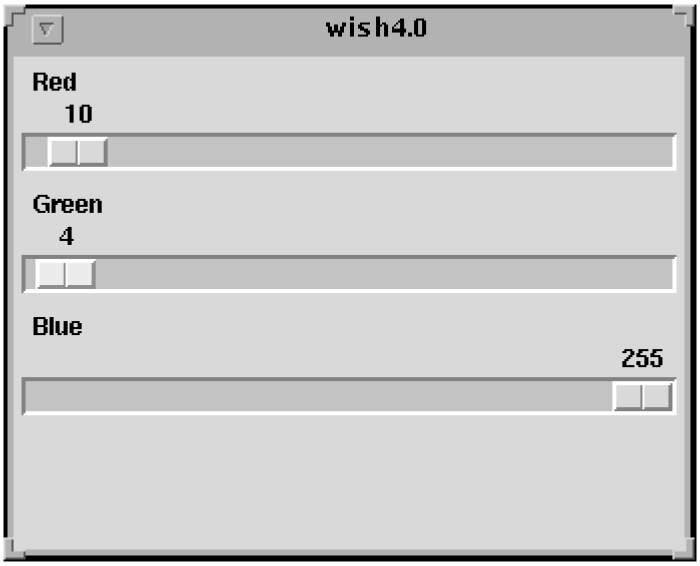


Figura 3.11: Três widgets do tipo escala

**Entradas (*entries*)**

São entradas para o usuário digitar variáveis que serão usadas pela aplicação. Permitem uma única linha de texto. Uma entrada está ilustrada na figura 3.12.


Figura 3.12: Uma widget do tipo entrada

### Menus e botões de menu (*menubuttons*)

As *widgets menus* são blocos usados para implementar uma série de tipos de menu, como menus *pull-down*, em *cascata* e *pop-up*. Os botões de menu são usados para criar várias opções em uma barra de menu. A figura 3.13 exibe uma janela com uma barra de nenu.


Figura 3.13: Widget do tipo menu

### Canvas e textos (*texts*)

*Canvas* são *widgets* que criam uma área de desenho onde se pode colocar vários tipos de figuras, como retângulos, círculos, elipses, linhas, polígonos, arcos, etc.

*Widgets* do tipo texto definem uma área de trabalho onde são exibidos textos de uma ou mais linhas, e permitem que estes textos possam ser editados. Permitem também a escolha do tipo de letra, tamanho, cor, além de outros parâmetros. A figuras 3.14 e 3.15 mostram duas *widgets*, das classes *canvas* e texto.


Figura 3.14: Widget do tipo canvas

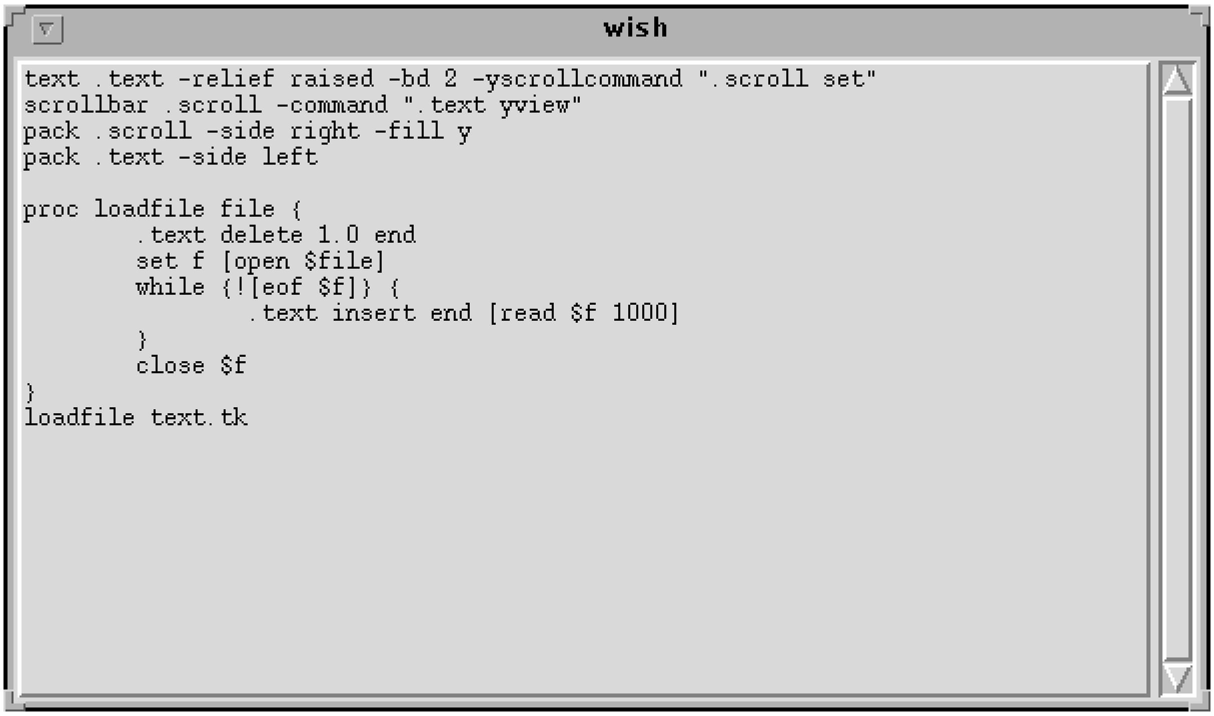


Figura 3.15: Widget do tipo texto

## 3.4 Escrevendo Aplicações Tcl em C

Existem algumas aplicações que não são propícias a serem escritas como um cenário Tcl. Primeiro, acessos a facilidades de baixo nível, como por exemplo soquetes de rede, não são possíveis através do Tcl. Segundo, se o programador estiver preocupado em eficiência e a aplicação requerê-la, como em complexos cálculos numéricos que envolvem grande quantidade de dados, é melhor que o código seja escrito em C. Por último existe o aspecto da complexidade. Se o programador desejar manipular estruturas de dados complexas, onde grande quantidade de código deve ser escrito, melhores resultados serão obtidos se for usada uma linguagem de mais baixo nível. Um cenário Tcl é muito pouco estruturado, o que se torna um ponto crucial quando estão sendo programadas muitas linhas de código.

Uma aplicação Tcl consiste em duas partes: a biblioteca Tcl, mais o código específico da aplicação, como mostra a figura 3.16. A biblioteca é formada pelo avaliador e pelos procedimentos para execução dos comandos do núcleo (como `set` e `if`) e uma série de utilitários para outros fins, como manipulação de palavras e avaliação de cenários e expressões. O código específico da aplicação pode ser dividido em três partes. A maior delas consiste nos procedimentos para implementação de novos comandos Tcl. A segunda parte é reservada para inicializações, como o registro de novos comandos Tcl para a aplicação. Por fim, tem-se o *loop* de comandos, que recolhe os cenários e os transfere ao avaliador.

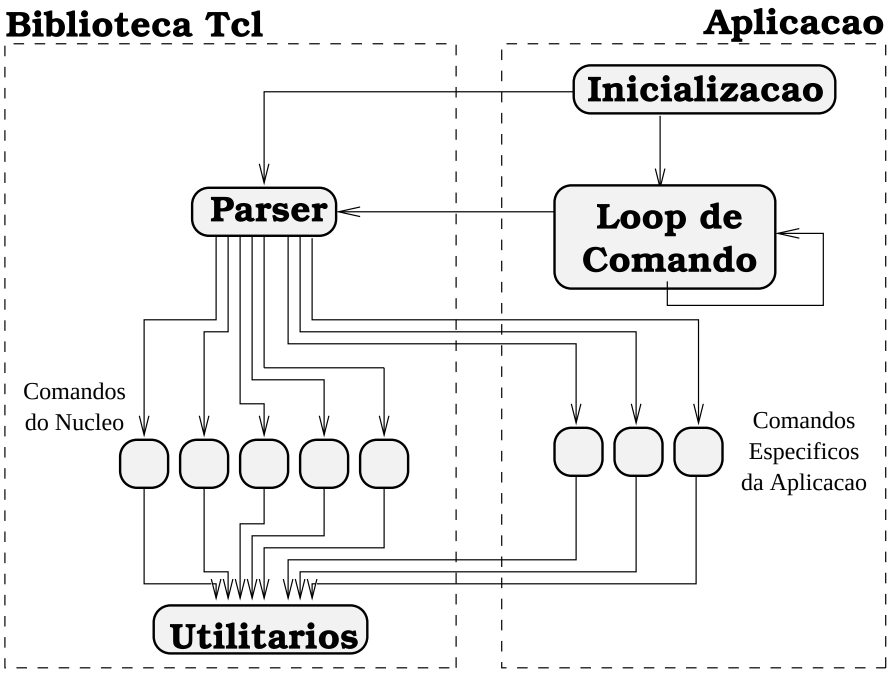


Figura 3.16: Esquemático de uma aplicação Tcl com a biblioteca mais o código específico da aplicação

### 3.4.1 Interpretadores e Avaliação de Cenários

A estrutura central de dados manipulada pela biblioteca Tcl é uma estrutura C do tipo Tcl_Interp. Essas estruturas são chamadas **interpretadores**. Os interpretadores armazenam o estado de execução de um cenário, incluindo os comandos implementados em C, os procedimentos Tcl, variáveis e uma "pilha de execução" que reflete os comandos e procedimentos avaliados. Quase todos os procedimentos da biblioteca Tcl têm como argumento um ponteiro para uma estrutura do tipo Tcl_Interp.

O programa a seguir ilustra a utilização de um interpretador. É usado também um procedimento, o Tcl_EvalFile, que recebe como argumento o nome do arquivo cenário que deve ser executado.

```
/* Programa para Avaliacao de Cenarios */
#include <stdio.h>
#include <stdlib.h>
#include </usr/local/include/tcl.h>
void main(argc, argv)
int argc;
char *argv[];
{
     Tcl_Interp *interp;
     int codigo;

     if (argc!=2)
     {
         fprintf (stderr, "Numero errado de argumentos");
         exit(1);
     }
     interp = Tcl_CreateInterp();
     codigo = Tcl_EvalFile(interp, argv[1]);
```

```
    if (*interp->result !=0)
    {
        printf("%s\n", interp->result);
    }
    if (codigo!=TCL_OK)
    {
        exit(1);
    }
    exit(0);
}
```

Mas, para que um interpretador seja usado, deve-se primeiramente criá-lo. Isto é feito com a função `Tcl_CreateInterp()`, como no exemplo acima.

Além do `Tcl_EvalFile`, existem outras formas de se executar um cenário. A função `Tcl_Eval` recebe como argumento um cenário, no lugar de receber o nome do arquivo que o contém.

```
char comando[] = ''set a 10''
    ...
Tcl_Eval(interp, comando);
```

### 3.4.2 Criando-se Novos Comandos Tcl

Todos os comandos Tcl são implementados como um procedimento em linguagem C. Quando um comando é invocado, o interpretador executa o código em C correspondente. Esses procedimentos são chamados **procedimentos de comando**.

Muitas vezes é necessário que, dentro de um cenário, sejam chamados trechos de código específicos da aplicação (figura 3.16). Para que isso seja possível, deve-se criar um novo comando Tcl associado ao código C correspondente.

A interface para um procedimento de comando é definida pelo protótipo `Tcl_CmdProc`.

```
typedef int Tcl_CmdProc (ClintData clientData, Tcl_Interp *interp,
                          int argc, char **argv);
```

Um procedimento de comando é invocado quando o comando Tcl correspondente é executado, recebendo quatro argumentos. O primeiro, `clientData`, é usado em casos especiais, recebendo geralmente o valor NULL. O argumento `interp` é utilizado para indicar em qual interpretador o comado será utilizado. Os dois últimos argumentos, `argc` e `argv`, têm a mesma funcionalidade quando usados na função `main()` da linguagem C.

Um procedimento de comando retorna dois valores. O primeiro, inteiro, é resultado do sucesso ou fracasso da execução do procedimento, retornando valores como `TCL_OK` ou `TCL_ERROR`. O segundo é atribuído à variável `interp->result`, e pode ser uma mensagem de erro ou uma mensagem com resultados.

A seguir será implementado como exemplo um procedimento de comando que compara dois argumentos.

```
int EqCmd (ClientData clientData, Tcl_Interp *interp, int argc, char **argv);
{
    if (argc != 3)
    {
          interp->result = "Erro no numero de argumentos";
          return TCL_ERROR;
    }
    if (strcmp (argv[1], argv[2]) == 0)
    {
          interp->result = "1";
    }
    else
```

```
    {
          interp->result = "0";
    }
    return TCL_OK;
}
```

Mas, para que o procedimento de comando `EqCmd` possa ser invocado pelo Tcl, é necessário que seja registrado. Para tal é utilizada a função `Tcl_CreateCommand`, como mostra o programa exemplo abaixo, que registra o procedimento `EqCmd` definido anteriormente.

```
/* Programa que define um novo comando Tcl */
#include <stdio.h>
#include </usr/local/include/tcl.h>
void main(argc, argv)
int argc;
char *argv[];
{
     Tcl_Interp *interp;
     int codigo;

     if (argc!=2)
     {
         fprintf (stderr, "Numero errado de argumentos");
         exit(1);
     }
     interp = Tcl_CreateInterp();
     Tcl_CreateCommand (interp, "eq", EqCmd, (ClientData) NULL,
         (Tcl_CmdDeleteProc *) NULL);
     codigo = Tcl_EvalFile(interp, argv[1]);
```

```
    if (*interp->result !=0)
    {
        printf("%s\n", interp->result);
    }
    if (codigo!=TCL_OK)
    {
        exit(1);
    }
    exit(0);
}
```

Os argumentos para a função `Tcl_CreateCommand` são cinco. O primeiro, `interp`, especifica o interpretador no qual o comando será usado. O segundo é o nome do comando que ficará disponível para ser usado em um cenário. O terceiro argumento é o nome do procedimento que será invocado quando da execução do comando `eq`. Os demais argumentos têm utilidade em casos especiais, e podem ser definidos como NULL em comandos simples, como o em questão.

Com o que foi especificado até o presente ponto, já é possível interfacear programas em C com *scripts* Tcl[3]. O capítulo 4, destinado à aplicação, extenderá os conceitos com exemplos práticos.

[3] e também Tk, como será visto no próximo capítulo

# Capítulo 4

# A Aplicação

O Grupo de Teleinformática e Automação vem há algum tempo desenvolvendo protocolos de comunicação de alto desempenho. Recentemente, decidiu-se aplicar um maior esforço em aplicações que se sirvam dos protocolos desenvolvidos pelo GTA.

Este trabalho tem o objetivo de implementar uma aplicação com interface gráfica com o usuário e que se serve de um protocolo de transporte. Paralelamente, estão em andamento outros projetos na área de multimídia que envolvem aplicações de transferência de áudio e vídeo.

Outro trabalho de fim de curso desenvolvido no GTA incumbiu-se de implementar as funções multiponto de acesso à rede através do *IP Multiponto* [5]. Tanto estas funções quanto a aplicação desenvolvida serão acopladas ao protocolo MCF.

## 4.1 Aspectos Gerais da Aplicação

Para se servir das potencialidades do protocolo MCF, foi desenvolvida uma aplicação de teleconferência que, além da troca de texto entre os participantes, há o intercâmbio de fotografias. Quando um novo membro toma parte no grupo, difunde sua foto. Os demais retornam as suas e, em cada estação, é montada uma janela do novo participante, dando-se continuidade à comunicação. A figura 4.1 esboça a janela da aplicação.

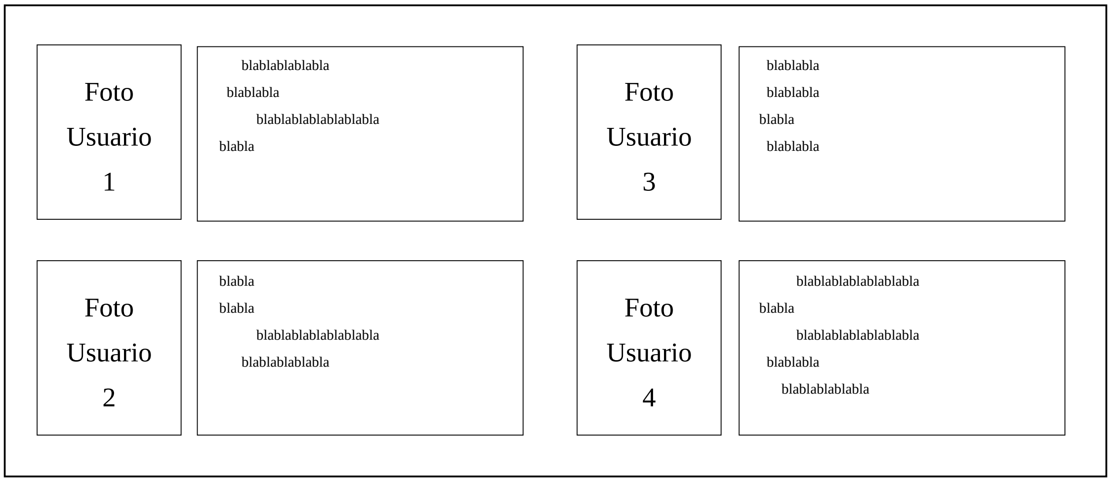


Figura 4.1: Esboço da aplicação de conferência

Todos os passos necessários ao desenvolvimento da aplicação serão detalhados.

## 4.2 A Aplicação no Protocolo MCF

Algumas características do protocolo devem ser ressaltadas para que se tornem mais claras algumas decisões concernentes à aplicação.

As comunicações realizadas com o protocolo MCF são do tipo **dinâmicas**, ou seja, em qualquer instante estações podem entrar e sair de um grupo sem que esta transação afete de algum modo a comunicação como um todo.

Em cada grupo existe uma máquina centralizadora, conhecida como o servidor de fichas. Uma outra estação que deseje tomar parte deste grupo só precisa se configurar para receber pacotes endereçados a ele. Apenas quando desejar transmitir alguma mensagem é que esta estação deverá contatar o servidor, com a intenção de pedir uma ficha. As demais estações não são perturbadas.

Surgem as perguntas. Se uma estação não sabe quando uma outra toma parte do grupo, como em uma aplicação de conferência elas saberão o momento de enviar as fotografias dos respectivos usuários? Durante uma conferência, se uma estação passa a pertencer a um grupo, como ela deve avisar às demais que passou a existir mais um membro?

As respostas não vêm do protocolo, e sim da própria aplicação. O protocolo MCF repassa para a aplicação todas as tarefas particulares que nela possam existir.

### 4.2.1 A Entrada ou Saída de um Membro de um Grupo

A estação que passa a tomar parte de um grupo, após pedir e receber uma ficha do servidor, envia uma mensagem destinada ao grupo com um código significando que um novo membro entrará na conferência. Esta mensagem é acompanhada da fotografia do novo participante. As demais estações, ao receberem esta mensagem retornam uma outra, também destinada ao grupo, com as respectivas fotos. Deve, neste caso, existir um código de que esta é uma mensagem respondedora, para não haver problema de repetição de membros nas várias estações. Desta forma, a dinâmica da comunicação é mantida, permanecendo esta característica no protocolo.

A máquina de estados da entrada ou saída de um novo membro em um grupo está mostrada na figura 4.2. Na seqüência (figura 4.3) está a máquina de estados das estações que já estão no grupo quando da entrada ou saída de outra.

### 4.2.2 Configuração dos Parâmetros do Protocolo

Os parâmetros do protocolo, tais como estabelecimento de conexão, controle de erro e fluxo e reconhecimentos, devem ser determinados de forma a abstrair o usuário de certas tarefas que não devem ser de sua incumbência.

Para estabelecimento de conexão e geração de reconhecimentos não foi preciso se escolher entre um ou outro mecanismo. O protocolo MCF oferece apenas uma opção para cada caso: estabelecimento implícito para conexões multiponto-a-multiponto e geração de reconhecimentos a partir de comando explícitos do transmissor.

Em relação ao controle de fluxo, a opção foi pelo mecanismo de janelas de

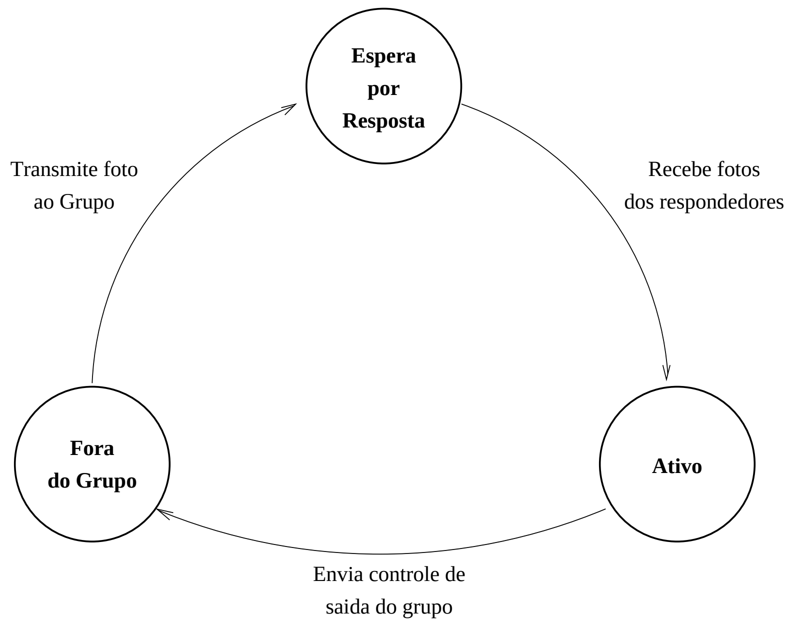


Figura 4.2: Máquina de estados da entrada ou saída de um membro de um grupo

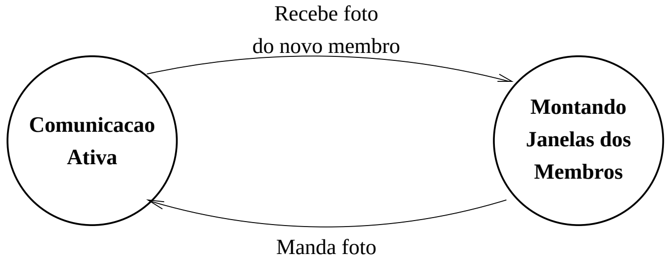


Figura 4.3: Máquina de estados dos membros de um grupo quando da entrada de um novo

transmissão, dado que a aplicação não envolve transmissão de dados em tempo real, tais como áudio e vídeo. A preocupação maior é com a possibilidade de enchimento dos *buffers* de recepção.

O mecanismo escolhido para recuperação de erros foi o de *go-back-N*. A escolha se deu pela maior simplicidade deste quando comparado ao mecanismo de retransmissão seletiva.

## 4.3 Implementação da Aplicação

A aplicação, apenas para reforçar o conceito, consiste numa conferência onde, além da troca de texto, há a troca de fotos entre os participantes. A parte no que se refere à troca de informações com o protocolo é toda feita em linguagem C. Já a parcela de interação com o usuário, toda em forma gráfica, foi desenvolvida em Tcl/Tk.

A interação da aplicação com o protocolo envolve a escolha dos parâmetros deste e a montagem das primitivas de serviço (apresentadas no capítulo 2). As fases de inicialização e finalização do sistema, bem como a implementação do programa principal são tarefas da aplicação.

As próximas seções se destinarão à implementação da interface gráfica com o usuário.

### 4.3.1 A Exibição de Imagens em *Widgets*

Como aplicação envolve imagens, é necessário observar como estas podem ser mostradas na tela. Algumas *widgets*, como botões e rótulos, permitem a exibição de figuras em seus interiores. Entretanto, o padrão para o Tk são as imagens no formato *X Bitmap*, que permitem apenas duas cores. A figura 4.4 mostra algumas imagens neste padrão.

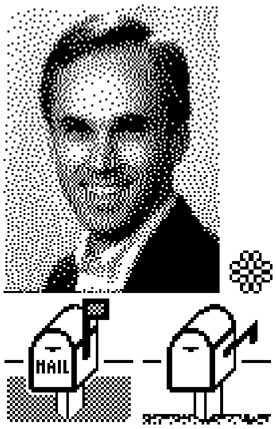


Figura 4.4: Algumas imagens no padrão *X Bitmap*

Este tipo de imagem não é interessante para este trabalho, que envolve exibição de fotografias (coloridas). Portanto, deve-se escolher outro padrão e verificar como podem (e se podem) ser mostrados na tela.

A versão 4.0 do Tk, ao contrário da versão 3.6, permite que algumas *widgets* exibam imagens diferentes de *X Bitmap* em seus interiores, nos padrões GIF ou PPM. A escolha para este trabalho foi pelo formato GIF, que tem a vantagem de possuir compactação, ao contrário do PPM. A exibição de uma imagem em uma *widget* se faz com a opção `-image`.

Para ilustrar a utilização desta opção, é apresentada figura 4.5, que exibe em dois rótulos imagens nos padrões GIF e PPM.

### 4.3.2 A Troca de Textos

As áreas de tela utilizadas para exibição dos textos trocados entre os participantes são *widgets* do tipo texto. Ao lado de cada texto é alocada uma barra de rolamento. A figura 4.6 mostra uma *widget* texto com uma barra de rolamento.

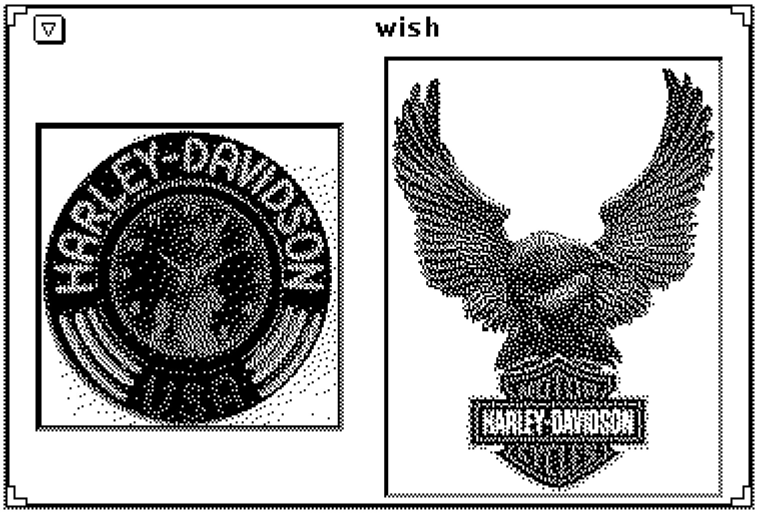


Figura 4.5: Duas *labels* exibindo imagens nos padrões GIF e PPM

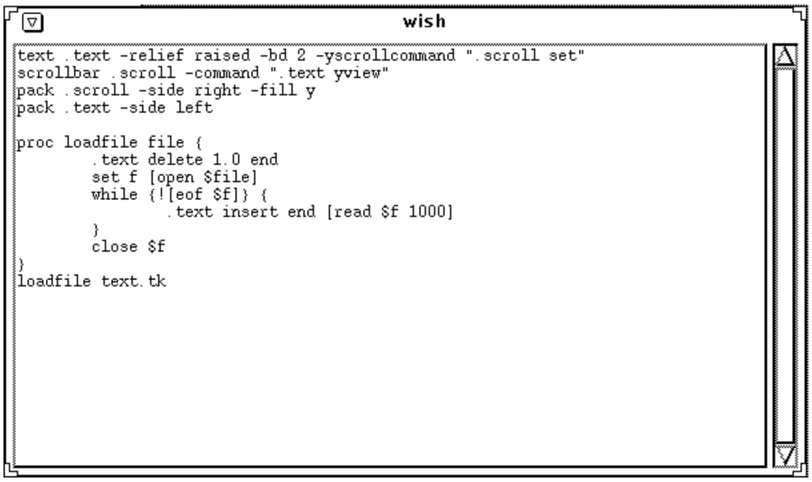


Figura 4.6: *Widget* onde serão exibidos os textos trocados entre os participantes


Figura 4.7: Barra de menu da janela da aplicação

### 4.3.3 A Barra de Menu

A barra de menu da aplicação é formada por cinco botões, como mostra a figura 4.7

O botão **File** oferece as seguintes opções: carregamento de texto (se um usuário desejar transmitir um texto pré-editado), armazenamento em arquivo e saída da aplicação.

Os botões **Connect** e **Disconnect** são usados para aderência e abandono de grupos multiponto, apesar dos nomes escolhidos não refletirem a realidade (não há propriamente conexões no protocolo MCF).

O botão **Preferences** é utilizado pelo usuário para configuração da aplicação, desde as fontes para as letras até as cores do ambiente.

E, por fim, o botão **Help**, que oferece recursos de ajuda para dúvidas que possam surgir.

## 4.4 A Janela TalK

O resultado final da implementação da aplicação é mostrado na figura 4.8. Consiste em uma conferência com quatro participantes.

O fragmento base do cenário correspondente à esta aplicação está listado no apêndice A deste documento. É importante notar a resumida quantidade de código desenvolvido para a implementação de tantas funcionalidades.

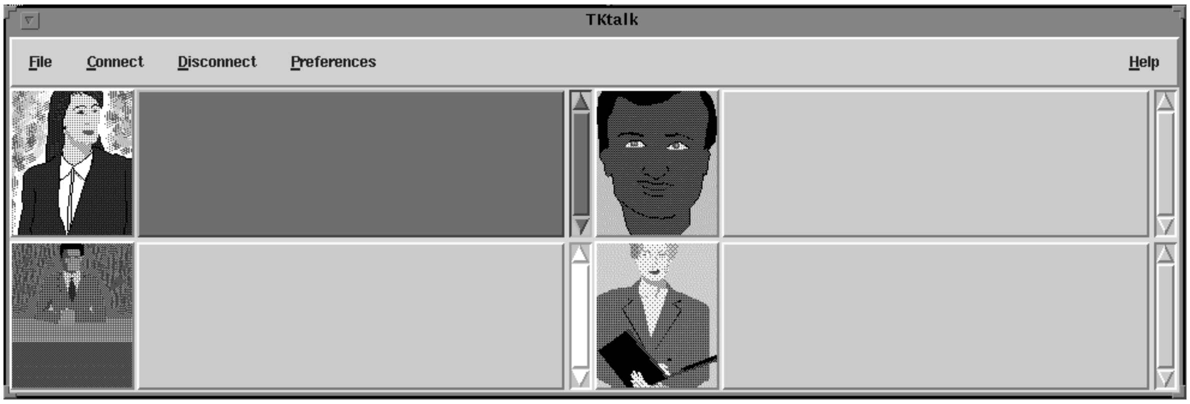


Figura 4.8: Janela Final da aplicação

# Capítulo 5

# Conclusões

Este trabalho tratou do estudo e utilização de ferramentas para desenvolvimento de aplicações.

De acordo com a experiência adquirida neste projeto, para a melhor utilização do Tcl/Tk, alguns procedimentos devem ser atentamente observadas. Por ser uma linguagem interpretada, existe uma deficiência que compromete a velocidade do sistema. Além disso, por ser pouco estruturada, a linguagem não é recomendada para implementação de grande quantidade de código. Outra questão é com relação ao alto nível em que a linguagem opera. Quando são necessários acessos a funções de baixo nível, é imprescindível que se utilize uma linguagem mais apropriada para este fim, como por exemplo o C.

Saber quando usar o Tcl/Tk ou o C é uma etapa importante no desenvolvimento das aplicações. A figura 5.1 mostra graficamente estas idéias.

Mas basta que os limites impostos sejam respeitados para que as vantagens apareçam. O tempo economizado e a pequena quantidade de código necessária para a implementação são ganhos claramente observados. Deve-se ressaltar também que aplicações podem ser desenvolvidas relativamente à parte do restante do sistema, para posterior integração.

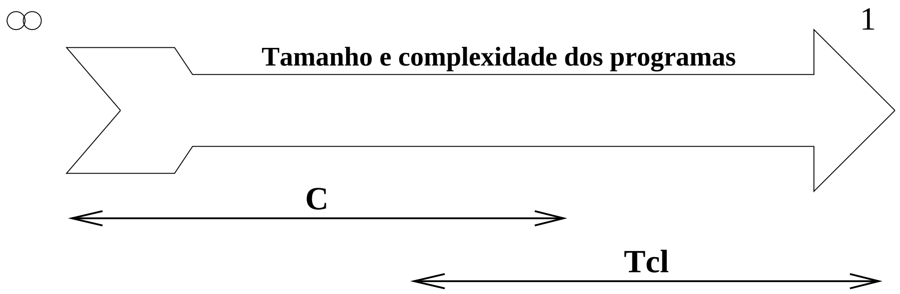


Figura 5.1: Utilização das linguagens C e Tcl/Tk

Para se caracterizar as vantagens do Tcl/Tk foi desenvolvida uma aplicação de teleconferência que se serve de um protocolo de transporte do GTA. Foram definidas todas as características desta aplicação de forma a aproveitar as principais potencialidades do protocolo, devidamente estudados durante a execução do trabalho.

A aplicação oferece vários recursos, como entrada e saída de um membro em um grupo, configuração do ambiente, entre outros. Uma conclusão importante que se pode extrair é que a implementação de uma aplicação sobre um protocolo de alta velocidade é um caso bastante geral e exige um projeto bem definido. Caso contrário, uma falha pode resultar em ineficiência, por motivos já ditos.

Para trabalhos futuros, ficam algumas sugestões:

- Utilizar mais potencialidades do Tcl/Tk, mesmo com as extensões disponíveis via rede
- Estudar outras linguagens para implementação de aplicações e realizar um trabalho comparativo com o Tcl/Tk
- Implementar aplicações que envolvam transferência de áudio e vídeo, em Tcl/Tk, para o protocolo MCF e outros desenvolvidos no GTA

# Apêndice A

# Listagem

```
#!/usr/local/bin/wish -f

#-------------------------------------------------
#
#  TKtalk - Aplicacao para Protocolo MCF
#
# Data Inicio:        23/12/95
#
# Ultima Modificacao: 07/02/96
#
#-------------------------------------------------

eval destroy [winfo child .]

wm title . "TKtalk"
wm iconname . "TKtalk"
wm geometry . +100+200
```

```
set directory ./

set bg_menu        [format #%02x%02x%02x 176 224 230]
set fg_menu        [format #%02x%02x%02x 0 0 205]
set bg_text_mine   [format #%02x%02x%02x 65 105 225]
set fg_text_mine   [format #%02x%02x%02x 255 255 255]
set bg_text_others [format #%02x%02x%02x 173 216 230]
set fg_text_others [format #%02x%02x%02x 0 0 0]
set bg_top         [format #%02x%02x%02x 0 0 128]
set fg_top         [format #%02x%02x%02x 255 255 255]
set bg_top_button  [format #%02x%02x%02x 255 255 0]
set fg_top_button  [format #%02x%02x%02x 0 0 128]

set c1 $bg_menu
set c2 $fg_menu
set c3 $bg_text_mine
set c4 $fg_text_mine
set c5 $bg_text_others
set c6 $fg_text_others
set c7 $bg_top
set c8 $fg_top
set c9 $bg_top_button
set c10 $fg_top_button

set change nothing

set font1 -Adobe-Helvetica-Medium-R-Normal--*-180-*
set font2 -Adobe-Helvetica-Medium-R-Normal--*-150-*
set font3 -Adobe-Helvetica-Medium-R-Normal--*-130-*
set font4 -Adobe-Helvetica-Medium-R-Normal--*-100-*
```

```
source listbox.tcl
source menu.tcl


# --
# Cria uma janela toplevel com a abertura
# da aplicacao.
# --

proc sairDefault {} {
   destroy .default
   wm deiconify .
}

proc botaoDefault {bD} {
   wm iconify .
   catch {destroy .default}
   toplevel .default
   wm title .default "Conference for MCF Protocol"
   wm iconname .default "TK"
   wm geometry .default +300+300
   image create photo imagemDefault -file $bD
   button .default.botaoInicial -image imagemDefault -command {sairDefault}
   pack .default.botaoInicial
   focus .default
}

botaoDefault tktalk.gif
```

```
# --
# Monta a janela da Aplicacao para quatro participantes
# com labels, texts e scrolbars.
# --

frame .menubar -bg $c1 -relief raised -borderwidth 2
frame .total
frame .left
frame .right
frame .left.up
frame .left.down
frame .right.up
frame .right.down
frame .left.up.in
frame .left.down.in
frame .right.up.in
frame .right.down.in

image create photo usu1 -file usu1.gif
image create photo usu2 -file usu2.gif
image create photo usu3 -file usu3.gif
image create photo usu4 -file usu4.gif

label .left.up.usu1 -relief raised -height 120 -width 100 -image usu1
label .left.down.usu2 -relief raised -height 120 -width 100 -image usu2
label .right.up.usu3 -relief raised -height 120 -width 100 -image usu3
label .right.down.usu4 -relief raised -height 120 -width 100 -image usu4

text .left.up.in.usu1 -relief raised -bd 2 -height 9 -width 50 -bg $c3 \
```

```
  -fg $c4 -yscrollcommand {.left.up.in.s set}
text .left.down.in.usu2 -relief raised -bd 2 -height 9 -width 50 -bg $c5 \
  -fg $c6 -state disabled -yscrollcommand {.left.down.in.s set}
text .right.up.in.usu3 -relief raised -bd 2 -height 9 -width 50 -bg $c5 \
  -fg $c6 -state disabled -yscrollcommand {.right.up.in.s set}
text .right.down.in.usu4 -relief raised -bd 2 -height 9 -width 50 -bg $c5 \
  -fg $c6 -state disabled -yscrollcommand {.right.down.in.s set}

scrollbar .left.up.in.s -orient vertical -bg $c3 -command {.left.up.in.usu1 \
  yview}
scrollbar .left.down.in.s -orient vertical -bg $c4 -command \
  {.left.down.in.usu2 yview}
scrollbar .right.up.in.s -orient vertical -bg $c5 -command \
  {.right.up.in.usu3 yview}
scrollbar .right.down.in.s -orient vertical -bg $c5 -command \
  {.right.down.in.usu4 yview}


# --
# Cria os botoes do menubar
# --

menubutton .menubar.file -text File -bg $c1 -fg $c2 -menu .menubar.file.m \
  -relief flat -underline 0
menu .menubar.file.m -bg $c1 -fg $c2
.menubar.file.m add command -label "Load Text" -underline 0 -command {load}
.menubar.file.m add cascade -label "Save Text" -menu .menubar.file.m.save
.menubar.file.m add separator
.menubar.file.m add command -label Quit -underline 0 -command exit
```

```
menu .menubar.file.m.save -bg $c1 -fg $c2
.menubar.file.m.save add checkbutton -label "Usuario 1" -variable salvaUsu1 \
  -command {save Usuario1}
.menubar.file.m.save add checkbutton -label "Usuario 2" -variable salvaUsu2 \
  -command {save Usuario2}
.menubar.file.m.save add checkbutton -label "Usuario 3" -variable salvaUsu3 \
  -command {save Usuario3}
.menubar.file.m.save add checkbutton -label "Usuario 4" -variable salvaUsu4 \
  -command {save Usuario4}

menubutton .menubar.connect -text Connect -bg $c1 -fg $c2 -relief flat -menu \
  .menubar.connect.m -underline 0
menu .menubar.connect.m -bg $c2 -fg $c2
.menubar.connect.m add command -label "Where to..." -underline 0 -command \
  {connect}

menubutton .menubar.disconnect -text Disconnect -bg $c1 -fg $c2 -relief flat \
  -underline 0 -menu .menubar.disconnect.m
menu .menubar.disconnect.m -bg $c1 -fg $c2
.menubar.disconnect.m add command -label "Quit from group" -underline 0 -command \
  {disconnect}

menubutton .menubar.preferences -text Preferences -bg $c1 -fg $c2 -relief flat \
  -underline 0 -menu .menubar.preferences.m
menu .menubar.preferences.m -bg $c1 -fg $c2
.menubar.preferences.m add command -label Colors -underline 0 -command \
  {conf_colors}

menubutton .menubar.help -text Help -bg $c1 -fg $c2 -relief flat -menu \
  .menubar.help.m -underline 0
```

```
menu .menubar.help.m -bg $c1 -fg $c2
.menubar.help.m add command -label "About TKtalk" -command {BaianoAutor}



# --
# Exibe tudo na tela
# --

pack .left.up.in.usu1 -side left
pack .left.up.in.s -side right -fill y

pack .left.down.in.usu2 -side left
pack .left.down.in.s -side right -fill y

pack .right.up.in.usu3 -side left
pack .right.up.in.s -side right -fill y

pack .right.down.in.usu4 -side left
pack .right.down.in.s -side right -fill y

pack .left.up.usu1 .left.up.in -side left

pack .left.down.usu2 .left.down.in -side left

pack .right.up.usu3 .right.up.in -side left

pack .right.down.usu4 .right.down.in -side left

pack .left.up .left.down -side top -in .left
```

```
pack .right.up .right.down -side top -in .right

pack .left .right -side left -in .total

pack .menubar.file .menubar.connect .menubar.disconnect .menubar.preferences \
  -side left -fill both -padx 2m -pady 2m -in .menubar
pack .menubar.help -side right -fill both -padx 2m -pady 2m -in .menubar
pack .total -side bottom
pack .menubar -side top -fill x


# --
# Toplevel com informacoes do autor
# --

proc BaianoAutor {} {
   catch {destroy .aboutAuthor}
   toplevel .aboutAuthor -relief raised
   wm title .aboutAuthor "About Author"
   wm iconname .aboutAuthor "About"
   wm geometry .aboutAuthor +450+300

   message .aboutAuthor.msgAbout -width 8c -justify center -relief raised -bg \
  $c7 -fg $c8 -bd 2 \
      -font -Adobe-Helvetica-Medium-R-Normal--*-180-* \
      -text "The author of this program is Marcelo Dias de Amorim. He can be \
  found via \
     e-mail: amorim@coe.ufrj.br"
   button .aboutAuthor.dismiss -font -Adobe-Helvetica-Medium-R-Normal--*-120-* \
```

```
 -text Dismiss -bg $c7 -fg $c8 -command "destroy .aboutAuthor"
   pack .aboutAuthor.msgAbout -side top
   pack .aboutAuthor.dismiss -side bottom -expand 1 -fill both
}
```

# Bibliografia


[1] Albuquerque, C.V.N., "Arquitetura de Implementação de Alto Desempenho para Sistemas de Comunicação Multimídia", Tese de Mestrado, PEE-COPPE/UFRJ, abril de 1995

[2] Baginski, L.F., Duarte, O.C.M.B., "Um Modelo de Implementação de Alto Desempenho para Sistemas Abertos", IX Congresso da Sociedade Brasileira de Telecomunicações, São Paulo, Brasil, pp. 1941-1945, setembro de 1991

[3] Baginski, L.F., Barros, F.M.C., Schatzmayr, R., Duarte, O.C.M.B., "Implementação e Análise de Desempenho em um Sistema de Transferência de Dados", X Congresso da Sociedade Brasileira de Computação, Vitória, Brasil, pp. 157-169, julho de 1991

[4] Baginski, L.F., "Ambiente de Implementação para Sistemas de Alto Desempenho", Tese de Mestrado, PEE-COPPE/UFRJ, janeiro de 1992

[5] Costa, H.A.A.S., "Implementação de Protocolo Multidestinatário no MBONE", Projeto de Fim de Curso, DEL/UFRJ, março de 1996

[6] Nunes, M.D., Albuquerque, C.V.N., Duarte, O.C.M.B., "Performance Measurements in a Manufacturing Communication System", ISCAS'94, Londres, maio de 1994

[7] Nunes, M.D., Duarte, O.C.M.B., "Análise de Mecanismos para Protocolos de Alto Desempenho", VI Simpósio Brasileiro de Arquitetura de Computadores e Computação de Alto Desempenho, Caxambu, agosto de 1994

[8] Nunes, M.D., "Protocolo Multidestinatário com Configuração de Facilidades e Gerenciamento de Acesso", Tese de Mestrado, PEE-COPPE/UFRJ, abril de 1995

[9] Ousterhout, J.K., "Tcl and Tk Toolkit", Addison-Wesley Publishing Company, 1994

[10] Tarouco, L.M.R., "Redes de Computadores: Locais e de Longa Distância", McGraw-Hill, São Paulo, 1986

[11] Tanembaum, "Computer Networks", Prentice-Hall, 1989

[12] Welch, B., "Practical Programming in Tcl and Tk", Addison-Wesley Publishing Company, abril de 1995