wxWindows: Uma Introdução

Carlos Neves Júnior
cnevesjr@dcc.unicamp.br

Tallys Hoover Yunes
tallys@dcc.unicamp.br

Fábio Nogueira de Lucena
fabio@dcc.unicamp.br

Hans Kurt E. Liesenberg
hans@dcc.unicamp.br

Relatório Técnico DCC–95-19

Dezembro de 1995

wxWindows: Uma Introdução

Carlos Neves Júnior
cnevesjr@dcc.unicamp.br

Tallys Hoover Yunes
tallys@dcc.unicamp.br

Fábio Nogueira de Lucena
fabio@dcc.unicamp.br

Hans Kurt E. Liesenberg
hans@dcc.unicamp.br

Projeto Xchart*
DCC/IMECC/UNICAMP
Caixa Postal 6065
13081-970 Campinas/SP, Brazil
e-mail: *xchart@dcc.unicamp.br*
URL: *http://www.dcc.unicamp.br/projects/Xchart*

1 Introdução

Este tutorial fornece ao usuário, com noções básicas de programação orientada a objetos em C++, uma introdução elucidativa à programação no *wxWindows*. Este toolkit é uma ferramenta de domínio público, com implementações para várias plataformas (`ftp.aiai.ed.ac.uk`, no diretório `pub/packages/wxwin`). Este texto trata exclusivamente da interface de programação do wxWindows e como ela pode ser usada na construção de interfaces. Vários exemplos são fornecidos.

O *wxWindows* implementa recursos comumente encontrados em interfaces homem-computador, tais como: menus, botões, caixas de diálogo, check-boxes etc., tornando mais simples e rápida a implementação destes elementos.

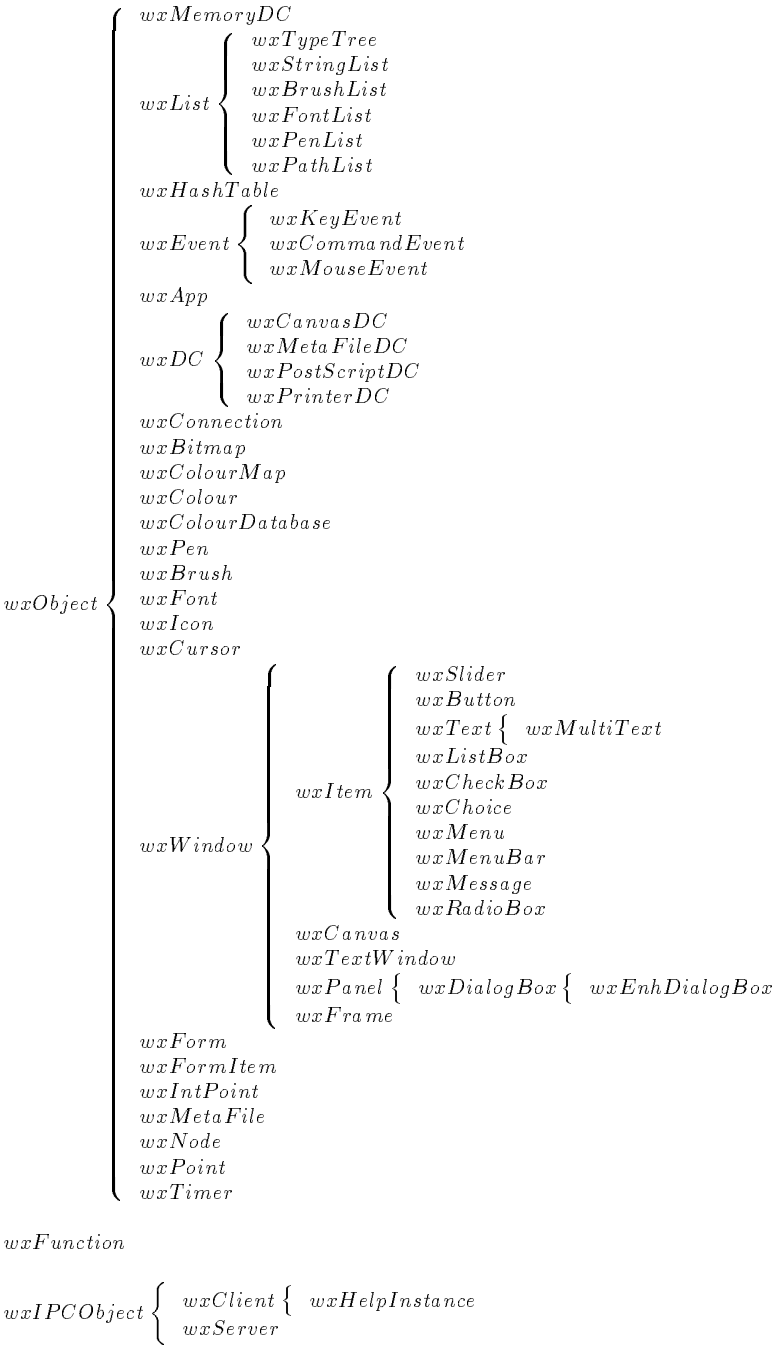
A organização das seções desse tutorial, segue a ordem apresentada abaixo:

- Hierarquia de Classes
- Gerando uma interface
- Widgets disponíveis
- Alguns Exemplos

Este tutorial não tem a pretensão de ser exaustivo. Limitamo-nos aos aspectos essenciais. Maiores detalhes relacionados a alguns recursos específicos e à implementação podem ser encontrados na documentação do *toolkit* e no próprio código fonte.

*Projeto que se concentra no desenvolvimento (especificação e implementação) de interfaces homem-computador. Por curiosidade: a 22ª letra do alfabeto grego ($\mathbf{X}$ e χ) é identificada em Inglês pela palavra CHI, que coincide com o acrônimo de *Computer-Human Interface.*

2 Hierarquia de Classes



3 Gerando uma Interface

Esta seção contém informações a respeito de makefiles, headers e outros arquivos necessários à geração de um programa executável de uma aplicação wxWindows.

A seguir é apresentado um exemplo de makefile utilizado para compilação em ambiente Windows NT com Borland C++ 4.5.

```
01 #!make
02 # Borland C++ 4.x makefile for noname
03 # Generated by wxBuilder.
04
05 CCC=bcc
06 MODEL=l
07 CFG = wxwin.cfg
08 PCH = wxwin.pch
09
10 # Change BCCDIR to wherever your Borland Compiler is found.
11 BCCDIR = C:\BC4
12
13 # Change WXDIR to wherever wxWindows is found.
14 BCCDIR = \\beatles\bc45
15 WXDIR = \\ntmaster\appl\develop\wxwindow
16 WXLIB = $(WXDIR)\lib\$(MODEL)wx.lib
17 WXINC = $(WXDIR)\include
18
19 TOOLBARDIR = $(WXDIR)\utils\toolbar
20 TOOLBARINC = $(TOOLBARDIR)\src
21 TOOLBARLIB = $(TOOLBARDIR)\lib\toolbar.lib
22
23 !ifdef FAFA
24 FAFALIB = $(WXDIR)\contrib\fafa\fafa.lib
25 !endif
26
27 THISDIR = z:\tallys\wx
28
29 OBJECTS = noname.obj
30
31 LIBS=$(WXLIB) $(TOOLBARLIB) $(FAFALIB) mathwl cwl import commdlg ddeml shell $(WXDIR)\contrib\ctl3d\ctl3d
32 !ifndef FINAL
33 FINAL=0
34 !endif
35
36 !if "$(FINAL)" == "0"
37 OPT=-Od
38 DEBUG_FLAGS=-v
39 LINKFLAGS=/v/Vt /Twe /L$(WXDIR)\lib;$(BCCDIR)\lib
40 !else
41 OPT=-O2
42 DEBUG_FLAGS=
43 LINKFLAGS=/Twe /L$(WXDIR)\lib;$(BCCDIR)\lib
44 !endif
45
46 CPPFLAGS=$(DEBUG_FLAGS) $(OPT) @$(CFG)
47
48 .cpp.obj:
49  $(CCC) $(CPPFLAGS) -c {$< }
50
51 all:    $(CFG) noname.exe
52
53 $(CFG): makefile.bcc
```

```
54  copy &&!
55 -H=$(PCH)
56 -3
57 -P
58 -Ff=512
59 -dc
60 -Fc
61 -d
62 -N
63 -w-hid
64 -w-par
65 -w-pia
66 -w-aus
67 -w-rch
68 -m$(MODEL)
69 -WSE
70 -I$(WXINC)\msw;$(WXINC)\base;$(TOOLBARINC);$(BCCDIR)\include
71 -Dwx_msw
72 ! $(CFG)
73
74 wx:
75  cd $(WXDIR)\src\msw
76  make -f makefile.bcc FINAL=$(FINAL) OPT=$(OPT)
77  cd $(THISDIR)
78
79 noname.obj: noname.h noname.cpp
80 noname.res :      noname.rc $(WXDIR)\include\msw\wx.rc
81  rc /i$(BCCDIR)\include /i$(WXDIR)\include\msw /i$(WXDIR)\include\base /i$(WXDIR)\contrib\fafa -r noname
82
83 noname.exe:      $(WXLIB) $(OBJECTS) noname.def noname.res
84  tlink $(LINKFLAGS) @&&!
85 c0wl.obj $(OBJECTS)
86 noname
87 nul
88 $(LIBS)
89 noname.def
90 !
91  rc -K noname.res
92
93 clean:
94  -erase *.obj
95  -erase *.exe
96  -erase *.res
97  -erase $(PCH)
```

O nome do arquivo fonte utilizado na compilação é `noname`. Desse modo, todas as ocorrências do nome `noname` no makefile acima devem ser substituídas pelo nome de seu arquivo fonte.

O diretório raiz do *wxWindows* em nosso ambiente de trabalho é

`\\ntmaster\appl\develop\wxwindow`

devendo-se, portanto, substituir todas as ocorrências desse *path* pelo diretório equivalente do ambiente em que seu programa será compilado. De forma análoga, deve-se alterar também o path do diretório onde se encontra o compilador utilizado (em nosso caso, Borland C++).

Além do makefile, são necessários mais dois arquivos para se gerar um programa executável em ambiente Windows: um `.def` e um `.rc`.

O arquivo `.def` deve conter definições a respeito da apresentação e funcionamento do programa executável, tais como: o nome que ele possuirá dentro de seu grupo de programas e o stub que será utilizado (no caso, "`WINSTUB.EXE`"). Segue abaixo o arquivo `.def` utilizado na geração desse exemplo.

```
NAME         Exemplo1
DESCRIPTION  'Nome do programa dentro do grupo'
EXETYPE      WINDOWS
STUB         'path do winstub\WINSTUB.EXE'
CODE         PRELOAD MOVEABLE DISCARDABLE
DATA         PRELOAD MOVEABLE MULTIPLE
HEAPSIZE     1024
STACKSIZE    8192
```

O arquivo `.rc` (como apresentado abaixo) deve conter informações a respeito dos *resources* que serão utilizados quando da compilação do programa, dentre eles, o ícone associado.

```
mondrian ICON mondrian.ico
#include "wx.rc"
```

Dispondo desses três arquivos, basta acrescentar na área de inclusões de seu arquivo fonte a linha "`#include wx.h`", que deve estar presente em todas as aplicações a serem desenvolvidas em *wxWindows*.

4 Widgets Disponíveis no *wxWindows*

Nesta seção serão comentados, isoladamente, botões, panels, mensagens e outros. Estes elementos são conhecidos por widgets. Widgets, portanto, implementam elementos típicos de uma interface. A lista abaixo não é exaustiva.

Textos típicos ("Yes","No", ...) que são apresentados ao usuário pelos widgets podem ser mostrados em várias línguas (default = inglês).

Um widget normalmente está relacionado com uma função, que deve ser chamada sempre que o usuário manipular este widget. Funções que servem a este propósito são denominados de callback.

4.1 wxBitmap

Um bitmap pode ser criado dinamicamente (fornecendo-se o vetor dos bits correspondente) ou carregado de um arquivo. Após a sua criação, o bitmap pode ser associado a um botão, copiado para um canvas ou para um *memory device context* (veja `wxMemoryDC`) a fim de que possa ser utilizado como superfície de desenho. São suportados tanto bitmaps monocromáticos como coloridos e suas dimensões e quantidade de cores podem ser manipulados.

4.2 wxBrush

Um brush é utilizado no preenchimento de regiões e fundo de figuras geométricas (retângulos, elipses, etc). Trata-se de um padrão de preenchimento para o qual é necessário definir os parâmetros cor e estilo. Vários brushes são predefinidos pelo wxWindows.

4.3 wxBrushList

Um brushlist é uma lista que contém todos os brushes que são criados por uma aplicação e é apontada pela variável `wxTheBrushList`. Pode-se procurar por brushes já existentes ou criá-los caso ainda não existam.

4.4 wxButton

Um button pode conter um texto ou uma figura (bitmap) relacionada. Um button pode tanto ser colocado como item de um panel (`wxPanel`) como de uma dialog box (veja `wxDialogBox`). Pode ser habilitado ou desabilitado.

4.5 wxCanvas

Um canvas é uma subjanela de um frame onde podem ser desenhados gráficos (p. ex.: figuras geométricas, bitmaps) e textos. Eventos gerados pelo mouse ou pelo teclado (ex: ESC) no interior de um canvas, podem ser detectados e ações apropriadas ao seu tratamento podem ser disparadas. Um canvas pode ser monocromático ou colorido e além disso dispor de barras de rolagem.

4.6 wxCanvasDC

Um canvas device context é automaticamente criado quando um canvas é definido. Este device context está relacionado com o canvas sendo possível utilizarmos o código genérico para desenho de figuras e textos em um device context (`wxDC`) a fim de escrevermos na região do canvas.

4.7 wxCheckBox

Um checkbox é uma pequena "caixa" associada a um label ou a um bitmap e que pode estar marcada (com um check mark) ou não. Pode-se verificar se um wxCheckBox está marcado ou não e ainda alterá-lo.

4.8 wxChoice

Um choice é utilizado para a seleção de uma opção dentre as existentes em uma lista. Diferentemente de uma listbox, o wxChoice deixa visível apenas o item selecionado. Um menu com a lista de itens para escolha fica oculto até que o usuário deseje alterar a sua opção. Há métodos para verificar qual item está selecionado, procurar por um determinado item e acrescentar um novo item na lista, por exemplo.

4.9 wxColour

Um colour é um objeto que representa uma cor no formato RGB. Este é utilizado na referência feita à cores para desenho de figuras e escrita de labels. Há um conjunto de cores predefinidas.

4.10 wxColourDatabase

Trata-se de um banco de dados de cores padrão RGB associadas a nomes predefinidos (ex: "BLACK", "LIGHT GREY"). Pode-se procurar uma cor pelo nome ou, inversamente, procurar o nome (cadeia de caracteres) dada a cor (`wxColour`). Cores podem ser adicionadas ao conjunto já existente.

4.11 wxCommandEvent

Trata-se de uma classe de eventos que contém informações sobre os eventos gerados por itens de panels. As funções de callback de itens de panels (`wxFunction`) recebem-na como um de seus parâmetros. Opcionalmente, uma classe desse tipo pode ser construída e utilizada para simular um comando gerado pelo usuário num item de panel. Através de uma instância dessa classe, pode-se verificar o valor de um checkbox, obter a cadeia de caracteres correspondente à seleção de um listbox, e outros.

4.12 wxCursor

Um cursor nada mais é do que um pequeno bitmap usado para identificar a posição do mouse. Um mesmo objeto do tipo cursor pode ser usado em diferentes janelas.

4.13 wxDC

Um device context é uma forma de representar um conjunto de atributos de dispositivos de saída de uma forma genérica, onde gráficos e textos são desenhados (um canvas e uma impressora tem um device context relacionados). Manipulando o wxDC, por exemplo, pode-se produzir figuras geométricas na tela ou na impressora indistintamente.

4.14 wxDialogBox

Um dialogbox é semelhante a um panel no que diz respeito à capacidade de acomodar itens de panels. Seu título e estado (em forma de ícone ou não) podem ser consultados e alterados. Além disso, é possível centralizá-lo na tela e controlar seu aparecimento (visível ou não).

4.15 wxEnhDialogBox

Um Enhdialogbox é um wxDialogBox com recursos adicionais. Separa uma dialogbox em quatro áreas (área de pin, área do usuário, área de comandos, área de status).

4.16 wxEvent

Um event é um tipo de estrutura que armazena informações a respeito de um evento. Trata-se de uma classe abstrata que serve de base para as classes de evento `wxCommandEvent` e `wxMouseEvent`.

4.17 wxFont

Um font é um objeto que representa uma forma de apresentação de um texto. Por exemplo, tipo de letra, estilo e outros.

4.18 wxFontList

Um fontlist é uma lista que contém todos os fonts criados e associados à variável predefinida `wxTheFontList`. É similar a wxBrushList.

4.19 wxForm

Um form é uma alternativa para apresentação de formatos de forma funcional, minimizando o árduo trabalho de definir os itens de um panel detalhadamente. Os itens de um form são definidos conjuntamente com algumas restrições onde, a partir destas, os itens são automaticamente escolhidos sem que o programador precise detalhar maiores informações.

4.20 wxFrame

Um frame é uma janela que pode conter subjanelas de vários tipos (um canvas, um panel ou uma textwindow). A janela tem um título e pode ter um barra de menu e uma linha de status. Um frame pode ter vários de seus atributos alterados, como movimentação, fechamento, mudança no tamanho, presença do menu de controle, e outros.

4.21 wxFunction

wxFunction é o tipo (assinatura) de uma função de callback.

4.22 wxHelpInstance

Esta classe implementa a interface para o sistema de Help. Na instanciação desta classe é feita a sua associação com um nome de arquivo. Através de métodos adequados é possível carregar o arquivo já relacionado inicialmente, procurar por uma palavra chave no arquivo e monitorar a saída do Help, por exemplo.

4.23 wxIcon

Um ícone é um pequeno bitmap de forma retangular que é representado por uma instância de wxIcon.

4.24 wxKeyEvent

Trata-se de uma classe de eventos que contém informações a respeito de eventos gerados pelo teclado. Um evento desse tipo, permite identificar se a tecla shift foi pressionada, qual o código ASCII da tecla pressionada, em que posição houve o pressionamento da tecla etc.

4.25 wxListBox

Um listbox permite selecionar uma ou mais cadeias de caracteres dentre uma lista apresentada por uma caixa com barra de deslocamento. Elementos podem ser acrescentados ou removidos da lista, bem como ter seu estado de seleção consultado ou alterado.

4.26 wxMenu

Um menu pode ser utilizado para se construir uma barra de menus ou um menu do tipo popup. A cada opção de um menu está associado um identificador que pode ser usado para localizá-la ou alterá-la. Além disso, os itens de um menu podem ter associados a si uma cadeia de caracteres que forneça uma breve explicação de sua utilidade, podem ser habilitados ou desabilitados, apresentar ou não um check mark a seu lado e ter seu rótulo alterado pelo programa em tempo de execução. Opcionalmente, pode-se criar submenus (pullright) em um menu e alterar seu título principal.

4.27 wxMenuBar

Um menu bar é uma série de menus acessíveis no topo de um frame. Selecionando uma opção, é aberto um menu do qual é selecionado um item. Um menu bar pode ter algumas de suas opções desabilitadas, pode ter a inclusão de novas opções com um menu associado, e outras facilidades para manipular as suas opções e rótulos.

4.28 wxMessage

Um message é uma linha de texto que pode ser colocada em uma panel, podendo ser alterada em tempo de execução. Só serve como saída, não gera eventos.

4.29 wxMouseEvent

Trata-se de uma classe de eventos que contém informações a respeito de eventos gerados pelo uso do mouse. Permite identificar qual dos botões do mouse foi pressionado, se houve double click na operação e se o mouse está sendo arrastado, por exemplo.

4.30 wxMultiText

Possui função e utilização semelhante ao wxText, só que dispondo de várias linhas de texto. Pode possuir barras de rolagem tanto horizontais como verticais e pode-se obter um ponteiro para um buffer contendo o texto correntemente armazenado.

4.31 wxPanel

Um panel é uma subjanela de uma frame no qual itens (button, choice, listbox, etc) podem ser colocados para permitir interação com o usuário. Existem vários controles que informam a posição do cursor, definem a fonte utilizada nos rótulos e botões, método para quebra de linha fazendo com que o próximo item seja posicionado no ínicio da próxima linha.

4.32 wxPen

Um pen é um objeto que representa uma forma com que as linhas e contornos de figuras são desenhados. Na definição de um pen são necessários três parâmetros: cor, espessura e estilo da linha. Estes parâmetros podem ser consultados e alterados através de seus métodos.

4.33 wxPenList

É similar a wxBrushList.

4.34 wxPostScripDC

Trata-se de um device context (DC) que permite a criação e alteração de arquivos PostScript.

4.35 wxRadioBox

Um radiobox é uma estrutura utilizada para se selecionar, de forma mútua e exclusiva, um item dentre vários itens pertencentes a uma lista que pode ser apresentada vertical ou horizontalmente. Botões do radiobox podem ser habilitados, desabilitados ou ter seu estado de seleção alterado. Além disso, pode-se obter o identificador ou rótulo do elemento correntemente selecionado, o número de escolhas possíveis e permitir ou não a visualização de controles individuais do radiobox.

4.36 wxSlider

Um slider é um dispositivo horizontal que possui um pequeno elemento que pode ser "arrastado" para a direita ou para a esquerda, alterando o seu valor. Seu valor corrente pode ser consultado ou alterado pelo programa em tempo de execução.

4.37 wxText

Um text é uma linha em que um texto pode ser editado. Seu conteúdo corrente pode ser consultado ou alterado pelo programa em tempo de execução.

4.38 wxTextWindow

Um text window é uma subjanela de uma frame que oferece facilidades para apresentar e editar textos. Por exemplo, copy, cut, paste, carga de um arquivo para edição e posterior gravação, destacar trechos do texto, e outros.

4.39 wxWindow

Esta é a classe base para todos os tipos de janelas e itens de panels. Dentre as operações principais que podem ser executadas sobre instâncias dessa classe ou de suas subclasses, podemos destacar: direcionamento de todas as entradas de mouse para essa janela, centralização na tela, habilitação ou desabilitação, consulta e alteração do tamanho da janela,

alteração do seu tipo de cursor e título e alteração do estado da janela para visível ou não. Além disso, é possível interceptar certos eventos executados sobre um descendente dessa classe, tais como: ativação ou desativação da janela, pressionamento de uma tecla, operações de "drop" de arquivos, operações com o mouse (cliques, "drags"), refresh etc.

5 Alguns Exemplos

A seguir são apresentados exemplos que buscam mostrar o emprego dos principais widgets do wxWindows. Por comodidade, os fontes dos exemplos podem ser obtidos com os autores.

Não se tem a pretensão de apresentar exemplos sofisticados, houve preocupação principal com aspectos didáticos.

Aconselhamos ao estudante experimentar os exemplos e realizar alterações nas mensagens (textos) que são exibidas, posicionamento dos widgets, acréscimo e remoção de widgets e outros, até que ele se sinta familiarizado com os recursos apresentados. Para estas mudanças é aconselhável consultar o Manual de Referência do wxWindows, onde detalhes e outros recursos são apresentados.

5.1 Criando um Frame

Em nosso primeiro contato com o *wxWindows* iremos introduzir dois dos principais conceitos necessários à construção de interfaces com o auxílio dessa ferramenta: o conceito de aplicação e o conceito de frame (moldura).

Uma aplicação pode ser entendida como um programa principal que fará uso de algumas das classes que compõem o *toolkit*. O toolkit pode ser visto como uma interface de mais alto nível ou abstração sobre o sistema de janela no qual é implementado. Para criar qualquer tipo de interface, é preciso criar, antes de mais nada, uma instância de uma aplicação (conexão com o sistema de janela) que, em *wxWindows*, é representada pela classe `wxApp`.

Um frame nada mais é do que uma das janelas visíveis da aplicação, representada pela classe `wxFrame`.

Ao se criar uma instância de uma aplicação (através da chamada ao método `OnInit`), é obrigatório que se crie também pelo menos um frame. Esta será a janela principal onde poderão ser colocados outros objetos que, juntos, irão compor a interface.

Através de uma instância de `wxApp`, é possível acessar diretamente os parâmetros fornecidos pela linha de comando quando da execução do programa, bem como interceptar um comando de término da aplicação, para que se possa executar algum processamento específico de finalização.

Além de objetos de tipos variados, um frame pode conter um título, uma barra de menus, uma linha de status ou um outro frame (frame secundário ou "filho"). Seu tamanho, posicionamento na tela principal, seu estado inicial (maximizado ou minimizado) e ícone associado podem ser definidos através de chamadas aos métodos apropriados.

Um frame também é responsável por gerenciar todas as operações executadas sobre os menus que possui.

A seguir apresentamos a listagem correspondente à implementação da Figura 1.

```
01 #include "wx.h"
02
03 // Definicoes de Classes
04 class MyFrame;
05 class MyApp: public wxApp
06 {
07   public:
08     MyFrame *frame;         // Esta aplicacao possui uma frame
09     wxFrame *OnInit(void);
10 };
11
12 class MyFrame: public wxFrame
13 {
14   public:
15     MyFrame(wxFrame *frame, char *title, int x, int y, int w, int h);
16     Bool OnClose(void);
17 };
18 MyApp myApp;
19
20 // Definicoes dos metodos relacionados
21 wxFrame *MyApp::OnInit(void)   // Metodo de inicializacao da aplicacao
22 {
23   // Criacao e inicializacao da frame principal
24   frame = new MyFrame(NULL, "Minha primeira aplica\347\343o", 50, 50, 300, 120);
25   frame->Show(TRUE);        //"Mostra" a frame
26   return frame;             // Retorna a frame principal
27 }
28
29 MyFrame::MyFrame(wxFrame *frame, char *title, int x, int y, int w, int h):
30               wxFrame(frame, title, x, y, w, h) {}
31
32 Bool MyFrame::OnClose(void)    // Metodo de termino da aplicacao
33 {
34   return TRUE;                 // Finaliza a aplicacao
35 }
```

O programa inicia com a inclusão do header `wx.h` (l. 1) que contém definições gerais para o uso dos widgets fornecidos pelo *wxWindows*. A classe derivada de `wxApp (MyApp) (l. 5)` representa a associação do programa com o *wxWindows*. Esta classe é a responsável pela declaração de um frame e contém o método `OnInit` (l. 21) que cria e retorna a janela principal da aplicação.

Um dos atributos da aplicação é uma instância da classe derivada de `wxFrame (MyFrame)` (l. 8) que é responsável pela definição da janela principal da aplicação. O frame é inicializado em `OnInit` (l. 24) e, logo a seguir, ativado pelo método `Show` (l. 25) que produz o efeito visto na Figura 1. O método `OnClose` (l. 32) é executado quando da finalização da aplicação, viabilizando o término da mesma pelo menu de controle da janela. Uma vez eliminado seu frame principal, a aplicação termina.

5.2 Panels e Buttons

A classe `wxPanel` define uma das subjanelas que podem ser inseridas num frame. É nesse tipo de subjanela que são depositados muitos dos itens que serão descritos ao longo deste tutorial. Seu tamanho, estilo e posicionamento dentro do frame correspondente são valores definidos no momento da instanciação.

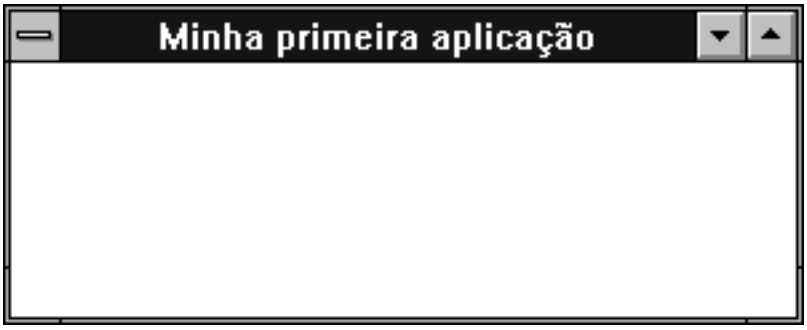


Figura 1: Minha primeira aplicação

Quando da criação de itens dentro de um panel, são levados em conta parâmetros de espaçamento vertical e horizontal, tipo de fonte utilizada e posicionamento dos labels relativos a cada item. Tais parâmetros podem ser ajustados de acordo com o desejo do usuário. Por fim, pode-se reajustar o tamanho do panel para que acomode melhor seus itens, caso necessário.

Um dos itens principais e mais comuns que podem ser colocados em um panel é um button. Este pode estar associado a uma cadeia de caracteres ou a um bitmap.

Buttons têm suas coordenadas e dimensões definidas no momento da instanciação, juntamente com a função que contém a ação associada ao seu pressionamento.

Opcionalmente, um button pode ser "nomeado" como *default button*, sendo acionado ao se pressionar a tecla `enter`. O tratamento de um button desse tipo pode ser feito de forma especial pelo respectivo panel, com o auxílio de alguns métodos específicos.

A figura 2 mostra a aparência de um button. O código fonte correspondente é apresentado a seguir.

```
01 #include "wx.h"
02
03 // Declara novas classes e funcoes para serem definidas mais tarde
04 class MyFrame;
05 class MyPanel;
06 void BotaoFim(wxButton& botao, wxCommandEvent& evento);
07
08 class MyApp: public wxApp
09 {
10   public:
11     wxFrame *OnInit(void);
12     MyFrame *frame;          // Esta aplicacao possui uma frame
13 };
14
15 class MyFrame: public wxFrame
16 {
17   public:
18     MyFrame(wxFrame *frame, char *title, int x, int y, int w, int h);
19     MyPanel *panel;          // Esta frame possui um panel
20     Bool OnClose(void);
21 };
22
23 class MyPanel: public wxPanel
24 {
```

```
25   public:
26     MyPanel(wxFrame *parent, int x, int y, int w, int h);
27     wxButton *botaofim;      // Este panel possui um botao
28 };
29 MyApp myApp;
30
31 // Definicoes dos metodos relacionados
32 wxFrame *MyApp::OnInit(void) // Metodo de inicializacao da aplicacao
33 {
34   // Criacao e inicializacao da frame principal
35   frame = new MyFrame(NULL, "Minha segunda aplica\347\343o", 50, 50, 300, 120);
36   return frame;              // Retorna a frame principal
37 }
38
39 MyFrame::MyFrame(wxFrame *frame, char *title, int x, int y, int w, int h):
40                  wxFrame(frame, title, x, y, w, h)
41 {
42   // Criacao e inicializacao do panel
43   panel = new MyPanel(this, 0, 0, 400, 300);
44   Show(TRUE);               // "Mostra" a frame
45 }
46
47 MyPanel::MyPanel(wxFrame *parent, int x, int y, int w, int h):
48                  wxPanel(parent, x, y, w, h)
49 {
50   // Cria o botao do panel
51   botaofim = new wxButton(this, (wxFunction)BotaoFim, "Fim");
52 }
53
54 Bool MyFrame::OnClose(void) // Metodo de termino da aplicacao
55 {
56   return TRUE;              // Finaliza a aplicacao
57 }
58
59 // Definicao da funcao associada ao botao
60 void BotaoFim(wxButton& bot, wxCommandEvent& evento)
61 {
62   delete myApp.frame;       // Deleta a frame principal
63 }
```

O panel é uma subjanela de um frame, nele podemos colocar vários itens como um botão, mensagens, lista de itens e janelas de texto, entre outros. Uma classe frame pode ter vários panels associados e a classe panel deve ter seus itens declarados como atributos.

Na classe `MyFrame` temos a inclusão de um atributo do tipo `MyPanel` (classe derivada de `wxPanel`) (l. 19) cuja finalidade é a definição de uma subjanela do frame. Na declaração do panel temos um item (atributo) do tipo `wxButton` (l. 27) que define um button para a janela. Na definição do método de entrada `OnInit (MyApp)` temos a instanciação do frame (l. 35) que provoca a criação da janela principal que é retornada pela aplicação.

Quando o frame é instanciado, seu construtor que provoca a criação do panel (instância do atributo `panel`) (l. 43) , posicionando o mesmo no frame. Uma vez instanciado, o panel executa o seu construtor que define o referido button (instância para o atributo `botaofim`) (l. 51). Este é posicionado no panel e relacionado com uma função responsável pelo término da aplicação `(BotaoFim)` (l. 60). Esta função deve ser declarada como do tipo `wxFunction`, um tipo especial para tratamento de ativação de eventos por itens do panel.

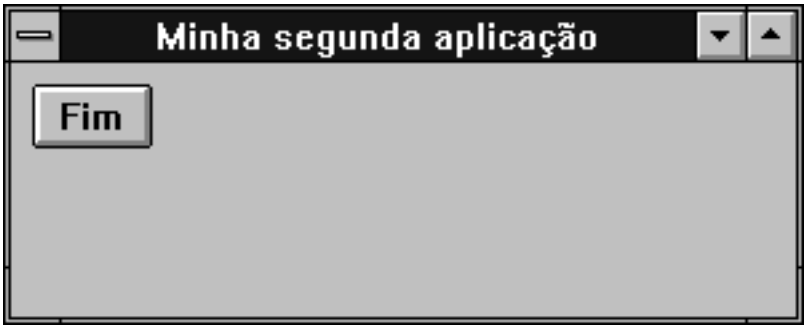


Figura 2: Minha segunda aplicação

5.3 Slider, CheckBoxes e RadioBoxes

Nesta seção serão abordados três novos widgets. O código fonte correspondente à figura 3 é apresentado a seguir:

```
001 #include "wx.h"
002
003 // Declara novas classes e funcoes para serem definidas mais tarde
004 class MyFrame;
005 class MyPanel1;
006 class MyPanel2;
007 void BotaoFim(wxButton& botao, wxCommandEvent& evento);
008 void BotaoStatus(wxButton& botao, wxCommandEvent& evento);
009 void Check1(wxCheckBox& check, wxCommandEvent& evento);
010 void Check2(wxCheckBox& check, wxCommandEvent& evento);
011 void Radio(wxRadioBox& radio, wxCommandEvent& evento);
012
013 class MyApp: public wxApp
014 {
015   public:
016     wxFrame *OnInit(void);
017     MyFrame *frame;          // Esta aplicacao possui uma frame
018 };
019
020 class MyFrame: public wxFrame
021 {
022   public:
023     MyFrame(wxFrame *frame, char *title, int x, int y, int w, int h);
024     MyPanel1 *panel1;          // Este frame possui dois panels
025     MyPanel2 *panel2;
026     Bool OnClose(void);
027 };
028
029 class MyPanel1: public wxPanel
030 {
031   public:
032     MyPanel1(wxFrame *parent, int x, int y, int w, int h);
033     wxMessage *msg1;         // Este panel possui uma mensagem,
034     wxButton *botaostatus;   // um botao,
035     wxCheckBox *check1;      // dois checkboxes
036     wxCheckBox *check2;      // e
037     wxRadioBox *radio;       // um radiobox
038 };
039
```

```
040 class MyPanel2: public wxPanel
041 {
042   public:
043     MyPanel2(wxFrame *parent, int x, int y, int w, int h);
044     wxMessage *msg2;         // Este panel possui uma mensagem,
045     wxButton *botaofim;      // um botao
046     wxSlider *slider;        // e um slider
047 };
048
049 // Cria uma instancia da classe e inicializa a aplicacao
050 MyApp myApp;
051
052 // Definicao dos metodos relacionados
053 wxFrame *MyApp::OnInit(void)
054 {
055   // Criacao e inicializacao da frame principal
056   frame = new MyFrame(NULL, "Uma aplica\347\343o mais sofisticada", 50, 50, 580, 400);
057
058   // Retorna a frame principal
059   return frame;
060 }
061
062 MyFrame::MyFrame(wxFrame *frame, char *title, int x, int y, int w, int h):
063   wxFrame(frame, title, x, y, w, h)
064 {
065   // Criacao e inicializacao dos panels
066   panel1 = new MyPanel1(this, 0, 0, 299, 400);
067   panel2 = new MyPanel2(this, 300, 0, 280, 400);
068   CreateStatusLine();      // Cria a linha de status e inicializa seu conteudo
069   SetStatusText("Uma linha de status em portugu\352s");
070   Show(TRUE);              // "Mostra" a frame
071 }
072
073 MyPanel1::MyPanel1(wxFrame *parent, int x, int y, int w, int h):
074     wxPanel(parent, x, y, w, h)
075 {
076   // Cria um vetor de opcoes para o radiobox
077   char *opcoes[3] = {"Portugu\352s","Ingl\352s","Franc\352s"};
078   // Cria a mensagem
079   msg1 = new wxMessage(this, "Panel Esquerdo", 95, 10);
080   // Cria o botao do panel
081   botaostatus = new wxButton(this, (wxFunction)BotaoStatus, "Limpa Status", 10, 50);
082   // Cria os dois checkboxes
083   check1 = new wxCheckBox(this, (wxFunction)Check1, "Habilita/Desabilita Fim", 10, 140);
084   check2 = new wxCheckBox(this, (wxFunction)Check2, "Habilita/Desabilita RadioBox", 10, 160);
085   // Cria o radiobox
086   radio = new wxRadioBox(this, (wxFunction)Radio, "L\355ngua", 7, 260, -1, -1, 3, opcoes);
087   // Habilita inicialmente os dois checkboxes
088   check1->SetValue(TRUE);
089   check2->SetValue(TRUE);
090 }
091
092 MyPanel2::MyPanel2(wxFrame *parent, int x, int y, int w, int h):
093     wxPanel(parent, x, y, w, h)
094 {
095   // Cria a mensagem
096   msg2 = new wxMessage(this, "Panel Direito", 100, 10);
097   // Cria o botao do panel
098   botaofim = new wxButton(this, (wxFunction)BotaoFim, "Fim", 218, 50);
099   // Cria o slider
100   slider = new wxSlider(this, NULL, "Valores", 1, 1, 100, 100, 10, 260);
101 }
```

```
102
103 // Funcao associada ao botao Limpa Status
104 void BotaoStatus(wxButton& botao, wxCommandEvent& evento)
105 {
106   myApp.frame->SetStatusText("");  // Limpa a linha de status
107 }
108
109 // Funcao associada ao botao fim
110 void BotaoFim(wxButton& botao, wxCommandEvent& evento)
111 {
112   delete myApp.frame;  // Deleta a frame principal, finalizando a aplicacao
113 }
114
115 // Funcao associada ao primeiro check
116 void Check1(wxCheckBox& check, wxCommandEvent& evento)
117 {
118   Bool chk = check.GetValue();
119   if (chk)                      // Habilita ou desabilita o botao Fim
120     myApp.frame->panel2->botaofim->Enable(TRUE);
121   else
122     myApp.frame->panel2->botaofim->Enable(FALSE);
123 }
124
125 // Funcao associada ao segundo check
126 void Check2(wxCheckBox& check, wxCommandEvent& evento)
127 {
128   Bool chk = check.GetValue();
129   for (int i = 0; i < 3; i++)
130     myApp.frame->panel1->radio->Enable(i\c{c}hk);
131 }
132
133 // Funcao associada ao radiobox
134 void Radio(wxRadioBox& radio, wxCommandEvent& evento)
135 {
136   int i = radio.GetSelection();
137   switch (i) {                  // Altera a mensagem da linha de status
138     case 0:
139       myApp.frame->SetStatusText("Uma linha de status em portugu\352s.");
140       break;
141     case 1:
142       myApp.frame->SetStatusText("A status line in english.");
143       break;
144     case 2:
145       myApp.frame->SetStatusText("Une linhe de status en fran\347ais.");
146   }
147 }
148
149 Bool MyFrame::OnClose(void)
150 {
151   return TRUE;      // Finaliza a aplicacao
152 }
```

Neste exemplo, começamos com a inclusão de dois panels independentes num mesmo frame (l. 24 e 25). Cada um desses panels, por sua vez, irá acomodar itens próprios que, no entanto, poderão interagir entre si.

No primeiro panel, temos uma instância de um message, um button, dois checkboxes e um radiobox (l. 33 a 37). No segundo panel, temos também um button, um message e um slider (l. 44 a 46). A instanciação do frame se dá da mesma forma que nos exemplos anteriores, só que agora, construímos dois panels e uma linha de status para a apresentação

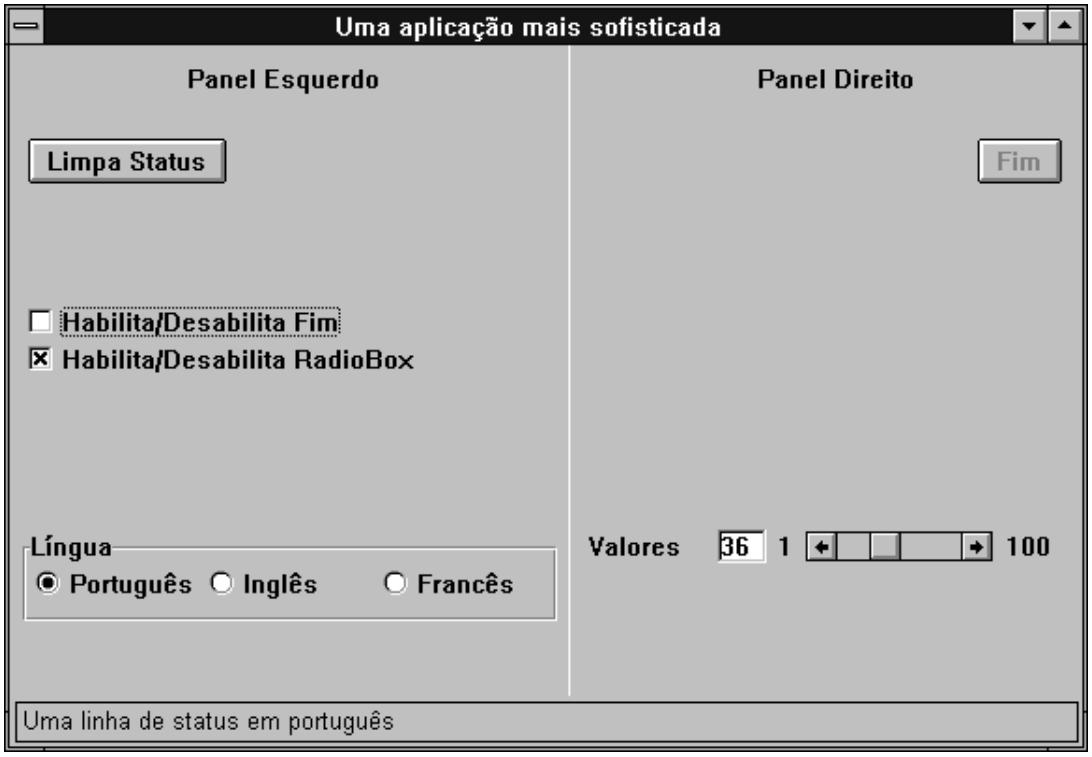


Figura 3: Uma aplicação mais sofisticada

de mensagens (l. 66 a 69). Os itens dos dois panels são instanciados dentro de seus respectivos construtores (l. 73 e 92), definindo-se seu posicionamento, labels, valores iniciais (quando for o caso) e funções de callback.

Como um exemplo de interação entre os diversos elementos da interface, temos o botão `Limpa Status` (que apaga o conteúdo da linha de status) (l. 106) e o primeiro checkbox (que desabilita o botão de finalização da aplicação de acordo com o seu estado corrente) (l. 118 a 122). O radiobox, por sua vez, é responsável por alterar o conteúdo da linha de status, dependendo da opção que esteja ativa (l. 136 a 145).

5.4 Text, TextWindow, Bitmap e Icon

Nesta seção iremos abordar a utilização de um text e um textwindow, bem como a inclusão de bitmaps e icons em uma aplicação. A seguir apresentamos o código fonte correspondente à figura 4:

Obs.: Devido à utilização de determinadas funções como, por exemplo, `SetSelection` este exemplo não roda em plataforma Windows.

```
001 #include "wx.h"
002 #include "aiai.xbm"   // Contem o vetor de bits do icone
003 #include "fload.xbm"  // Contem o vetor de bits do bitmap
```

```
004
005 // Declara novas classes e funcoes para serem definidas mais tarde
006 class MyFrame;
007 class MyPanel;
008 void BotaoFim(wxButton& botao, wxCommandEvent& evento);
009 void BotaoAbre(wxButton& botao, wxCommandEvent& evento);
010 void BotaoFind(wxButton& botao, wxCommandEvent& evento);
011 void BotaoReplace(wxButton& botao, wxCommandEvent& evento);
012 void Procura(char *xfind, long *from, long *to);
013
014 class MyApp: public wxApp
015 {
016   public:
017     wxFrame *OnInit(void);
018     MyFrame *frame;          // Esta aplicacao possui uma frame
019 };
020
021 class MyFrame: public wxFrame
022 {
023   public:
024     MyFrame(wxFrame *frame, char *title, int x, int y, int w, int h);
025     MyPanel *panel;          // Esta frame possui um panel
026     wxTextWindow *texto;     // e uma janela de texto
027     Bool OnClose(void);
028 };
029
030 class MyPanel: public wxPanel
031 {
032   public:
033     MyPanel(wxFrame *parent, int x, int y, int w, int h);
034     wxButton *botaoabre;
035     wxButton *botaofim;
036     wxButton *botaofind;
037     wxButton *botaoreplace;
038     wxText *find;
039     wxText *replace;
040 };
041
042 // Declara um bitmap e um icone
043 wxBitmap *bmp;
044 wxIcon   *icone;
045
046 // Cria uma instancia da classe e inicializa a aplicacao
047 MyApp myApp;
048
049 // Definicao dos metodos relacionados
050 wxFrame *MyApp::OnInit(void)
051 {
052   // Criacao e inicializacao da frame principal
053   frame = new MyFrame(NULL, "Um teste para Find/Replace", 100, 100, 530, 500);
054
055   // Retorna a frame principal
056   return frame;
057 }
058
059 MyFrame::MyFrame(wxFrame *frame, char *title, int x, int y, int w, int h):
060   wxFrame(frame, title, x, y, w, h)
061 {
062   // Criacao e inicializacao do panel
063   panel = new MyPanel(this, 0, 0, 530, 500);
064   texto = new wxTextWindow(this, 10, 110, 510, 390);
065   icone = new wxIcon(aiai_bits, aiai_width, aiai_height);
```

```
066   SetIcon(icone);          // Da um icone a frame
067   CreateStatusLine();      // Cria a linha de status
068   Show(TRUE);              // "Mostra" a frame
069 }
070
071 MyPanel::MyPanel(wxFrame *parent, int x, int y, int w, int h):
072   wxPanel(parent, x, y, w, h)
073 {
074   // Cria os botoes do panel
075   bmp = new wxBitmap(fload_bits, fload_width, fload_height);
076   botaoabre = new wxButton(this, (wxFunction)BotaoAbre, bmp, 10, 10);
077   botaofim = new wxButton(this, (wxFunction)BotaoFim, "Fim", 480, 10);
078   botaofind = new wxButton(this, (wxFunction)BotaoFind, "Encontra", 200, 60);
079   botaoreplace = new wxButton(this, (wxFunction)BotaoReplace, "Substitui", 200, 85);
080   // Cria os texts
081   find = new wxText(this, NULL, "Encontrar", "", 10, 60, 100);
082   replace = new wxText(this, NULL, "Substituir", "", 10, 85, 100);
083 }
084
085 // Funcao associada ao botao Fim
086 void BotaoFim(wxButton& botao, wxCommandEvent& evento)
087 {
088   delete myApp.frame;  // Deleta a frame principal, finalizando a aplicacao
089 }
090
091 // Funcao associada ao botao de abertura de arquivo
092 void BotaoAbre(wxButton& botao, wxCommandEvent& evento)
093 {
094   // Abre uma janela para o usuario procurar por um arquivo
095   char *s = wxFileSelector("Abrir Arquivo", NULL, NULL, NULL, "*.txt");
096   myApp.frame->texto->LoadFile(s);
097 }
098
099 // Funcao associada ao botao Encontrar
100 void BotaoFind(wxButton& botao, wxCommandEvent& evento)
101 {
102   long from, to;
103   myApp.frame->SetStatusText("");
104   char *xfind = myApp.frame->panel->find->GetValue();
105   Procura(xfind, &from, &to);
106   if (from != -1) {
107     myApp.frame->texto->ShowPosition(from);
108     myApp.frame->texto->SetSelection(from,to);
109     myApp.frame->texto->SetInsertionPoint(to);
110   }
111   else
112     myApp.frame->SetStatusText("Palavra n\343o encontrada!");
113 }
114
115 // Funcao associada ao botao Substituir
116 void BotaoReplace(wxButton& botao, wxCommandEvent& evento)
117 {
118   long from, to;
119   myApp.frame->SetStatusText("");
120   char *xfind = myApp.frame->panel->find->GetValue();
121   char *xreplace = myApp.frame->panel->replace->GetValue();
122   Procura(xfind, &from, &to);
123   if (from != -1) {
124     myApp.frame->texto->ShowPosition(from);
125     myApp.frame->texto->SetSelection(from,to);
126     myApp.frame->texto->SetInsertionPoint(to);
127     // Exibe uma caixa perguntando ao usuario o que ele deseja fazer
```

```
128     int ok = wxMessageBox("Confirma substitui\347\343o?","Confirma\347\343o",
129    wxYES_NO | wxCENTRE | wxICON_QUESTION);
130     if (ok == wxYES)
131       myApp.frame->texto->Replace(from,to,xreplace);
132   }
133   else
134     myApp.frame->SetStatusText("Palavra n\343o encontrada!");
135 }
136
137 Bool MyFrame::OnClose(void)
138 {
139   delete myApp.frame;  // Deleta a frame principal, finalizando a aplicacao
140   return TRUE;
141 }
142
143 // Funcao que procura pela palavra selecionando-a ao encontra-la
144 void Procura(char *xfind, long *from, long *to)
145 {
146   long xpos = myApp.frame->texto->GetInsertionPoint(); // Posicao atual
147   int nlines = myApp.frame->texto->GetNumberOfLines(); // numero de linhas
148   char linha[800]; // buffer para linha
149
150   // Xview (Implementacao interna de PositionToXY())
151   long x = 0;
152   long y = 0;
153   int n;
154   while (xpos > 0) {
155     n = myApp.frame->texto->GetLineLength(y) + 1;
156     if (xpos - n <= 0) {
157       x = xpos;
158       y--;
159     }
160     xpos = xpos - n;
161     y++;
162   }
163
164   char *p;
165   for (long i = y; i < nlines; i++) {
166     n = myApp.frame->texto->GetLineText(i,linha);  // linha de texto
167     linha[n] = 0;
168     p = strstr(linha + x,xfind);  // Procura seq. de carac. na linha
169         // Encontrou a palavra
170     if (p) {
171       x = p - linha;
172       *from = myApp.frame->texto->XYToPosition(x,i);
173       *to = myApp.frame->texto->XYToPosition(x+strlen(xfind)-1,i);
174       return;
175     }
176     x = 0;     // Procura desde o inicio da linha
177   }
178   *from = -1;  // Nao encontrou a palavra
179   return;
180 }
```

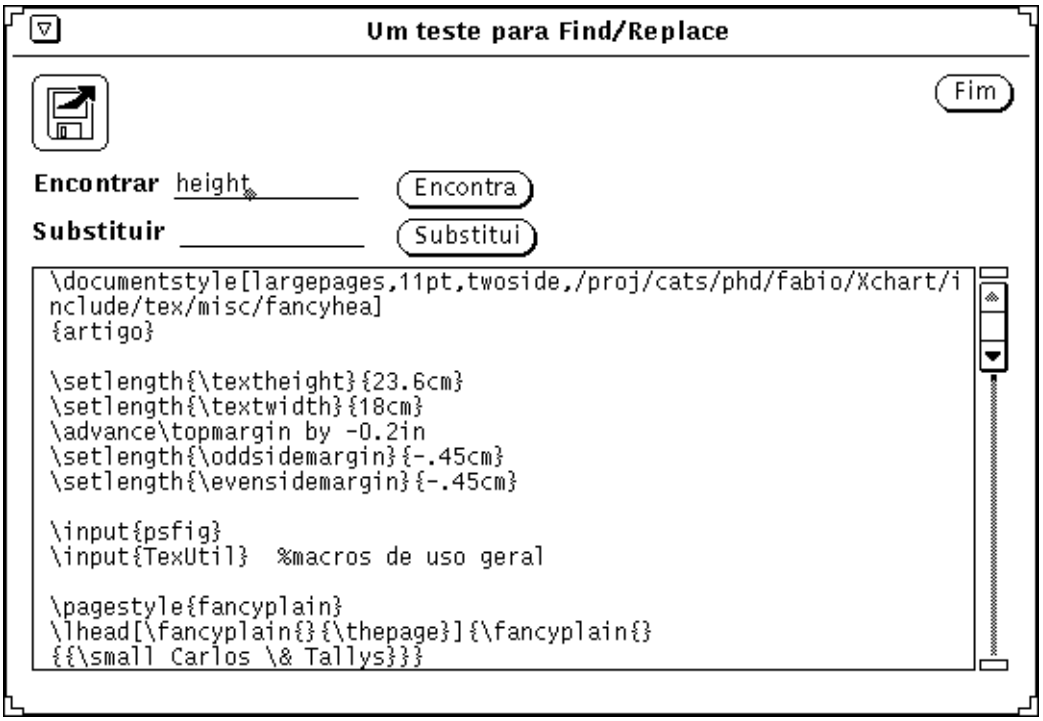


Figura 4: Um teste para find replace

Esta aplicação possui um frame, que agora contém uma nova subjanela do tipo `wxTextWindow` (l. 26) e um panel com um novo tipo de item, um text (l. 38 e 39). Além disso, foram incluídos os cabeçalhos de dois arquivos do tipo XBM, que serão utilizados na criação de um bitmap e um icon para a aplicação (l. 2 e 3).

O bitmap, que será associado ao botão de abertura de arquivo, é instanciado dentro do construtor do panel, utilizando variáveis predefinidas dentro do arquivo *fload.xbm* (respectivos `bits`, `width` e `height`) (l. 75). Em seguida, é colocado no lugar do label do botão quando da instanciação do mesmo (l. 76).

O ícone é criado dentro do construtor do frame, de forma semelhante ao bitmap (l. 65). Em seguida, o método `SetIcon` faz com que ele se torne o ícone associado ao frame em seu estado minimizado (l. 66).

Dentro da função de callback do botão de abertura de arquivo, utilizamos uma função do wxWindows especialmente criada para facilitar esta operação (`wxFileSelector`) (l. 95), permitindo a navegação pelos diretórios do disco e retornando o nome do arquivo selecionado pelo usuário, juntamente com o respectivo path.

A procura de palavras no texto (função `BotaoFind`) (l. 100)é feita da seguinte forma: recupera-se o conteúdo do primeiro text (l. 104) e, a partir da posição atual do cursor, vai-se percorrendo as linhas até que a palavra seja encontrada ou até que seja atingido o fim do arquivo. Caso se encontre a palavra, os métodos `ShowPosition` e `SetSelection` (l. 107 e 108) são chamados, respectivamente, para exibir a linha que contém a palavra e para destacá-la. Caso contrário, uma mensagem de falha é exibida na linha de status (l. 112).

A substituição de palavras é feita de forma semelhante à busca, só que, quando a palavra é encontrada, utilizamos uma outra função do wxWindows (`wxMessageBox`) bastante adequada à entrada de feedbacks por parte do usuário (l. 128). O formato utilizado aqui apresenta uma pergunta com duas opções de escolha (Sim ou Não), permitindo que o programa tome a decisão apropriada. Caso a substituição seja confirmada, o método `Replace` executa a tarefa (l. 131).

5.5 Canvas, Menu, MenuBar, Brush, Pen, Cursor e DC

Nesta seção iremos tratar mais uma subjanela de um frame (`wxCanvas`), juntamente com seus itens e métodos mais comumente utilizados. Outro widget importante (`wxMenu`) também será abordado.

Abaixo segue o código fonte correspondente à figura 5, que implementa um pequeno editor gráfico:

```
001 #include "wx.h"
002
003 #include "math.h"    // Contem declaracoes de funcoes matematicas
004
005 // Definicao de constantes utilizadas no programa
006 #define LIMPAR 1     // Opcoes do menu
007 #define SAIR   2
008 #define RED    3
009 #define GREEN  4
010 #define BLUE   5
011 #define SOBRE  6
012
013 #define LINHA  0     // Opcoes do radiobox
014 #define RET    1
015 #define CIRC   2
016
017 // Declara novas classes e funcoes para serem definidas mais tarde
018 class MyFrame;
019 class MyPanel;
020 class MyCanvas;
021 void Radio(wxRadioBox& radio, wxCommandEvent& evento);
022
023 class MyApp: public wxApp
024 {
025   public:
026     wxFrame *OnInit(void);
027     wxMenu  *menu_cor;       // Esta aplicacao possui um menu
028     MyFrame *frame;          // e um frame
029 };
030
031 class MyFrame: public wxFrame
032 {
033   public:
034     MyFrame(wxFrame *frame, char *title, int x, int y, int w, int h);
035     MyPanel *panel;             // Esta frame possui um panel
036     MyCanvas *canvas;           // e um canvas
037     int cor;                    // Cor atual dos desenhos
038     void OnMenuCommand(int id);
039     Bool OnClose(void);
040 };
041
042 class MyPanel: public wxPanel
043 {
044   public:
045     MyPanel(wxFrame *parent, int x, int y, int w, int h);
046     wxRadioBox *radio;
047 };
048
049 class MyCanvas: public wxCanvas
050 {
051   public:
052     MyCanvas(wxFrame *parent, int x, int y, int w, int h);
```

```
053     wxCanvasDC *dc;
054     int tipo;       // Tipo de desenho
055     float xant;     // Coordenadas atuais
056     float yant;     // do desenho
057     void OnEvent(wxMouseEvent& evento);
058 };
059
060 // Declara um icone
061 wxIcon *icone;
062
063 // Declara vetores de canetas e pinceis
064 wxBrush *brush[3];
065 wxPen   *pen[3];
066
067 // Cria uma instancia da classe e inicializa a aplicacao
068 MyApp myApp;
069
070 // Definicao dos metodos relacionados
071 wxFrame *MyApp::OnInit(void)
072 {
073   // Cria as canetas e os pinceis
074   pen[0]   = new wxPen("RED",1,wxSOLID);
075   pen[1]   = new wxPen("GREEN",1,wxSOLID);
076   pen[2]   = new wxPen("BLUE",1,wxSOLID);
077   brush[0] = new wxBrush("RED",wxSOLID);
078   brush[1] = new wxBrush("GREEN",wxSOLID);
079   brush[2] = new wxBrush("BLUE",wxSOLID);
080
081   // Criacao e inicializacao da frame principal
082   frame = new MyFrame(NULL, "Um simples editor gr\341fico", 100, 100, 530, 465);
083
084   // Cria alguns menus
085   wxMenu *menu_desenho = new wxMenu;
086          menu_cor     = new wxMenu;
087   wxMenu *menu_ajuda   = new wxMenu;
088
089   // Inicializa as opcoes dos menus
090   menu_desenho->Append(LIMPAR, "Limpar");
091   menu_desenho->AppendSeparator();
092   menu_desenho->Append(SAIR, "Sair");
093
094   menu_cor->Append(RED, "Vermelho");
095   menu_cor->Append(GREEN, "Verde");
096   menu_cor->Append(BLUE, "Azul");
097   menu_cor->Enable(RED,FALSE);
098
099   menu_ajuda->Append(SOBRE, "Sobre");
100
101   // Cria uma barra de menus
102   wxMenuBar *barra_menu = new wxMenuBar;
103
104   // Insere os menus na barra de menus
105   barra_menu->Append(menu_desenho, "Desenho");
106   barra_menu->Append(menu_cor, "Cor");
107   barra_menu->Append(menu_ajuda, "Ajuda");
108   frame->SetMenuBar(barra_menu);
109
110   // Retorna a frame principal
111   return frame;
112 }
113
114 MyFrame::MyFrame(wxFrame *frame, char *title, int x, int y, int w, int h):
```

```
115   wxFrame(frame, title, x, y, w, h)
116 {
117   cor = RED;      // Seta a cor inicial
118   // Criacao e inicializacao do panel
119   panel  = new MyPanel(this, 0, 0, 530, 74);
120   canvas = new MyCanvas(this, 10, 75, 510, 390);
121   icone  = new wxIcon("mondrian");
122   SetIcon(icone); // Da um icone a frame
123   Show(TRUE);     // "Mostra" a frame
124 }
125
126 MyPanel::MyPanel(wxFrame *parent, int x, int y, int w, int h):
127   wxPanel(parent, x, y, w, h)
128 {
129   // Cria um vetor de opcoes para o radiobox
130   char *opcoes[3] = {"Linha","Ret\342ngulo","C\355rculo"};
131   // Cria o radiobox
132   radio = new wxRadioBox(this, (wxFunction)Radio, "Figura", 125, 15, -1, -1, 3, opcoes);
133 }
134
135 MyCanvas::MyCanvas(wxFrame *parent, int x, int y, int w, int h):
136                    wxCanvas(parent,x,y,w,h)
137 {
138   dc   = GetDC();     // "Pega" o DC do canvas
139   tipo = LINHA;       // Inicializa os valores
140   xant = -1;          // do tipo de desenho e
141   yant = -1;          // coordenadas iniciais
142   dc->SetBrush(brush[0]); // Seta os tipos de pincel
143   dc->SetPen(pen[0]);     // e caneta iniciais
144   wxCursor *cursor = new wxCursor(wxCURSOR_PENCIL);
145   SetCursor(cursor);  // Seta o tipo de cursor para o canvas
146 }
147
148 // Funcao associada ao acionamento de alguma opcao do menu
149 void MyFrame::OnMenuCommand(int id)
150 {
151   switch (id) {
152     case LIMPAR:
153       myApp.frame->canvas->dc->Clear(); // Limpa o canvas
154       break;
155     case SAIR:
156       delete myApp.frame;               // Finaliza o programa
157       break;
158     case RED:                           // Muda a cor do pincel
159     case GREEN:                         // e da caneta
160     case BLUE:
161       myApp.menu_cor->Enable(cor,TRUE);
162       cor = id;
163       myApp.menu_cor->Enable(cor,FALSE);
164       myApp.frame->canvas->dc->SetBrush(brush[id-3]);
165       myApp.frame->canvas->dc->SetPen(pen[id-3]);
166       break;
167     case SOBRE:                         // Exibe a caixa sobre
168       wxMessageBox("\n\
169 ED Graphic\nVers\343o 1.0\n\n\
170 Programadores:\n\n\
171 Carlos Neves Junior\n\
172 Tallys Hoover Yunes\n","Sobre",wxOK | wxCENTRE);
173   }
174 }
175
176 // Funcao que intercepta os cliques de mouse no canvas
```

```
177 void MyCanvas::OnEvent(wxMouseEvent& evento)
178 {
179   float x, y;                  // "Pega" as coordenadas do
180   evento.Position(&x, &y);     // clique de mouse
181
182   if (evento.ButtonDown(1)) {  // Se pressionou o botao esquerdo...
183     if (xant == -1) {
184       dc->DrawPoint(x,y);
185       xant = x;
186       yant = y;
187     }
188     else if (tipo != LINHA) {
189       if (tipo == RET)
190      dc->DrawRectangle(xant, yant, fabs(xant - x), fabs(yant - y));
191
192       else if (tipo == CIRC) {
193      float raio = sqrt(pow((xant - x),2) + pow((yant - y),2));
194      dc->DrawEllipse(xant - raio, yant - raio, 2*raio, 2*raio);
195       }
196       xant = -1;
197       yant = -1;
198     }
199   }
200   else if (evento.ButtonUp(1) && tipo == LINHA) {
201     xant = -1;  // Se liberou o botao esquerdo,
202     yant = -1;  // "apaga" as coordenadas
203   }
204   else if (evento.Dragging() && evento.LeftIsDown() && tipo == LINHA) {
205     dc->DrawLine(xant, yant, x, y);
206     xant = x;
207     yant = y;
208   }
209 }
210
211 // Funcao associada ao acionamento do radiobox
212 void Radio(wxRadioBox& radio, wxCommandEvent& evento)
213 {
214   // Altera o tipo de desenho de acordo com a selecao do radiobox
215   myApp.frame->canvas->tipo = radio.GetSelection();
216 }
217
218 Bool MyFrame::OnClose(void)
219 {
220   return TRUE;         // Finaliza a aplicacao
221 }
```

Inicialmente temos a definição das constantes associadas aos identificadores das opções dos menus e do radiobox (l. 6 a 15). Em seguida, definimos o frame, contendo uma subjanela do tipo `wxCanvas` (l. 36), o panel (l. 42) e o próprio canvas (l. 49).

Um desenho dentro de um canvas utiliza um certo tipo de brush e pen, desse modo, declaramos três tipos de cada um desses itens (l. 64 e 65), a fim de proporcionar mais flexibilidade na edição dos gráficos. Os pens e brushes são instanciados dentro do método de inicialização da aplicação, tendo definidos seus estilos e cores (l. 74 a 79).

Os menus, por sua vez, são construídos através da adição de opções a uma instância da classe `wxMenu` (l. 85 a 99). Estando prontos os menus, constrói-se a barra de menus, adicionando-se um a um os menus à instância da classe `wxMenuBar` (l. 105 a 107). Por fim, associamos o menubar ao frame (l. 108).

Quando da "construção" do canvas, recuperamos seu *device context* (através da chamada ao método `GetDC` (l. 138)), onde serão feitos os desenhos e criamos um cursor (instância da classe `wxCursor`) a partir de um modelo de cursor predefinido (`wxCURSOR_PENCIL`) (l. 144), que será utilizado como indicador da posição do mouse durante os desenhos.

O tratamento da ativação das opções do menu é feito pelo frame correspondente através do método `OnMenuCommand` (l. 149), que retorna o identificador do item selecionado. Dentre as opções possíveis, temos a eliminação do conteúdo do canvas (`Clear`) (l. 153), o fechamento da aplicação (l. 156), a mudança da cor utilizada nos desenhos (`SetBrush` e `SetPen`) (l. 164 e 165) e a exibição de informações sobre o programa (l. 168 a 172).

Eventos gerados pelo mouse dentro do canvas são interceptados pelo método `OnEvent` (l. 177), dentro do qual podemos identificar a posição de ocorrência do evento e o seu tipo. Dessa forma, o desenho apropriado é feito de acordo com o tipo de desenho atual (armazenado no radiobox e na variável `tipo` (l. 54), cor (armazenada na variável `cor` (l. 37) e indicada no menu como uma opção desabilitada) e evento gerado pelo usuário.

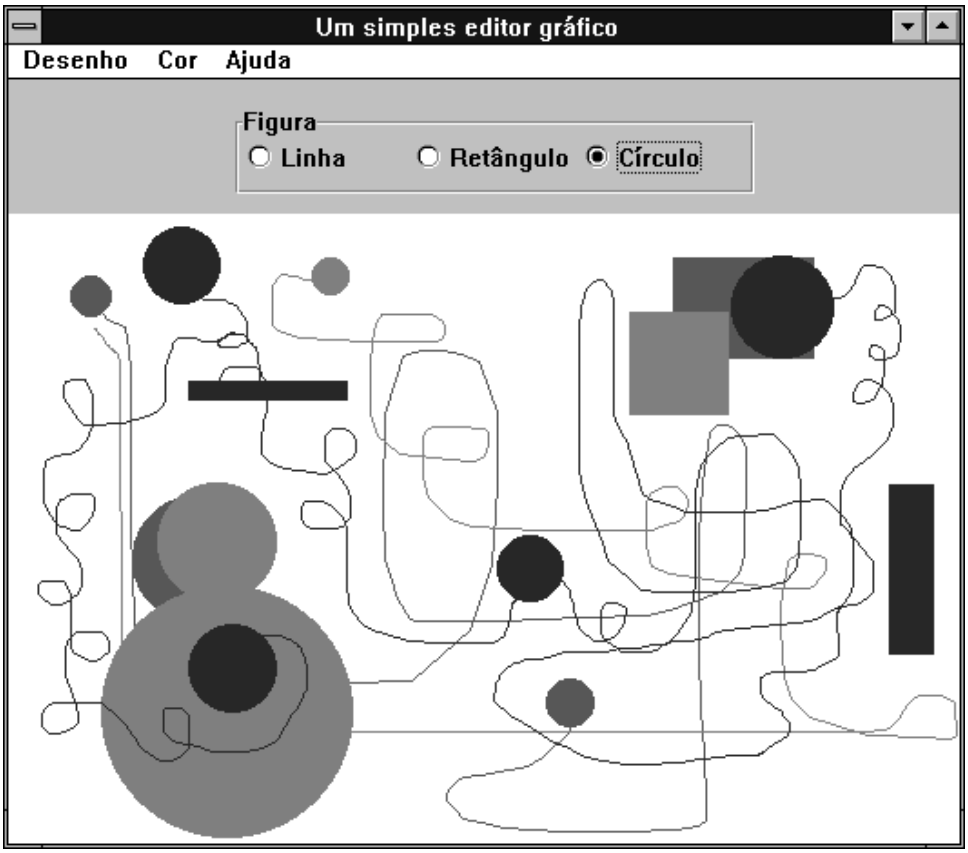


Figura 5: Um simples editor gráfico

Relatórios Técnicos – 1992

92-01 **Applications of Finite Automata Representing Large Vocabularies,** *C. L. Lucchesi, T. Kowaltowski*

92-02 **Point Set Pattern Matching in d-Dimensions,** *P. J. de Rezende, D. T. Lee*

92-03 **On the Irrelevance of Edge Orientations on the Acyclic Directed Two Disjoint Paths Problem,** *C. L. Lucchesi, M. C. M. T. Giglio*

92-04 **A Note on Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams,** *W. Jacometti*

92-05 **An (l, u)-Transversal Theorem for Bipartite Graphs,** *C. L. Lucchesi, D. H. Younger*

92-06 **Implementing Integrity Control in Active Databases,** *C. B. Medeiros, M. J. Andrade*

92-07 **New Experimental Results For Bipartite Matching,** *J. C. Setubal*

92-08 **Maintaining Integrity Constraints across Versions in a Database,** *C. B. Medeiros, G. Jomier, W. Cellary*

92-09 **On Clique-Complete Graphs,** *C. L. Lucchesi, C. P. Mello, J. L. Szwarcfiter*

92-10 **Examples of Informal but Rigorous Correctness Proofs for Tree Traversing Algorithms,** *T. Kowaltowski*

92-11 **Debugging Aids for Statechart-Based Systems,** *V. G. S. Elias, H. Liesenberg*

92-12 **Browsing and Querying in Object-Oriented Databases,** *J. L. de Oliveira, R. de O. Anido*

Relatórios Técnicos – 1993

93-01 **Transforming Statecharts into Reactive Systems,** *Antonio G. Figueiredo Filho, Hans K. E. Liesenberg*

93-02 **The Hierarchical Ring Protocol: An Efficient Scheme for Reading Replicated Data,** *Nabor das C. Mendonça, Ricardo de O. Anido*

93-03 **Matching Algorithms for Bipartite Graphs,** *Herbert A. Baier Saip, Cláudio L. Lucchesi*

93-04 **A lexBFS Algorithm for Proper Interval Graph Recognition,** *Celina M. H. de Figueiredo, João Meidanis, Célia P. de Mello*

93-05 **Sistema Gerenciador de Processamento Cooperativo,** *Ivonne. M. Carrazana, Nelson. C. Machado, Célio. C. Guimarães*

93-06 **Implementação de um Banco de Dados Relacional Dotado de uma Interface Cooperativa,** *Nascif A. Abousalh Neto, Ariadne M. B. R. Carvalho*

93-07 **Estadogramas no Desenvolvimento de Interfaces,** *Fábio N. de Lucena, Hans K. E. Liesenberg*

93-08 **Introspection and Projection in Reasoning about Other Agents,** *Jacques Wainer*

93-09 **Codificação de Seqüências de Imagens com Quantização Vetorial,** *Carlos Antonio Reinaldo Costa, Paulo Lício de Geus*

93-10 **Minimização do Consumo de Energia em um Sistema para Aquisição de Dados Controlado por Microcomputador,** *Paulo Cesar Centoducatte, Nelson Castro Machado*

93-11 **An Implementation Structure for RM-OSI/ISO Transaction Processing Application Contexts,** *Flávio Morais de Assis Silva, Edmundo Roberto Mauro Madeira*

93-12 **Boole's conditions of possible experience and reasoning under uncertainty,** *Pierre Hansen, Brigitte Jaumard, Marcus Poggi de Aragão*

93-13 **Modelling Geographic Information Systems using an Object Oriented Framework,** *Fatima Pires, Claudia Bauzer Medeiros, Ardemiris Barros Silva*

93-14 **Managing Time in Object-Oriented Databases,** *Lincoln M. Oliveira, Claudia Bauzer Medeiros*

93-15 **Using Extended Hierarchical Quorum Consensus to Control Replicated Data: from Traditional Voting to Logical Structures,** *Nabor das Chagas Mendonça, Ricardo de Oliveira Anido*

93-16 $\mathcal{LL}$ – **An Object Oriented Library Language Reference Manual,** *Tomasz Kowaltowski, Evandro Bacarin*

93-17 **Metodologias para Conversão de Esquemas em Sistemas de Bancos de Dados Heterogêneos,** *Ronaldo Lopes de Oliveira, Geovane Cayres Magalhães*

93-18 **Rule Application in GIS – a Case Study,** *Claudia Bauzer Medeiros, Geovane Cayres Magalhães*

93-19 **Modelamento, Simulação e Síntese com VHDL,** *Carlos Geraldo Krüger e Mário Lúcio Côrtes*

93-20 **Reflections on Using Statecharts to Capture Human-Computer Interface Behaviour,** *Fábio Nogueira de Lucena e Hans Liesenberg*

93-21 **Applications of Finite Automata in Debugging Natural Language Vocabularies,** *Tomasz Kowaltowski, Cláudio Leonardo Lucchesi e Jorge Stolfi*

93-22 **Minimization of Binary Automata,** *Tomasz Kowaltowski, Cláudio Leonardo Lucchesi e Jorge Stolfi*

93-23 **Rethinking the DNA Fragment Assembly Problem,** *João Meidanis*

93-24 EGOLib — **Uma Biblioteca Orientada a Objetos Gráficos,** *Eduardo Aguiar Patrocínio, Pedro Jussieu de Rezende*

93-25 **Compreensão de Algoritmos através de Ambientes Dedicados a Animação,** *Rackel Valadares Amorim, Pedro Jussieu de Rezende*

93-26 GeoLab: **An Environment for Development of Algorithms in Computational Geometry,** *Pedro Jussieu de Rezende, Welson R. Jacometti*

93-27 **A Unified Characterization of Chordal, Interval, Indifference and Other Classes of Graphs,** *João Meidanis*

93-28 **Programming Dialogue Control of User Interfaces Using Statecharts,** *Fábio Nogueira de Lucena e Hans Liesenberg*

93-29 EGOLib – **Manual de Referência,** *Eduardo Aguiar Patrocínio e Pedro Jussieu de Rezende*

Relatórios Técnicos – 1994

94-01 **A Statechart Engine to Support Implementations of Complex Behaviour,** *Fábio Nogueira de Lucena, Hans K. E. Liesenberg*

94-02 **Incorporação do Tempo em um SGBD Orientado a Objetos,** *Ângelo Roncalli Alencar Brayner, Claudia Bauzer Medeiros*

94-03 **O Algoritmo KMP através de Autômatos,** *Marcus Vinícius A. Andrade e Cláudio L. Lucchesi*

94-04 **On Edge-Colouring Indifference Graphs,** *Celina M. H. de Figueiredo, João Meidanis, Célia Picinin de Mello*

94-05 **Using Versions in GIS,** *Claudia Bauzer Medeiros and Geneviève Jomier*

94-06 **Times Assíncronos: Uma Nova Técnica para o Flow Shop Problem,** *Hélvio Pereira Peixoto e Pedro Sérgio de Souza*

94-07 **Interfaces Homem-Computador: Uma Primeira Introdução,** *Fábio Nogueira de Lucena e Hans K. E. Liesenberg*

94-08 **Reasoning about another agent through empathy,** *Jacques Wainer*

94-09 **A Prolog morphological analyser for Portuguese,** *Jacques Wainer, Alexandre Farcic*

94-10 **Introdução aos Estadogramas,** *Fábio N. de Lucena, Hans K. E. Liesenberg*

94-11 **Matching Covered Graphs and Subdivisions of K_4 and $\overline{C_6}$,** *Marcelo H. de Carvalho and Cláudio L. Lucchesi*

94-12 **Uma Metodologia de Especificação de Times Assíncronos,** *Hélvio Pereira Peixoto, Pedro Sérgio de Souza*

Relatórios Técnicos – 1995

95-01 **Paradigmas de algoritmos na solução de problemas de busca multidimensional,** *Pedro J. de Rezende, Renato Fileto*

95-02 **Adaptive enumeration of implicit surfaces with affine arithmetic,** *Luiz Henrique de Figueiredo, Jorge Stolfi*

95-03 **W3 no Ensino de Graduação?,** *Hans Liesenberg*

95-04 **A greedy method for edge-colouring odd maximum degree doubly chordal graphs,** *Celina M. H. de Figueiredo, João Meidanis, Célia Picinin de Mello*

95-05 **Protocols for Maintaining Consistency of Replicated Data,** *Ricardo Anido, N. C. Mendonça*

95-06 **Guaranteeing Full Fault Coverage for UIO-Based Methods,** *Ricardo Anido and Ana Cavalli*

95-07 **Xchart-Based Complex Dialogue Development,** *Fábio Nogueira de Lucena, Hans K.E. Liesenberg*

95-08 **A Direct Manipulation User Interface for Querying Geographic Databases,** *Juliano Lopes de Oliveira, Claudia Bauzer Medeiros*

95-09 **Bases for the Matching Lattice of Matching Covered Graphs,** *Cláudio L. Lucchesi, Marcelo H. Carvalho*

95-10 **A Highly Reconfigurable Neighborhood Image Processor based on Functional Programming,** *Neucimar J. Leite, Marcelo A. de Barros*

95-11 **Processador de Vizinhança para Filtragem Morfológica,** *Ilka Marinho Barros, Roberto de Alencar Lotufo, Neucimar Jerônimo Leite*

95-12 **Modelos Computacionais para Processamento Digital de Imagens em Arquiteturas Paralelas,** *Neucimar Jerônimo Leite*

95-13 **Modelos de Computação Paralela e Projeto de Algoritmos,** *Ronaldo Parente de Menezes e João Carlos Setubal*

95-14 **Vertex Splitting and Tension-Free Layout,** *P. Eades, C. F. X. de Mendonça N.*

95-15 **NP-Hardness Results for Tension-Free Layout,** *C. F. X. de Mendonça N., P. Eades, C. L. Lucchesi, J. Meidanis*

95-16 **Agentes Replicantes e Algoritmos de Eco,** *Marcos J. C. Euzébio*

95-17 **Anais da II Oficina Nacional em Problemas Combinatórios: Teoria, Algoritmos e Aplicações,** *Editores: Marcus Vinicius S. Poggi de Aragão, Cid Carvalho de Souza*

Departamento de Ciência da Computação — IMECC
Caixa Postal 6065
Universidade Estadual de Campinas
13081-970 – Campinas – SP
BRASIL

reltec@dcc.unicamp.br