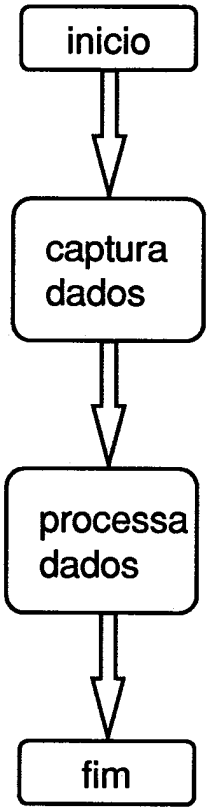
Carlos Henrique Levy

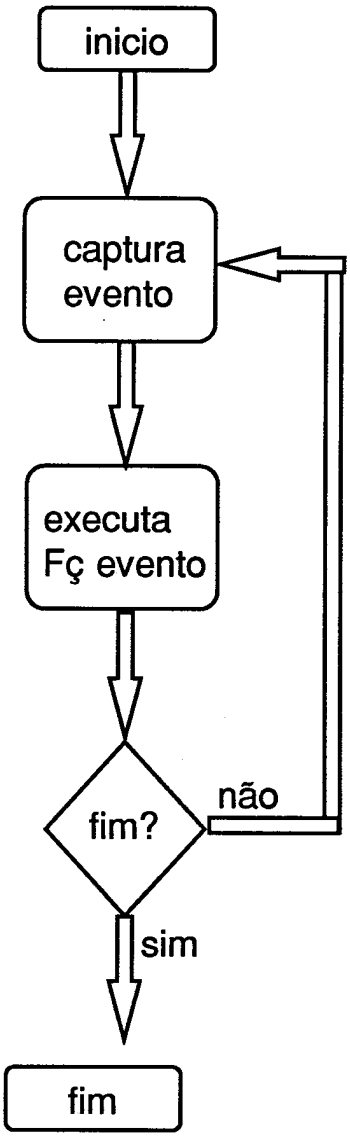
IUP/LED: Uma Ferramenta Portátil de Interface com Usuário

Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro

Agosto 1993

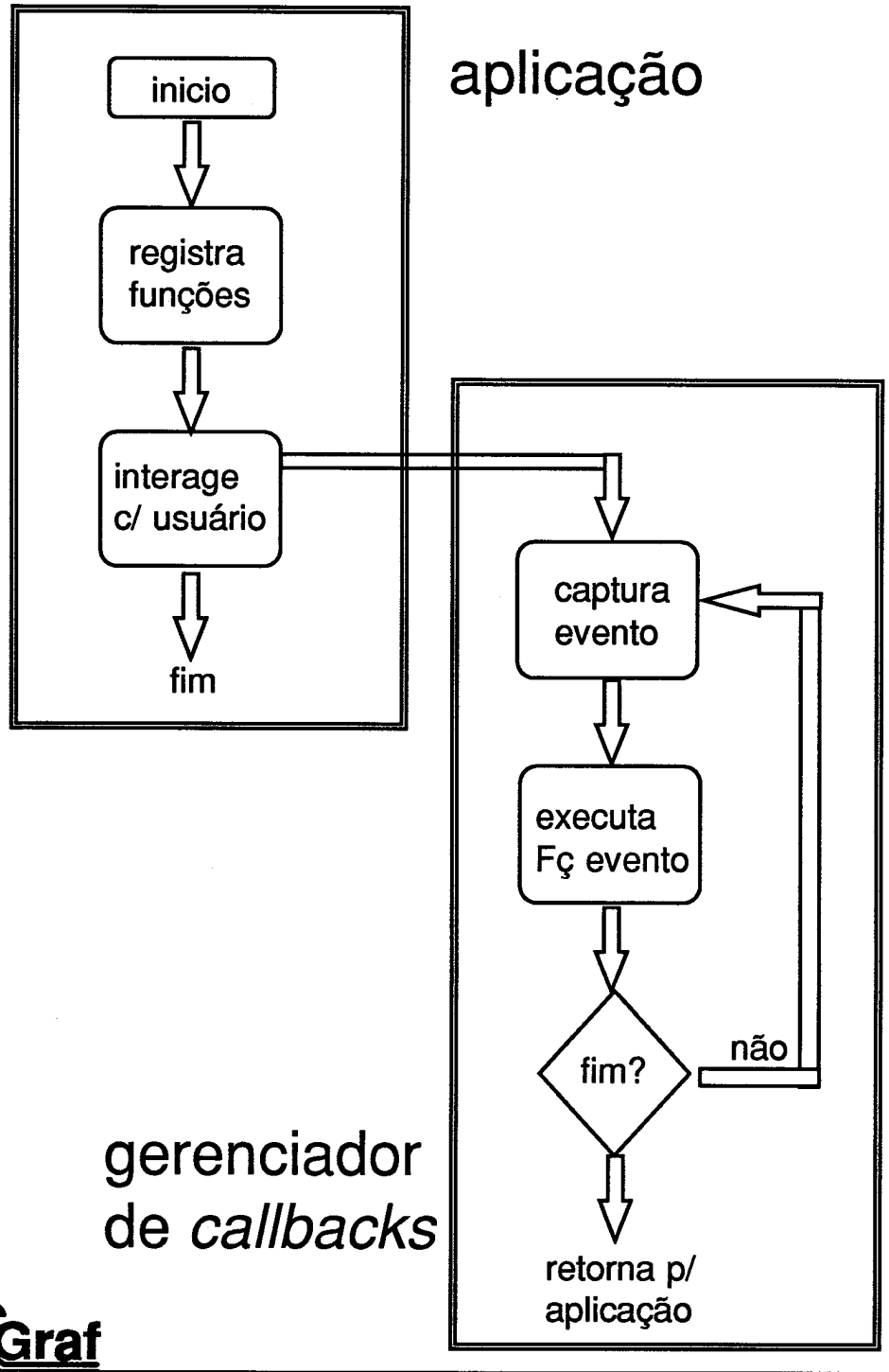
Programação Convencional
vs.
Programação Orientada a Eventos





TeCGraf

Programação por *Callbacks*



TeCGraf

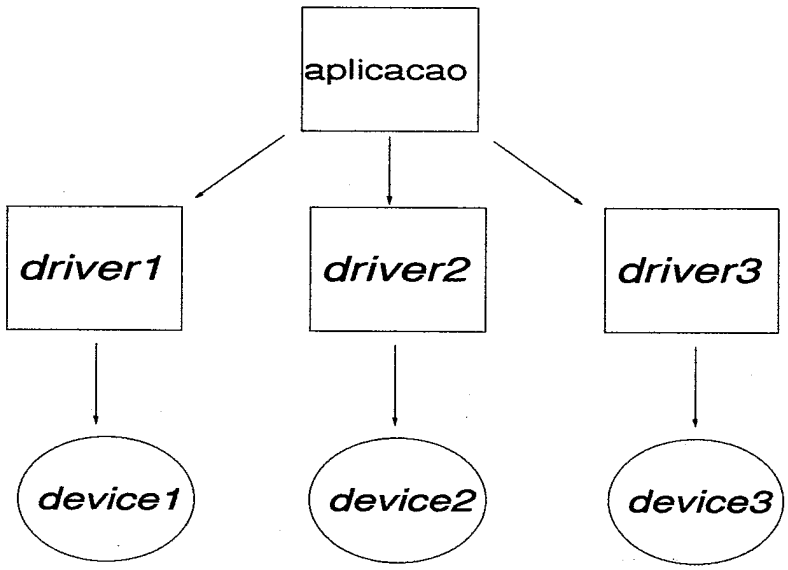
Portabilidade

- Portátil × Portável
  - ▷ minimizar re-programação

- Obstáculos potenciais
  - ▷ *hardware*
  - ▷ sistema operacional
  - ▷ compilador
  - ▷ dispositivo de saída
  - ▷ sistema de interface

Portabilidade de Aplicações Gráficas

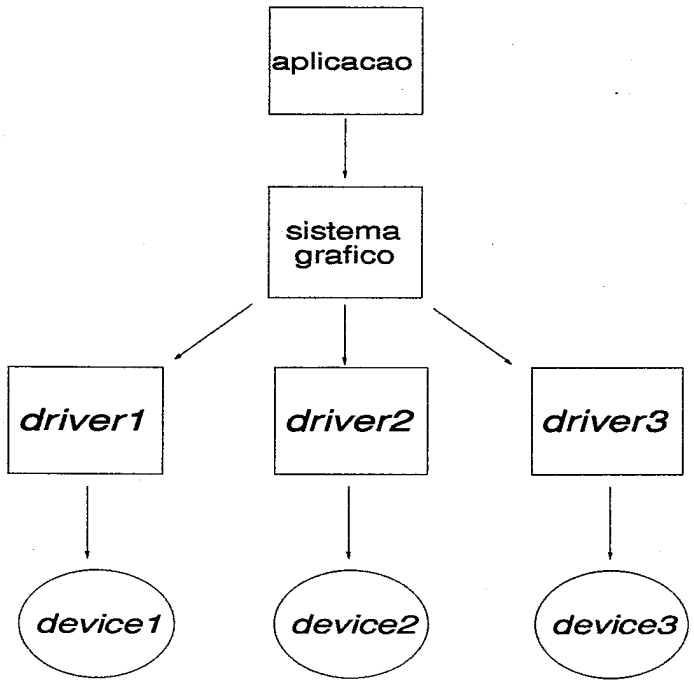
- Estratégia com *drivers*



- Vantagens
  - ▷ eficiência
- Desvantagens
  - ▷ API não portátil
  - ▷ programador especialista no dispositivo
  - ▷ variedades de dispositivo
  - ▷ influencia estrutura da aplicação

Portabilidade de Aplicações Gráficas

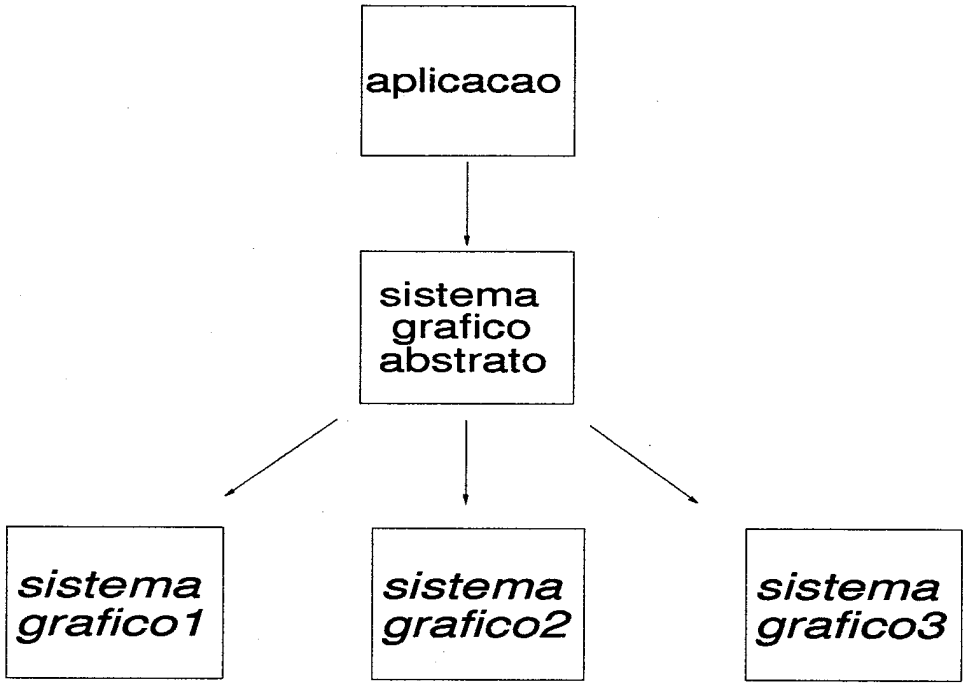
- Estratégia com sistema gráfico



- Vantagens
  - ▷ sistema independente de dispositivo
  - ▷ portabilidade reduzida ao sistema gráfico
- Desvantagens
  - ▷ não há padrão
  - ▷ eficiência

Portabilidade de Aplicações Gráficas

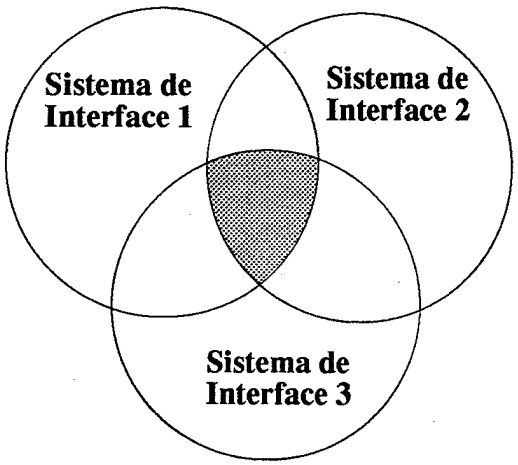
- Estratégia com sistema gráfico abstrato



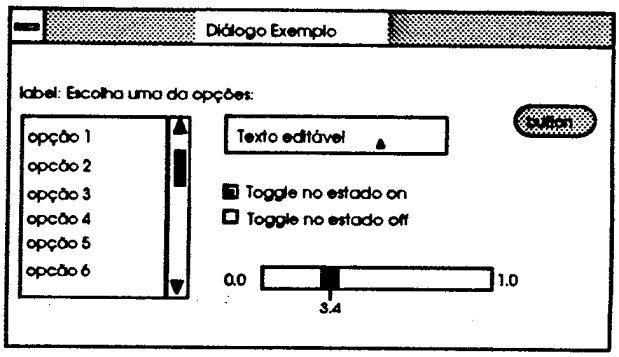
- Vantagens
  - ▷ sob medida para aplicação
  - ▷ implementação sobre sistemas gráficos
- Desvantagens
  - ▷ eficiência
  - ▷ projeto conceitual da aplicação

Portabilidade de Sistemas de Interface

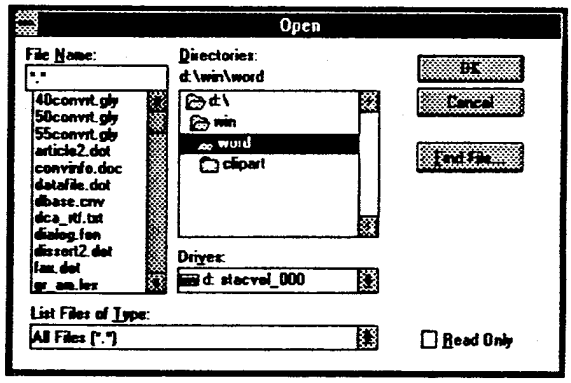
- Estratégias para aplicações gráficas
  - ▷ *look-and-feel* nativo
  - ▷ fluxo de informação bidirecional
  - ▷ arranjo bidimensional
- Estratégia: denominador comum



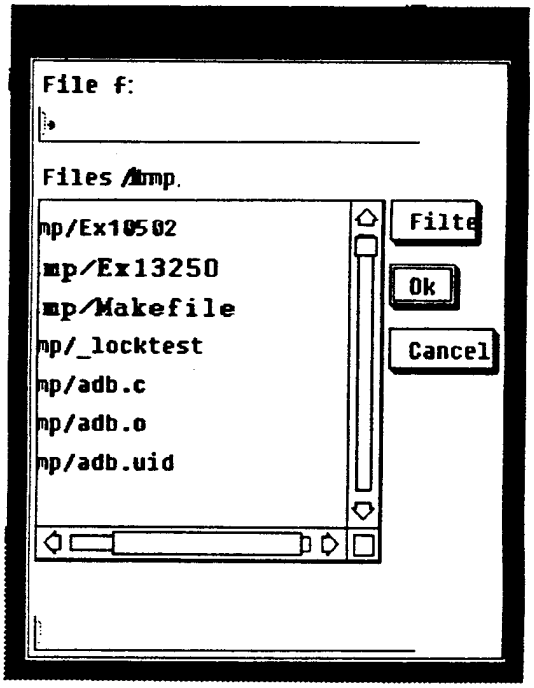
- Vantagens
  - ▷ eficiência
- Desvantagens
  - ▷ interfaces pobres



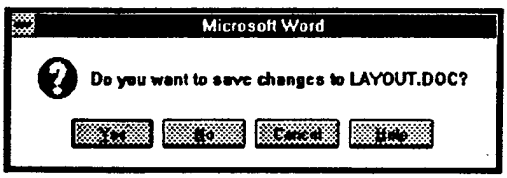
elementos de interfaces: *button, toggle, label, text, list, valuator*



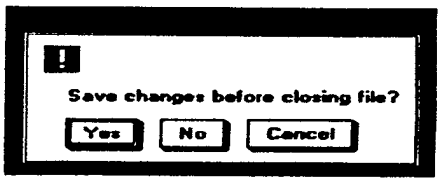
file selection box do Windows



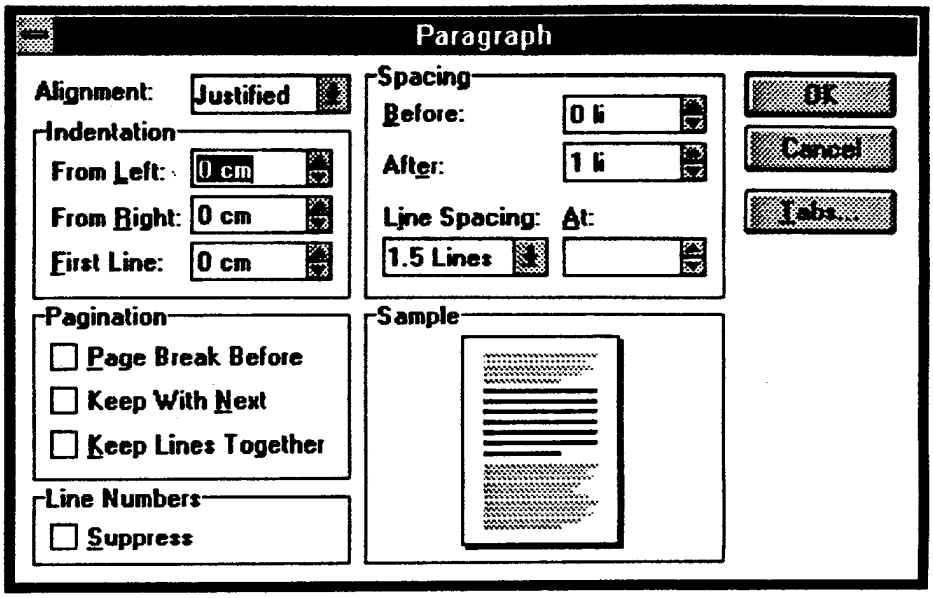
file selection box do Motif



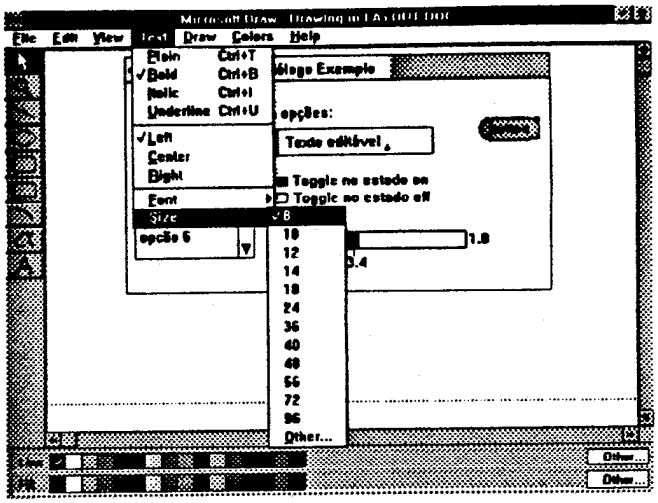
message do Windows



message do Motif



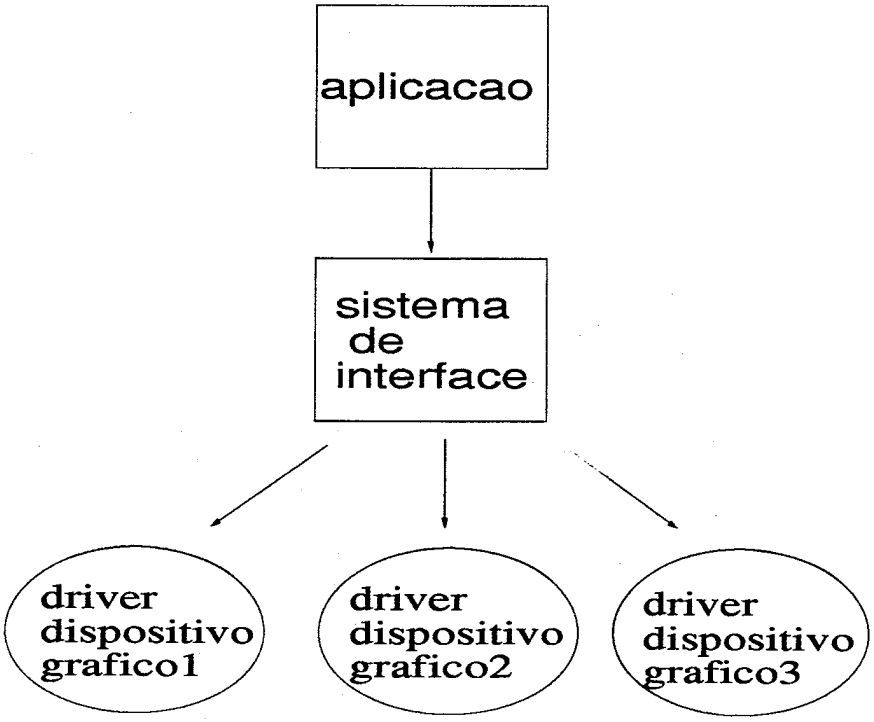
elemento de interface *dialog* do MS-Windows



elemento de interface *menu* do MS-Windows

Portabilidade de Sistemas de Interface

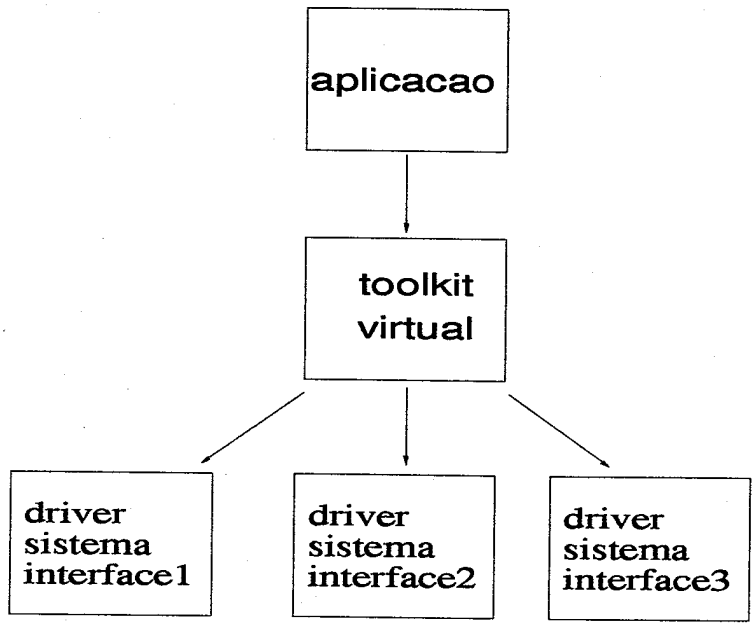
- Estratégia: portar um sistema de interface



- Vantagens
  - ▷ *look-and-feel* fixo
- Desvantagens
  - ▷ *look-and-feel* nativo
  - ▷ eficiência

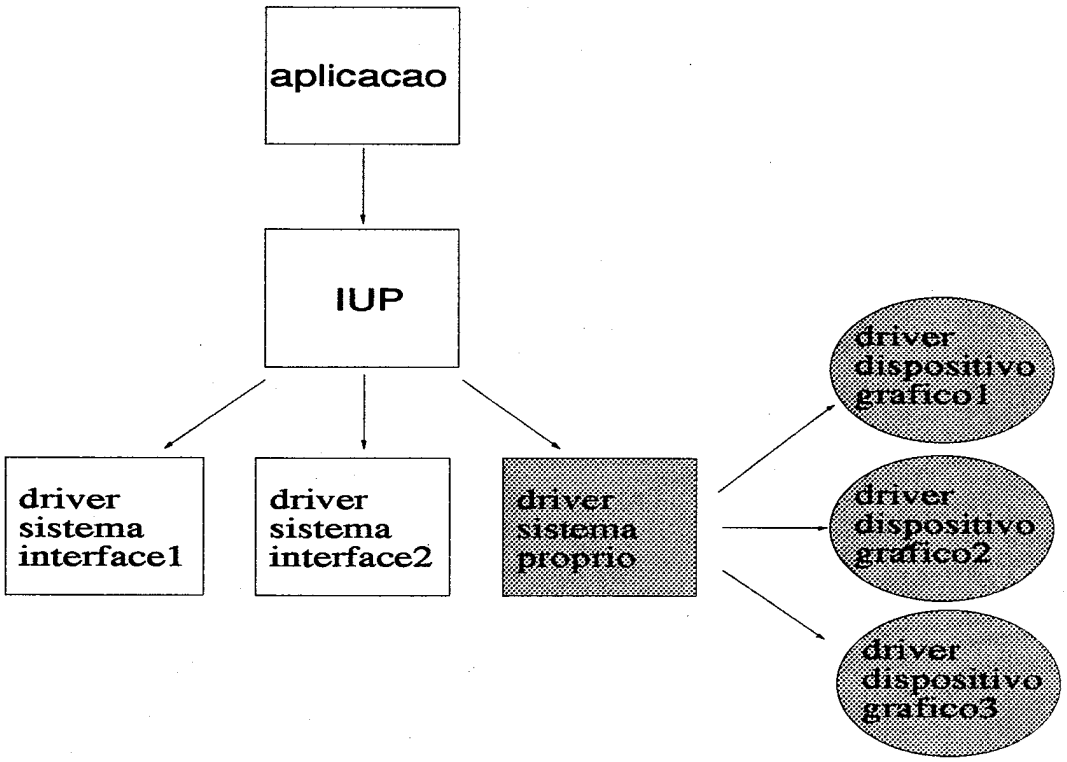
Portabilidade de Sistemas de Interface

- Estratégia: portar um *toolkit* virtual

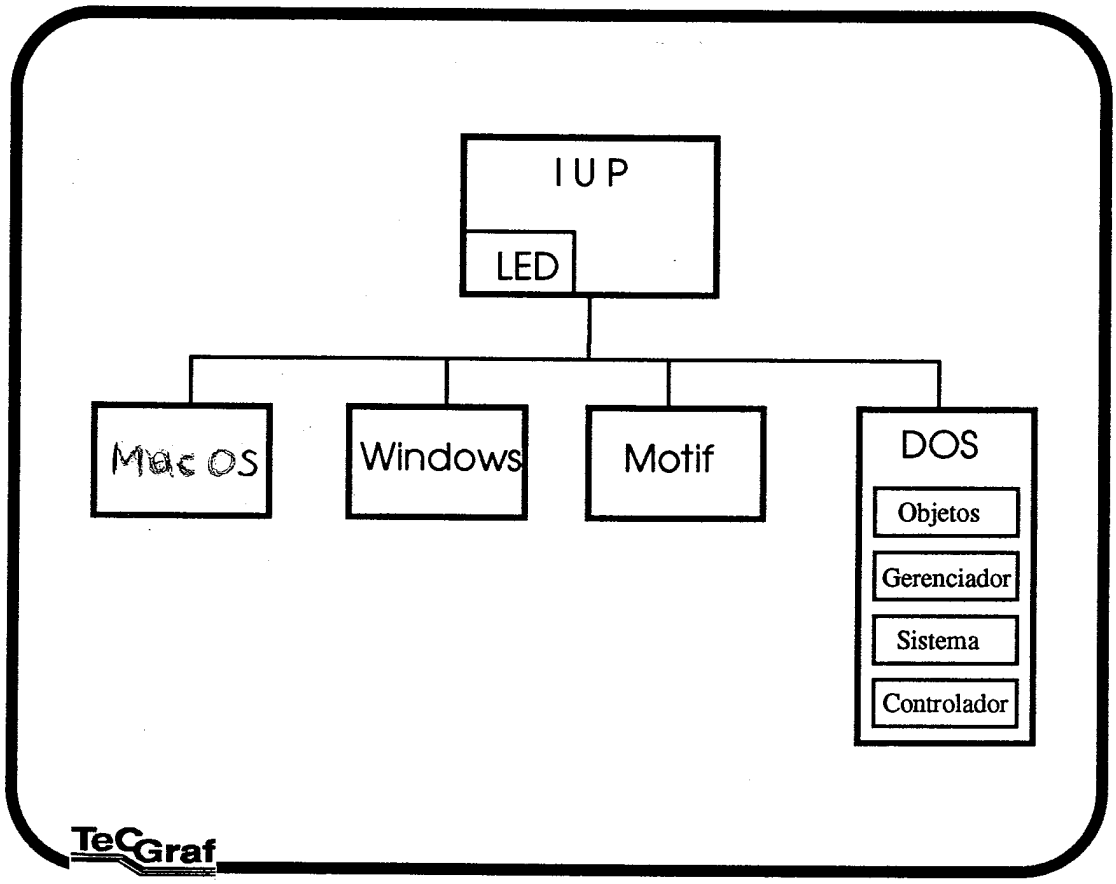


- Vantagens
  - ▷ *look-and-feel* nativo
  - ▷ eficiência
- Desvantagens
  - ▷ *look-and-feel* fixo
  - ▷ como projetar *toolkit* virtual?

Arquitetura do IUP/LED



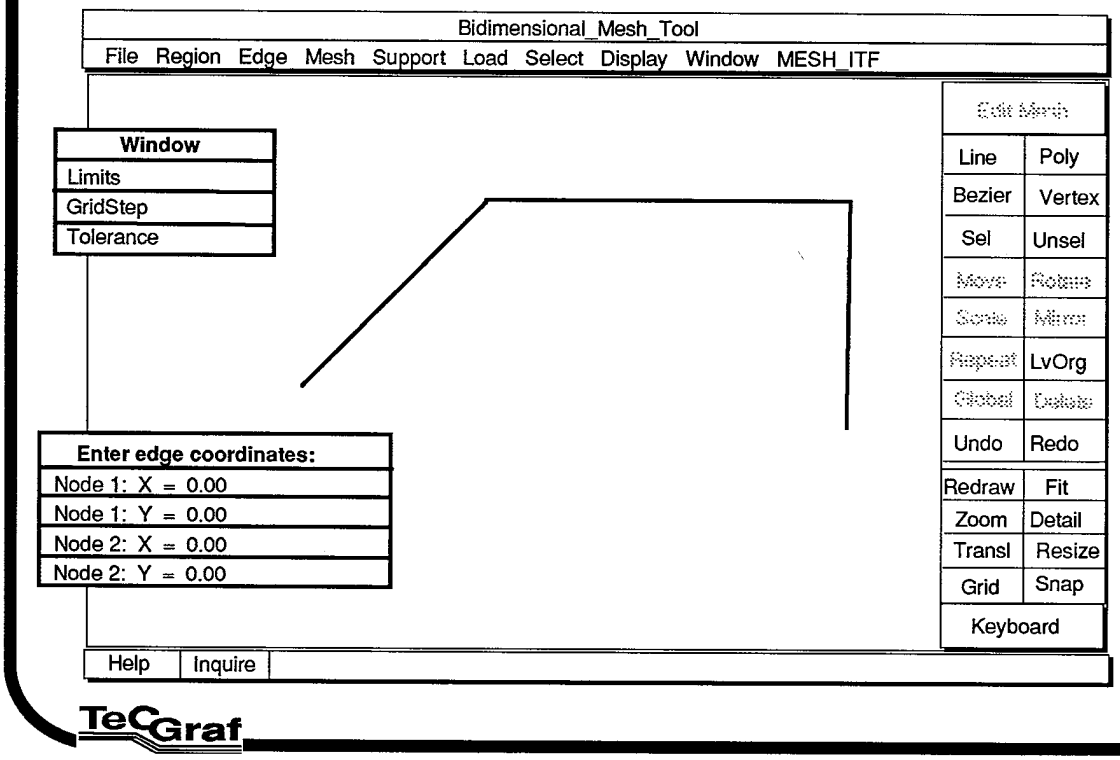
- Vantagens
  - ▷ *look-and-feel* nativo e fixo
  - ▷ facilmente portado
- Desvantagens
  - ▷ desenvolvimento de um *toolkit* próprio


IUP
LED
Mac OS
Windows
Motif
DOS
Objetos
Gerenciador
Sistema
Controlador
TeCGraf

Modelo do IUP/LED

- aplicação = conjunto de diálogos
- diálogos = hierarquia de elementos de interface
- especificação de *layout*
  - ▷ concreto × abstrato
- atributos definem aparência
- linguagem de especificação de diálogos: LED
- *toolkit* virtual: IUP

Aplicação IUP/LED



TeCGraf

Diálogos

. Definidos pelos Usuários (em LED)
. Concorrentes ou Não
. Posição e Tamanho Variáveis
. Estruturas Hierárquicas de Elementos de Interface
. Compilados em Tempo de Execução

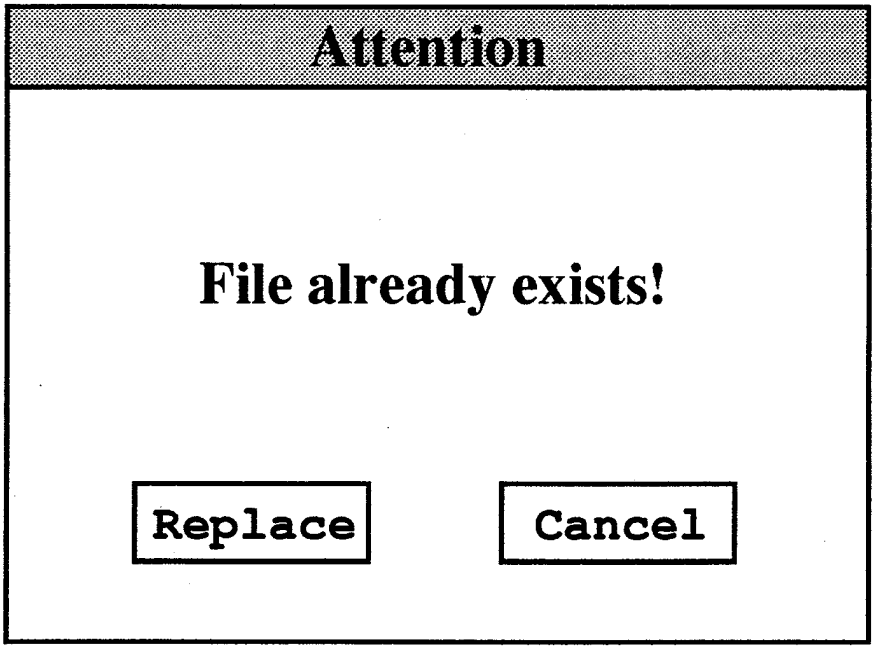
Elementos de Interface

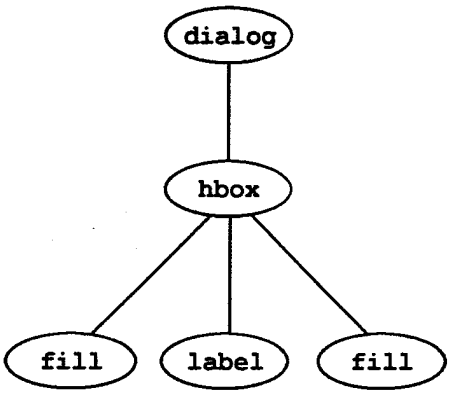
- Primitivos
  - ▷ `button`, `canvas`, `frame`, `hotkeys`
  - ▷ `image`, `itens`, `labels`, `list`
  - ▷ `submenu`, `text`, `toggle`, `valuator`
- Agrupamento
  - ▷ `dialog`
  - ▷ `radio`
  - ▷ `menu`
- Composição
  - ▷ `hbox`
  - ▷ `vbox`
- Preenchimento
  - ▷ `fill`

Modelo do *Layout* Abstrato

- Paradigma *boxes-and-glue* do TEX
  - ▷ simples ⇒ rápido aprendizado

- *Layout* concreto automaticamente calculado
  - ▷ mantém o *layout* abstrato

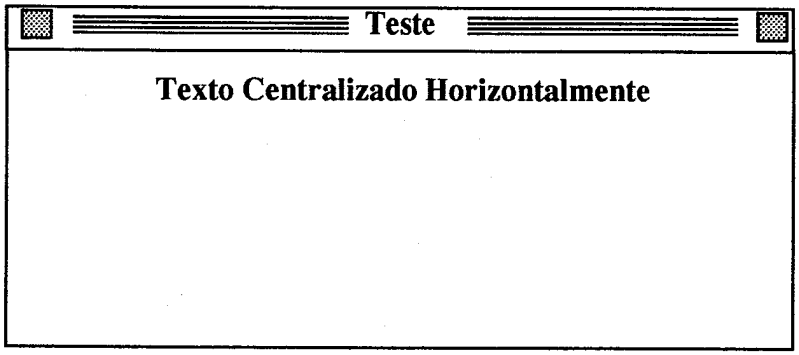



dialog
hbox
fill
label
fill

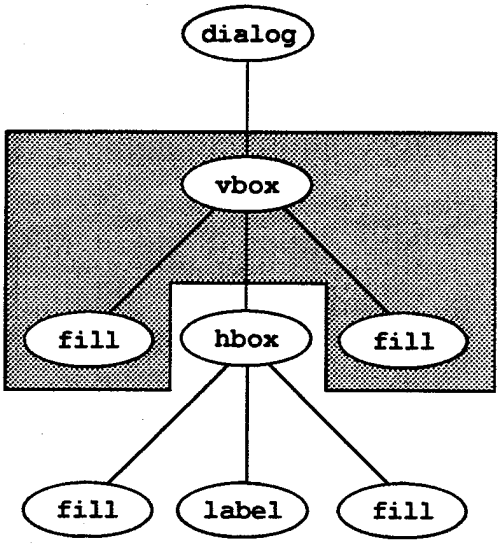
(a)


Teste
Texto Centralizado Horizontalmente

(b)


Teste
Texto Centralizado Horizontalmente

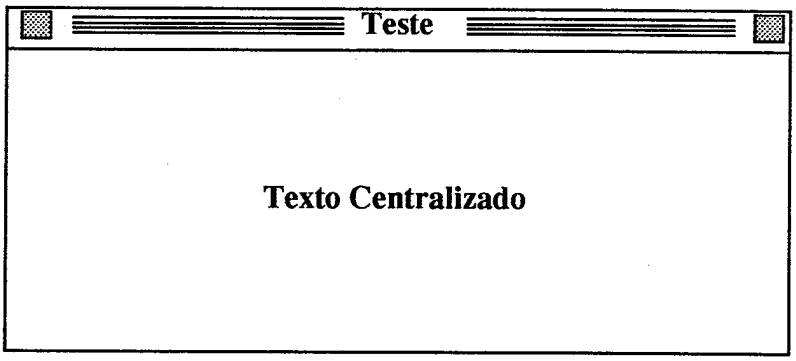
(c)


dialog
vbox
fill
hbox
fill
fill
label
fill

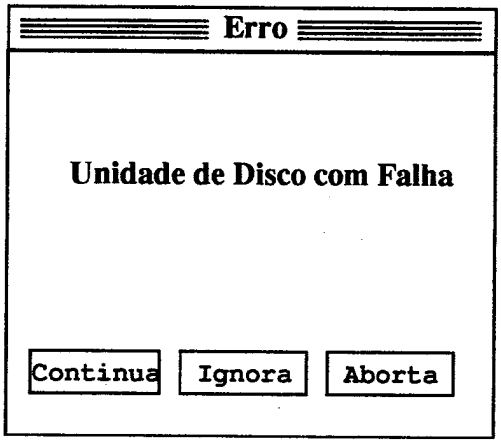
(a)


Teste
Texto Centralizado

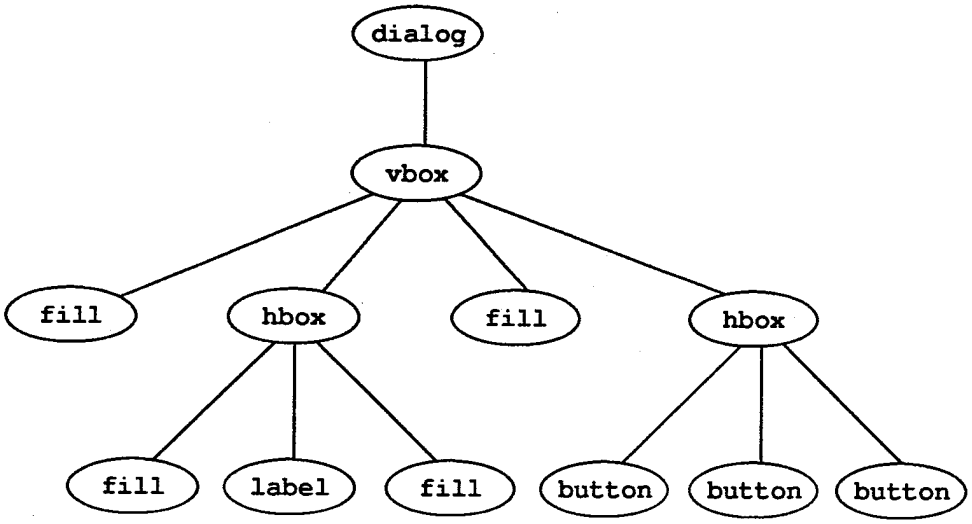
(b)


Teste
Texto Centralizado

(c)

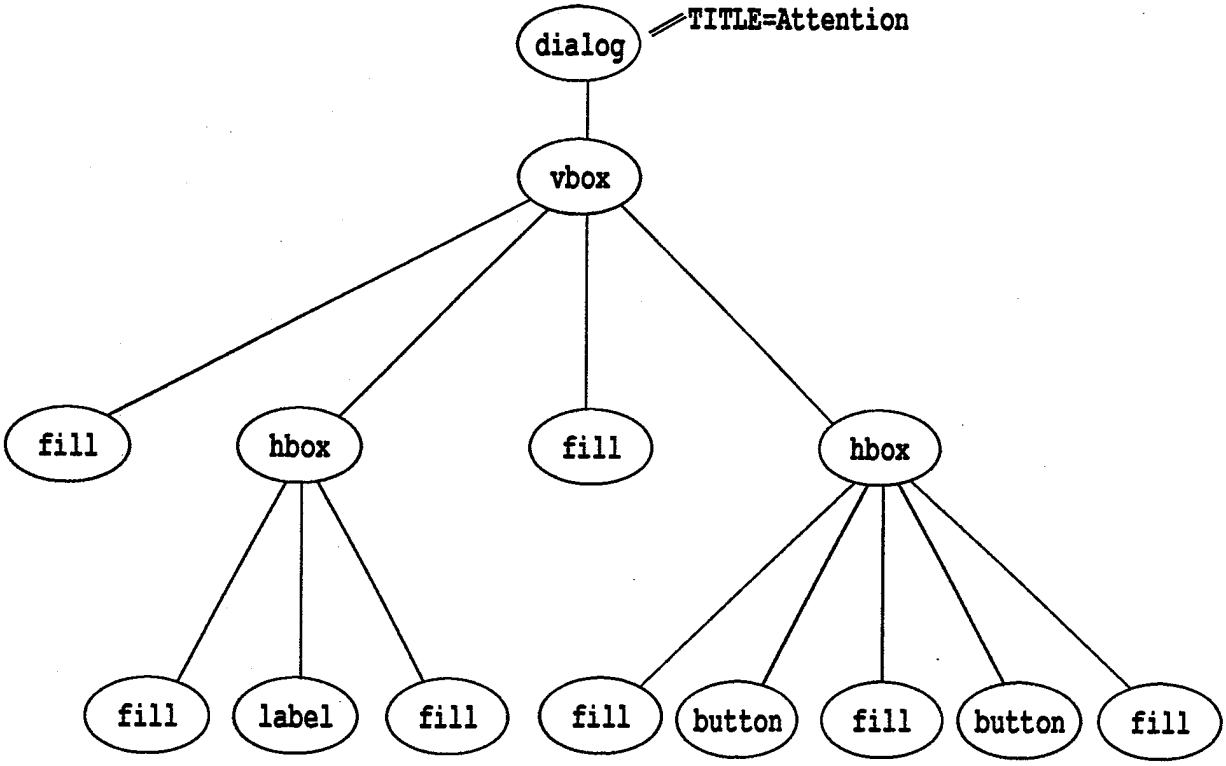

Erro
Unidade de Disco com Falha
Continua
Ignora
Aborta

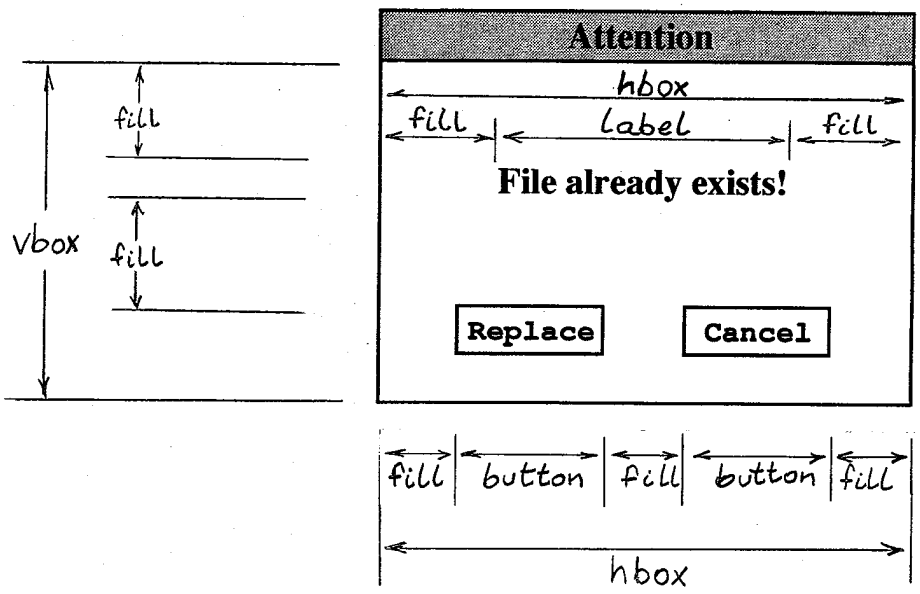
(a)


dialog
vbox
fill
hbox
fill
hbox
fill
label
fill
button
button
button

(b)

Estrutura Hierárquica do Exemplo





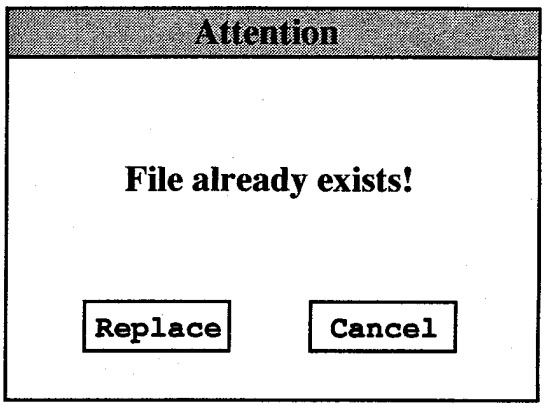
LED

- Linguagem de expressões
  - ▷ sintaxe simples

$$v = f[a](p)$$

- Exemplo (arquivo: attention.led)

```
replace=button("Replace",do_replace)
cancel=button("Cancel",do_cancel)

buttons=hbox(fill(),replace,fill(),cancel,fill())

warning=label("File already exists!")
prompt=hbox(fill(),warning,fill())

body=vbox(fill(),prompt,fill(),buttons)
confirm=dialog[TITLE="Attention"](body)
```



LED com elementos anônimos

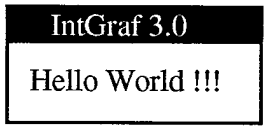
```
confirm=dialog[TITLE="Attention"](
    vbox(
        fill(),
        hbox(
            fill(),
            label("File already exists!"),
            fill()
            ),
        fill(),
        hbox(
            fill(),
            button("Replace",do_replace),
            fill(),
            button("Cancel",do_cancel),
            fill()
            )
    )
)
```

Código IUP para o Exemplo

```
#include "iup.h"

void main (void)
{
 Ihandle *confirm;
                          identificador do diálogo
 IupOpen ();

 IupSetFunction ("do_cancel",fcancel);
 IupSetFunction ("do_replace",freplace);
                        registra funções das ações

 IupLoad ("attention.led");
                        compila especificação LED

 confirm = IupGetHandle ("confirm");
 IupShow (confirm);
                                    exibe diálogo
 IupMainLoop ();
                              interage com usuário
 IupClose ();
}
```

HW - Menor Aplicação IUP/LED

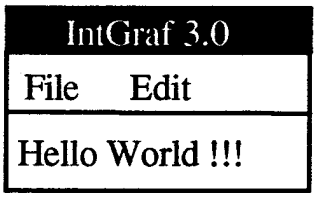
```
hw.led
    hw = dialog [TITLE="IntGraf 3.0"]
                (label("Hello World !!!"))
```




```
#include "iup.h"
void main (void)
{
  Ihandle *d;
  IupLoad ("hw.led");
  d = IupGetHandle ("hw");
  IupShow (d);
  IupMainLoop ( );
}
```

TeCGraf

HW com Menu de Barra




```
hw.led
    hw = dialog [title="IntGraf 3.0",
                 menu = MainMenu]
                (label("Hello World !!!"))
```

TeCGraf

HW com menu de barra

```
MainMenu = menu (
        submenu("File", menu (
                        item("New",  fnew),
                        item("Open", fopen),
                        item("Save", fsave),
                        item("Exit",  fexit))),
        submenu("Edit", menu (
                        item("Copy", null),
                        item("Cut",   null),
                        item("Paste", null)))
```

HW com Menu de Barra

```
open = item ("Open", fopen)
save = item ("Save", fsave)

EditMenu = menu (item("Copy",null),
                 item("Cut",null),
                 item("Paste",null))
FileMenu = menu (open,
                 save,
            exit=item("Exit",fexit))
MainMenu = menu (
      submenu ("File", FileMenu),
      submenu ("Edit", EditMenu))
```

```
============================================================
Arquivo main.led
============================================================

bar = menu(
 projeto = submenu( "Projeto",
  menu(
   item( "Info", ProjectInfo ),
   item( "Fim", Exit )
  )
 ),
 processa = submenu( "Processa",
  menu(
   item( "Desenho", Display )
  )
 )
)

main = dialog[TITLE="Primeiro Trabalho Grafico",
              SIZE=400x250,
              MENU=bar,
              FGCOLOR="0 0 0"](
 datascreen = canvas[RESIZE_CB=Resize]( Redraw )
)


============================================================
Arquivo main.c
============================================================
Ihandle* Omain;
Ihandle* Odatascreen;

int main( int argc, char* argv[] )
{
 GraOpen( );
 IupOpen( );

 IupSetFunction( "Exit",        Exit );
 IupSetFunction( "ProjectInfo", ProjectInfo );
 IupSetFunction( "Resize",      Resize );
 IupSetFunction( "Redraw",      Redraw );
 IupSetFunction( "Display",     Redisplay );

 IupLoad( "main.led" );

 Omain = IupGetHandle( "main" );
 Odatascreen = IupGetHandle( "datascreen" );

 IupShow( Omain );

 ProjectInit( );

 IupMainLoop( );

 IupClose( );
 return( 0 );
}
```

IUP

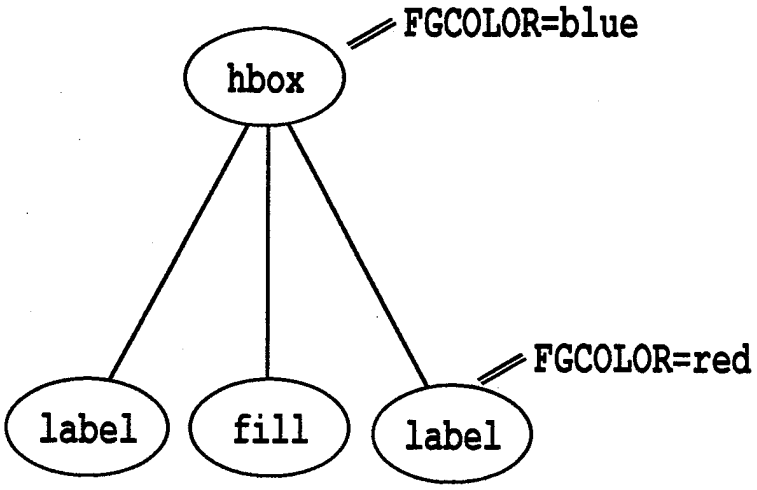
- Biblioteca de funções implementadas em C
  - ▷ $\approx$ 30 funções

- Principais serviços
  - ▷ compilar LED
  - ▷ criar e manipular elementos de interface
  - ▷ registrar funções correspondentes às ações
  - ▷ associar nomes aos elementos
  - ▷ associar atributos aos elementos

Atributos

- Principalmente para aparência

- Mecanismo de herança



- *Fine-tuning*

  ▷ uma só especificação LED

- Aplicações podem criar atributos próprios

  ▷ evita variáveis globais