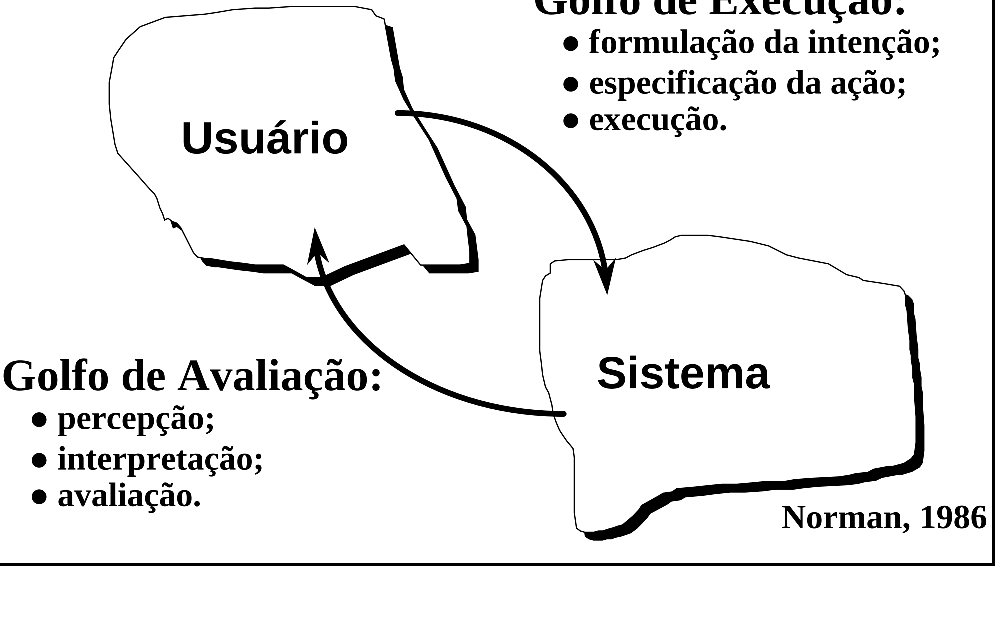
Sistemas de Interfaces com o Usuário

Processo de Interação (Eng. Cognitiva)



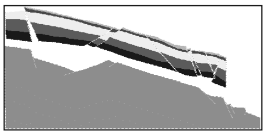
Golfo de Execução:
- **formulação da intenção;**
- **especificação da ação;**
- **execução.**

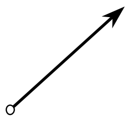
Golfo de Avaliação:
- **percepção;**
- **interpretação;**
- **avaliação.**

Norman, 1986

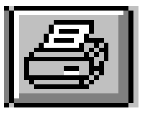
Signos (Semiótica)

- ***Índices***





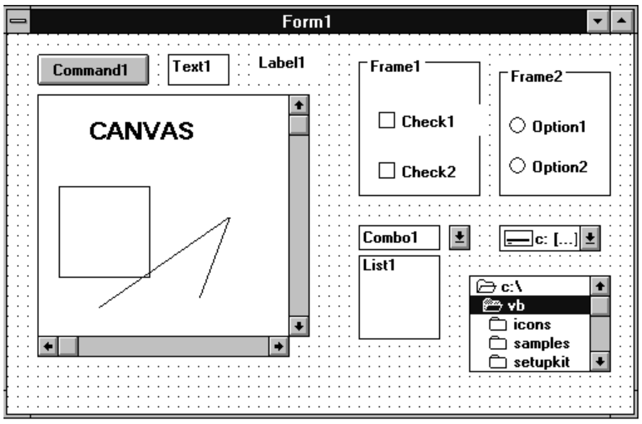
- ***Ícones***



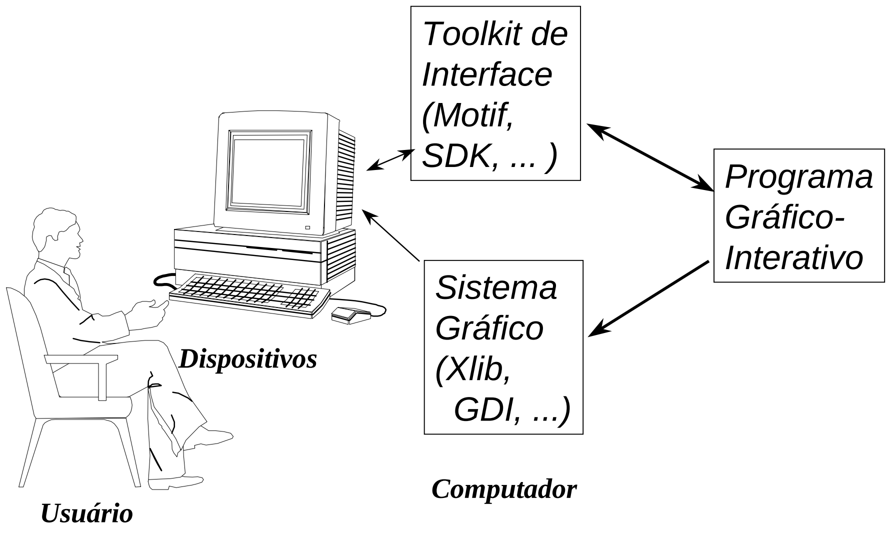
- ***Símbolos***

A

Objetos de comuns interface



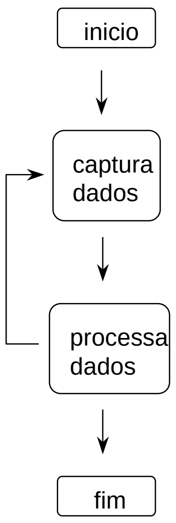
Modelo de Programação


Toolkit de Interface (Motif, SDK, ...)
Programa Gráfico-Interativo
Sistema Gráfico (Xlib, GDI, ...)
Dispositivos
Computador
Usuário

Programação Convencional

Programação Convencional

Os comandos são executados segundo uma ordem pré-estabelecida e seqüencial.



Técnicas de Interação

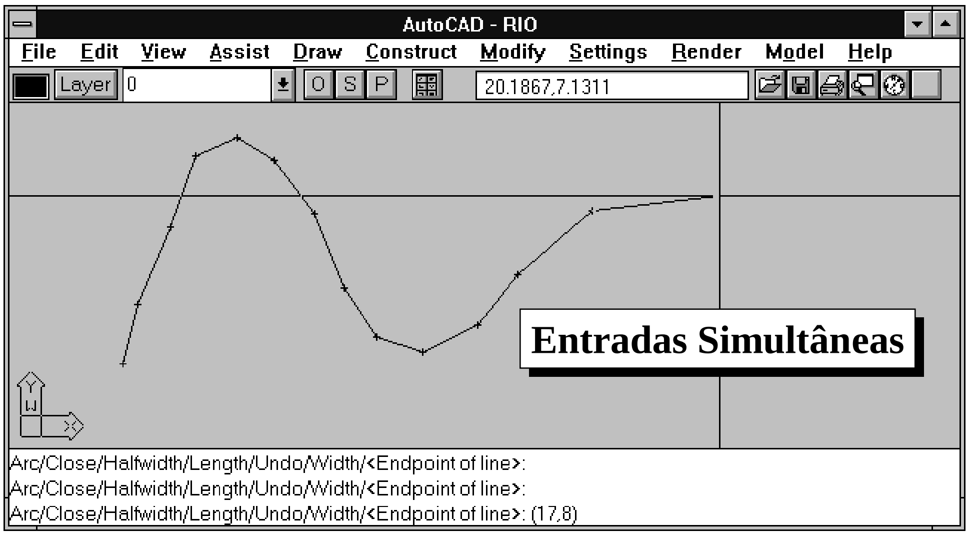
- **Solicitação (*Request*)**
- **Amostragem (*Sample*)**
- **Eventos (*Event*)**
  - **eventos [Xlib, SDK]**
  - ***callbacks* [*Motif, IUP, Visual…*]**
  - ***listeners* [*Java/OO*]**

Exemplo de Interação no Modo de Solicitação (*Request*)

```
{
 Gqloc  lc;
 static Gdlimit area = {0.0, 0.1, 0.0, 0.1};
 static Glocrec rec = {{NULL}};
 static Gloc      init = {0, 0.5, 0.5};
     ...
 Message ("Defina um ponto na tela ( ESC cancela )");
 do  {
   ginitloc (1, 1, &init, 3, &area, &rec);
   lc = greqloc (1, 1);
   if (lc.status == NONE)
      return;
  } while (lc.loc.transform != 1);

/* trata as coordenadas utilizando a estrutura lc ( lc.loc.position )              */
       ...
}
```

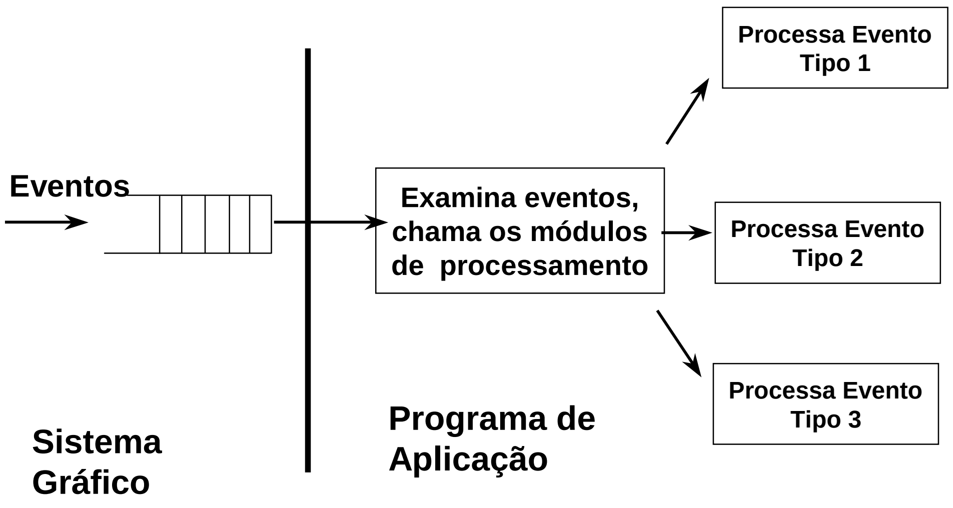
Entrada de um ponto da tela

Problemas com solicitação


AutoCAD - RIO
File Edit View Assist Draw Construct Modify Settings Render Model Help
Layer 0
O S P
20.1867,7.1311
Entradas Simultâneas

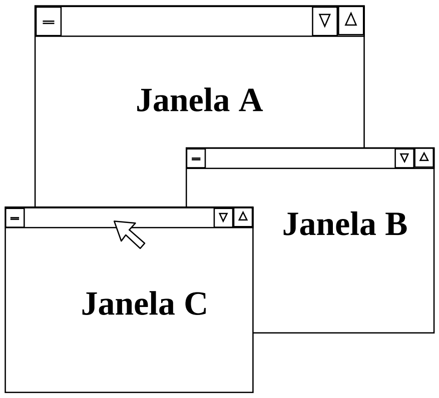
```
Arc/Close/Halfwidth/Length/Undo/Width/<Endpoint of line>:
Arc/Close/Halfwidth/Length/Undo/Width/<Endpoint of line>:
Arc/Close/Halfwidth/Length/Undo/Width/<Endpoint of line>: (17,8)
```

Modelo de Eventos



Eventos típicos (WIMP)

KeyPress
KeyRelease
ButtonPress
ButtonRelease
Motion
LeaveNotify
EnterNotify
WindowExposure
Resize
Timer
Idle



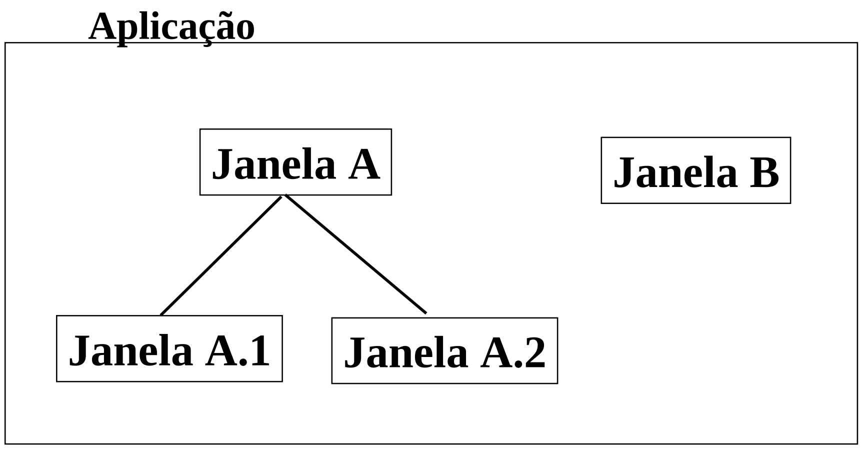
```
XEvent     report;

/* Select event types wanted */
XSelectInput(display, win,  ExposureMask | KeyPressMask | StructureNotifyMask);

while (1)  {                /* get events, use first to display text and graphics */
            XNextEvent(display, &report);
            switch  (report.type) {
                        case Expose:
                                    ...
                                    break;
                        case ButtonPress:
                                    ...
                                    exit(1);
                        case ConfigureNotify:
                                    ...
                                    break;
                        default:
                                    break;
            } /* end switch */
} /* end while */
```

Xlib/ X Window

```
static int nextevent(EventRecord* theEvent, int* x, int* y)
{
 while (1)
  {
   if   (!WaitNextEvent (everyEvent,theEvent,0,0L))
         theEvent->what=idleEvent;

   switch (theEvent->what)
    {
     case idleEvent:      doIdle( );  return 1;
     case keyDown:        ...         return 1 ;
     case mouseDown:  ...             return 1;
     default:                         return 1;
  }
 }
}
```

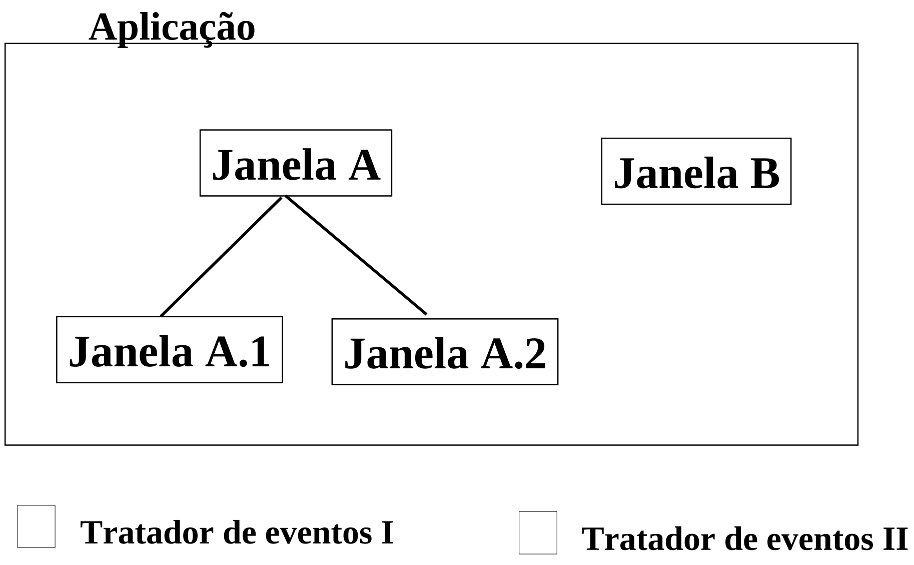
QuickDraw /Macintosh

Janelas e tratadores de eventos



- Geralmente um tratador de eventos por aplicação

Janelas e tratadores de eventos



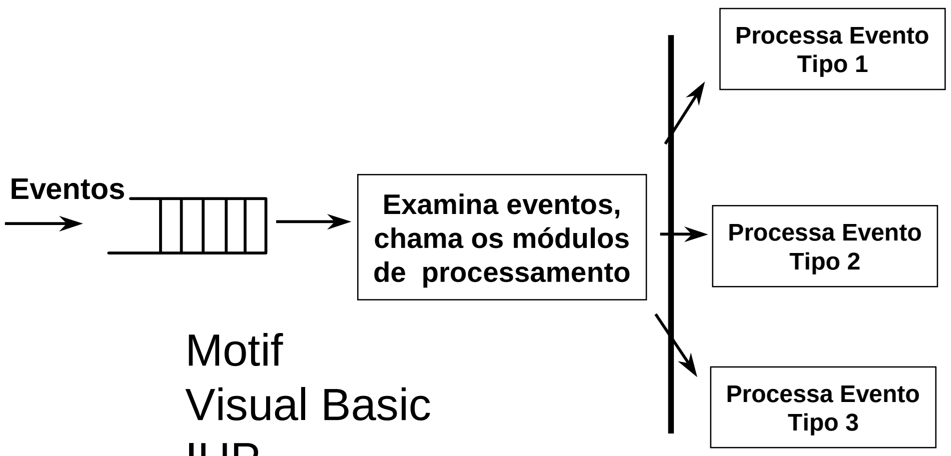
```
int PASCAL WinMain (HANDLE hCopia, HANDLE hCopiaAnterior,
                            LPSTR  lpszParamCmd, int nCmdMostrar)
{
  if (!hCopiaAnterior)  {
        classejan.lpfnWndProc   = ProcJan ;
        classejan.lpszClassName = szNomeAplic ;
        . . .
        RegisterClass (&classejan) ;
        }

 hjan = CreateWindow (szNomeAplic,... ) ;

 ShowWindow (hjan, nCmdMostrar) ;
 UpdateWindow (hjan) ;

   while (GetMessage (&msg, NULL, 0, 0))
      {
        TranslateMessage (&msg) ;
         DispatchMessage (&msg) ;
       }
     return msg.wParam ;
}
```

MS Windows SDK

MS Windows (cont.)

```
long FAR PASCAL _export  ProcJan ( HWND hjan,
        UINT mensagem, UINT wParam, LONG lParam)
 {
  . . .
  switch (mensagem)
    {
    case WM_PAINT:
      hdc = BeginPaint (hjan, &ps) ;
      DrawText (hdc, "Ola', Windows!", ... ) ;
      EndPaint (hjan, &ps) ;
      return 0 ;

     case WM_DESTROY:
       PostQuitMessage (0) ;
       return 0 ;
     }
   ...
}
```

Modelo de *Call Backs*



Motif
Visual Basic
IUP
...

Programa de Aplicação

Motif

```
static void repaint(Widget widget, char* client,
                      XmDrawingAreaCallbackStruct *data) { ...}
void main(int argc, char *argv[])
{
  static char                *vec[]    = {"canvas.uid"};
  static MrmRegisterArg regvec[] = {  {"a_repaint",(caddr_t)repaint}   };

  MrmInitialize ();                                             // init UIL
  toplevel = XtAppInitialize(NULL, "hello", NULL, 0,
                              &argc, argv, NULL, NULL, 0);    // init Motif
  MrmOpenHierarchy (1, vec, NULL, &hier);                       // load arq
  MrmRegisterNames (regvec, regnum);                            // reg callbacks
  MrmFetchWidget (hier, "main", toplevel, &mainwidget, &class);
  XtManageChild(mainwidget);                                    // manage  main
  XtRealizeWidget(toplevel);                         // realize managed child

  XtAppMainLoop(XtWidgetToApplicationContext(toplevel));
}
```

UIL/Motif

```
module canvasuil

procedure  a_repaint();

object    main : XmBulletinBoard {
                  controls {   XmDrawingArea   canvas; };
     };

object    canvas : XmDrawingArea {
       arguments {   XmNx = 15;    XmNy = 60;        };
       callbacks {   XmNexposeCallback = procedure a_repaint(); };
    };

end module;
```

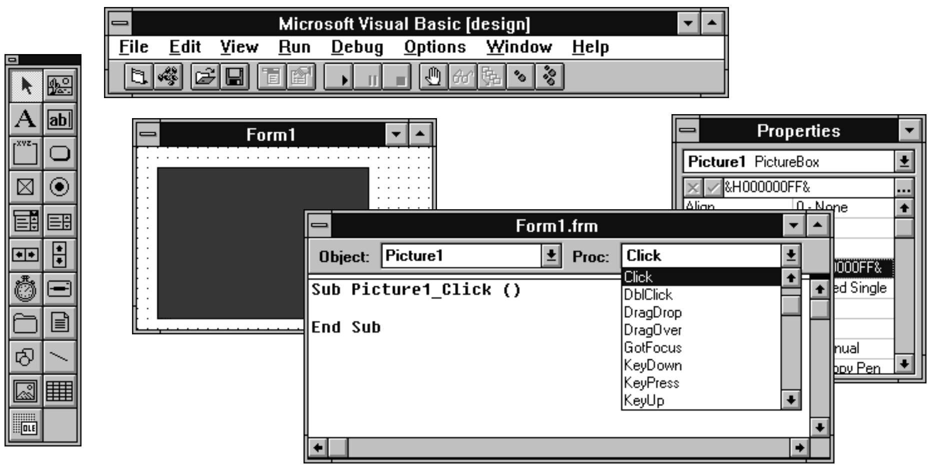
IUP/LED

```
static int repaint (Ihandle *self)
{  ...   }

void main (void)
{
  IupOpen( );
    IupLoad ("canvas.led");
    IupSetFunction ("a_repaint", (Icallback)repaint);
    IupShow (IupGetHandle("main"));
    IupMainLoop( );
 IupClose( );
}
```

canvas.led

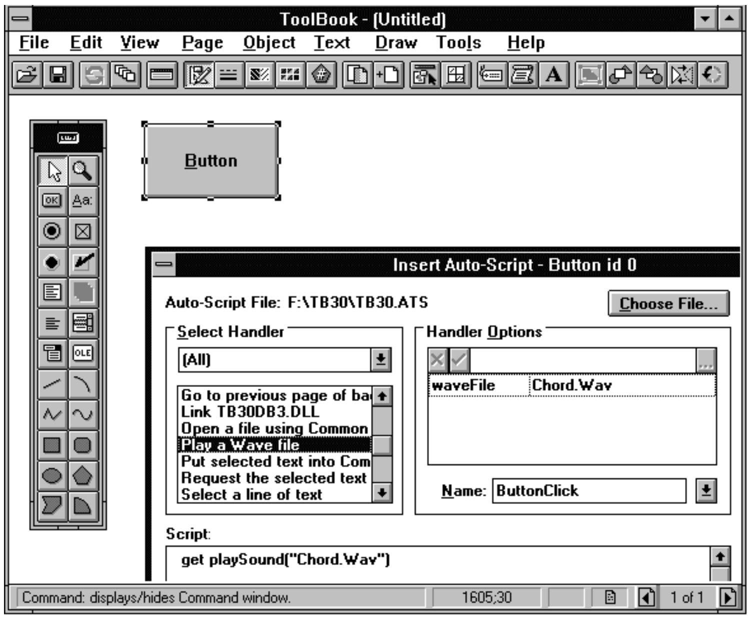
main = *dialog* [TITLE="IUP Canvas"]   (*canvas* (a_repaint))

Visual Basic



```
Sub Picture1_Click ()

End Sub
```

ToolBook


ToolBook - (Untitled)
File Edit View Page Object Text Draw Tools Help
Button
Insert Auto-Script - Button id 0
Choose File...
Select Handler
Handler Options

Auto-Script File: F:\TB30\TB30.ATS

Select Handler: (All)

```
Go to previous page of ba
Link TB30DB3.DLL
Open a file using Common
Play a Wave file
Put selected text into Com
Request the selected text
Select a line of text
```

Handler Options:

waveFile	Chord.Wav

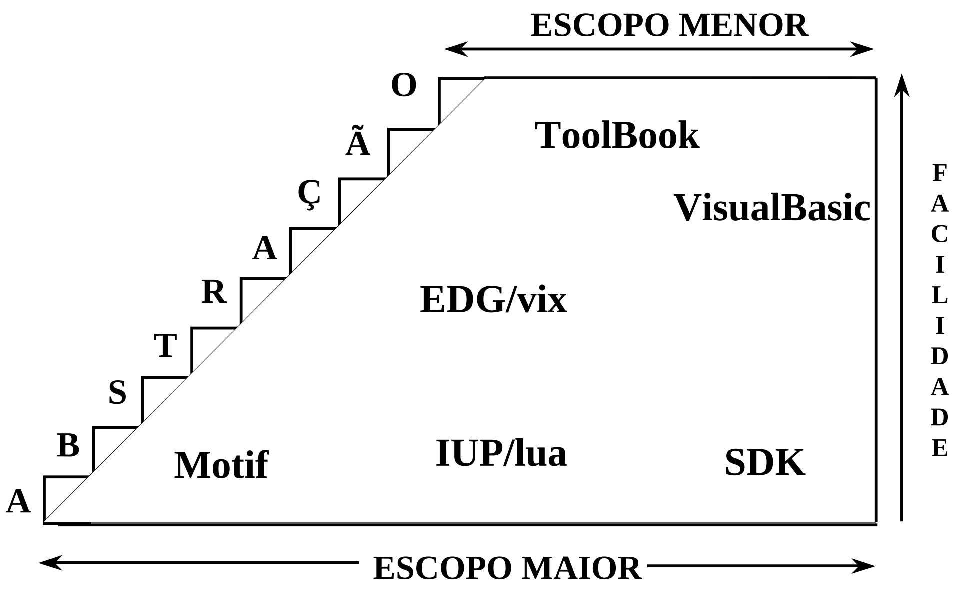
Name: ButtonClick

Script:

```
get playSound("Chord.Wav")
```

Command: displays/hides Command window. 1605;30 1 of 1

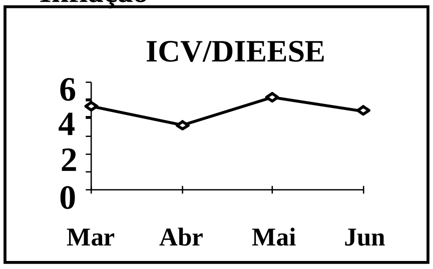
Nível de abstração e escopo



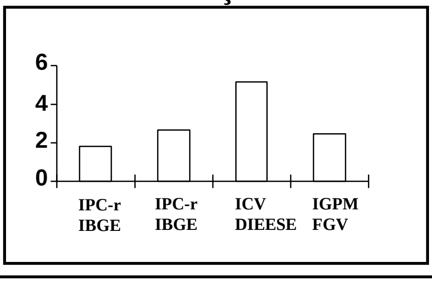
Em que usar o que?

Modelos Quantitativos

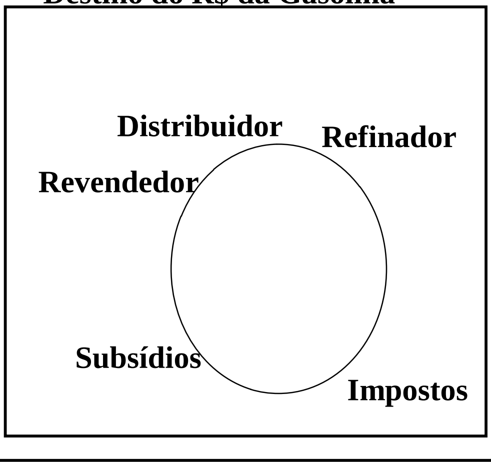
Inflação



Indices de Inflação em Jul95

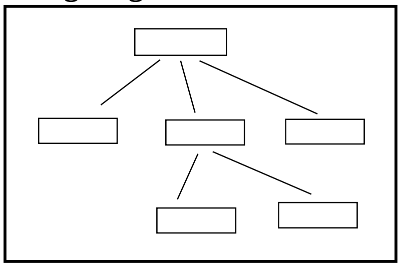


Destino do R$ da Gasolina

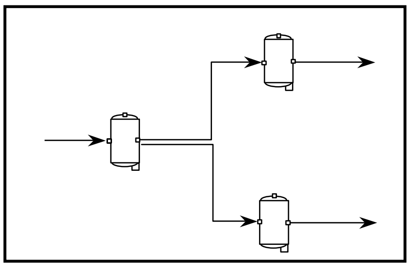


Modelos Discretos

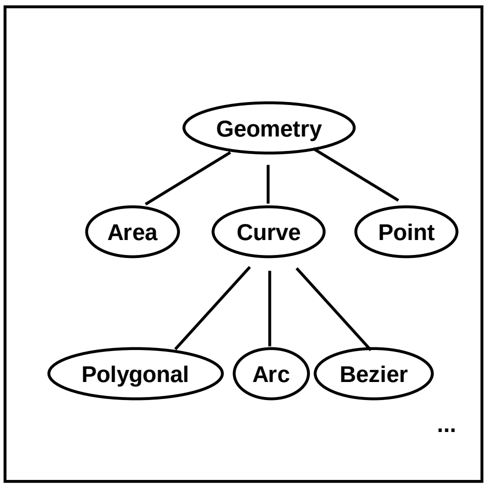
Organogramas



Petrox

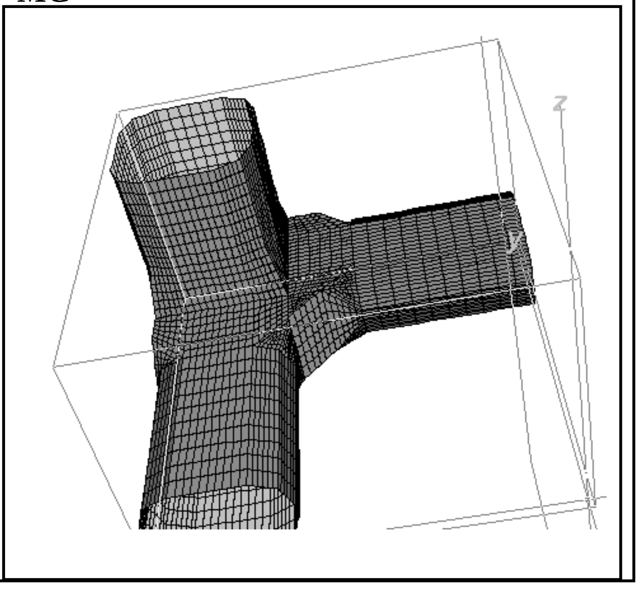


Classes de OO

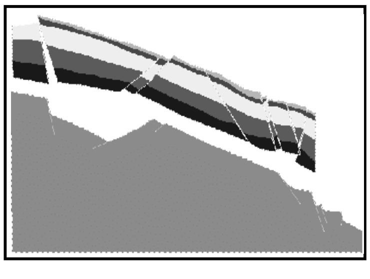


Modelos Geométricos

MG



RECON



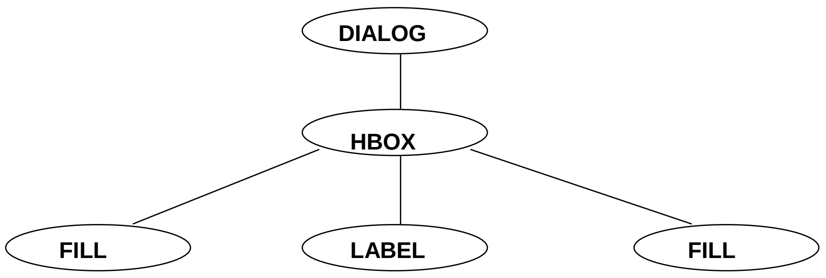
Modelo do IUP/LED

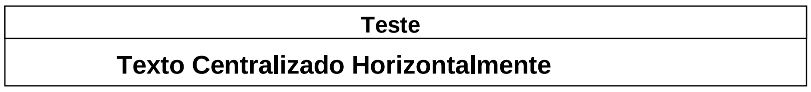
- **Aplicação = conjunto de diálogos**
- **Diálogos = hierarquia de elementos de interface**
- **Especificação de *layout***
  - **Concreto X Abstrato**
- **Atributos definem a aparência**

Elementos de Interface

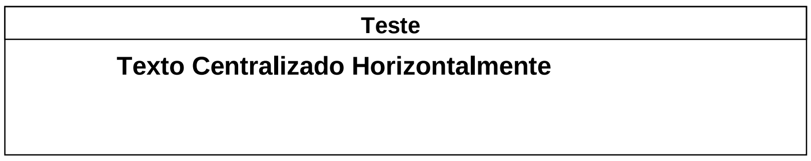
- **Primitivos**
  - **Button, Canvas, Frame, Image, Item, Label, List, Submenu, Text, Toggle,Matrix,Multiline**
- **Agrupamento**
  - **Dialog, Radio, Menu**
- **Composição**
  - **Hbox, Vbox, Zbox**
- **Preenchimento**
  - **Fill**

Composição do *layout*



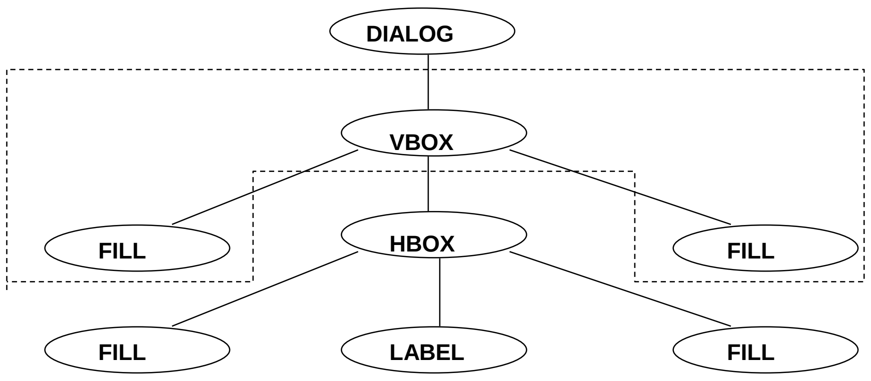

Teste

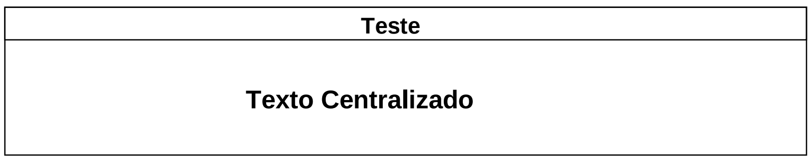
Texto Centralizado Horizontalmente


Teste

Texto Centralizado Horizontalmente

Centralizando elementos

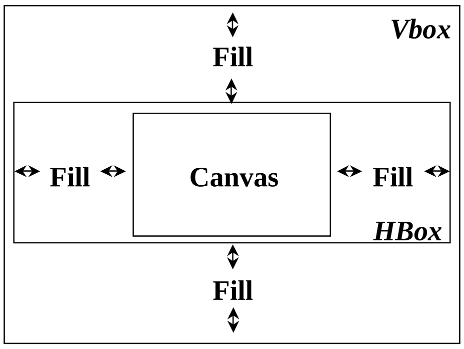



Teste

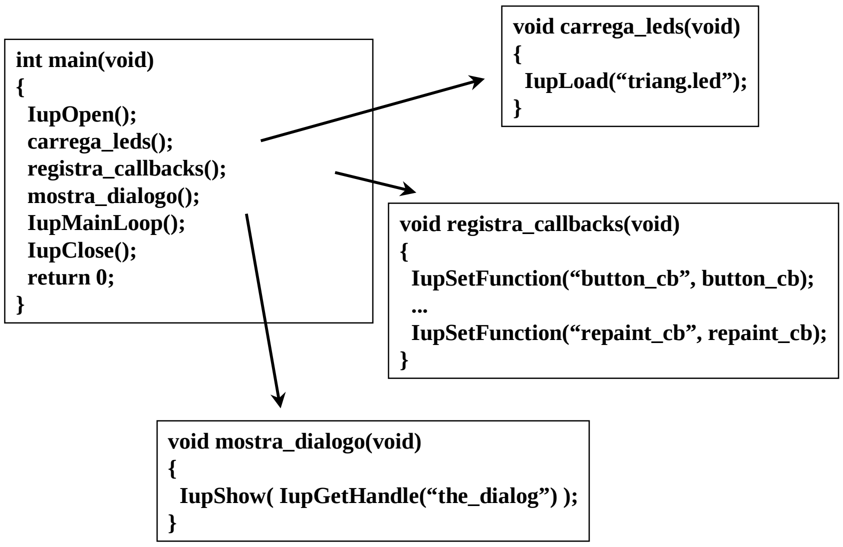
Texto Centralizado

Layout abstrato

```
the_menu = ...

the_canvas = CANVAS[ BUTTON_CB = button_cb,
           MOTION_CB = motion_cb](repaint_cb)

the_dialog = DIALOG[ MENU=the_menu ]
(
 VBOX(
  FILL(),
  HBOX(
   FILL(),
   the_canvas,
   FILL()
  ),
  FILL()
 )
)
```



Esqueleto da aplicação



```
int main(void)
{
  IupOpen();
  carrega_leds();
  registra_callbacks();
  mostra_dialogo();
  IupMainLoop();
  IupClose();
  return 0;
}
```

```
void carrega_leds(void)
{
  IupLoad("triang.led");
}
```

```
void registra_callbacks(void)
{
  IupSetFunction("button_cb", button_cb);
  ...
  IupSetFunction("repaint_cb", repaint_cb);
}
```

```
void mostra_dialogo(void)
{
  IupShow( IupGetHandle("the_dialog") );
}
```

OpenGL/GLUT

```
#include <glut.h>

void main(int argc, char** argv)
{

/* Standard GLUT initialization */

    glutInit(&argc,argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);    /* default, not needed */
    glutInitWindowSize(500,500);                      /* 500 x 500 pixel window */
    glutInitWindowPosition(0,0);              /* place window top left on display */
    glutCreateWindow("Sierpinski Gasket");    /* window title */
    glutDisplayFunc(display);  /* display callback invoked when window opened */

      myinit(); /* set attributes */

    glutMainLoop(); /* enter event loop */
}
```

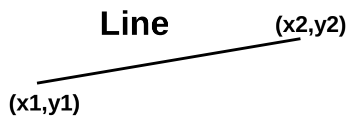
Exemplo simples da GLUT

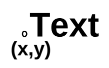
```
void myinit(void)
{
 /* attributes */
  glClearColor(1.0, 1.0, 1.0, 1.0);          /* white background */

/* set up viewing w/ 500 x 500 window with origin lower left */
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(0.0, 500.0, 0.0, 500.0);
   glMatrixMode(GL_MODELVIEW);
}
```

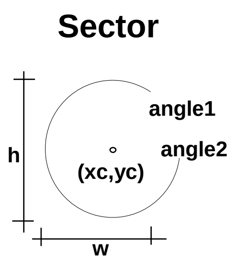
```
void display(void)
{
  glClear(GL_COLOR_BUFFER_BIT);  /*clear the window */
  glColor3f(1.0, 0.0, 0.0);                    /* draw in red */
  glRectf(0.0,0.0, 500.0, 500.0);
}
```

Primitivas do *CanvasDraw*

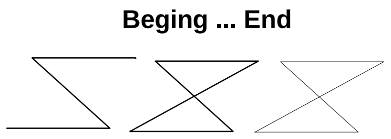
void **cdLine** (int x1, int y1, int x2, int y2);
void **cdBox** (int xmin, int xmax, int ymin, int ymax);
void **cdArc** (int xc, int yc, int w, int h, double angle1, double angle2);
void **cdSector** (int xc, int yc, int w, int h, double angle1, double angle2);
void **cdText** (int x, int y, char *s);
void **cdMark** (int x, int y);





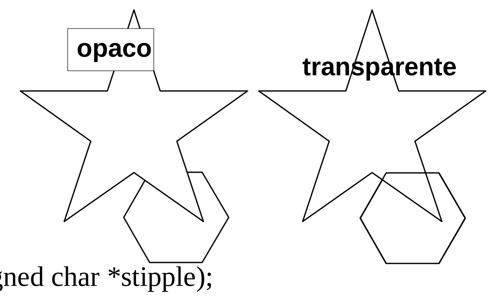


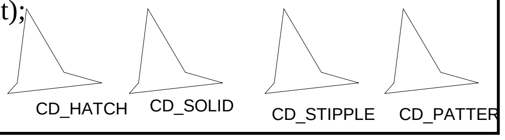
void **cdBegin** (int mode);
   void **cdVertex** (int x, int y);
void **cdEnd** (void);



Atributos do *CanvasDraw*

int **cdBackOpacity** (int opacity);
int **cdWriteMode** (int mode);
int **cdLineStyle** (int style);
int **cdLineWidth** (int width);
int **cdInteriorStyle** (int style);
void **cdHatch** (int style);
void **cdStipple** (int n, int m, unsigned char *stipple);
void **cdPattern** (int n, int m, long int *pattern);
void **cdFont** (int type_face, int style, int size);
void **cdFontDim** (int *max_width, int *height, int *ascent, int *descent);
void **cdTextSize** (char *s, int *width, int *height);
int **cdTextAlignment** (int alignment);
int **cdMarkType** (int type);
int **cdMarkSize** (int size);

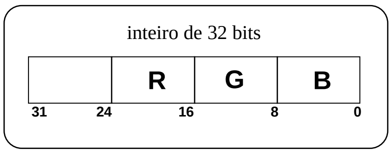




Codificação de cor no *CanvasDraw*

long int **cdEncodeColor** (unsigned char red, unsigned char green, unsigned char blue);
void **cdDecodeColor** (long int color,
unsigned char *red, unsigned char *green, unsigned char *blue);

long int **cdForeground** (long int color);
long int **cdBackground** (long int color);



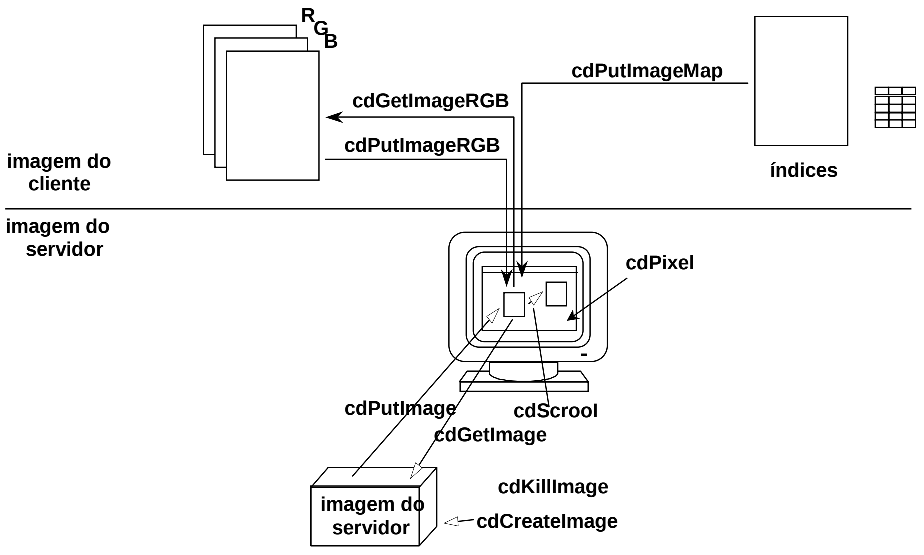
int **cdGetColorPlanes** (void);
void **cdPalette** (int n, long int *palette, int mode);

Imagens no *CanvasDraw*

/ client images */*
void **cdGetImageRGB** (unsigned char *r, unsigned char *g, unsigned char *b,
int x, int y, int w, int h);
void **cdPutImageRGB** (int iw, int ih,
unsigned char *r, unsigned char *g, unsigned char *b,
int x, int y, int w, int h);
void **cdPutImageMap** (int iw, int ih,
unsigned char *index, long int *colors,
int x, int y, int w, int h);

/ server images */*
void **cdPixel** (int x, int y, long int color);
void* **cdCreateImage** (int w, int h);
void **cdGetImage** (void* image, int x, int y);
void **cdPutImage** (void* image, int x, int y);
void **cdKillImage** (void* image);
void **cdScrollImage** (int xmin, int xmax, int ymin, int ymax, int dx, int dy);

Esquema de Imagens do *CanvasDraw*

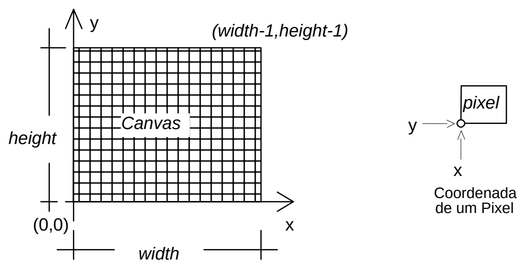


Controle no *CanvasDraw*

cdCanvas ***cdCreateCanvas** (cdContext driver, void *data);
int **cdActivate** (cdCanvas *canvas);
void **cdKillCanvas** (cdCanvas *canvas);

void **cdFlush** (void);
void **cdClear** (void);
int **cdPlay** (cdContext driver, void *data);

Coordenadas no *CanvasDraw*



void **cdGetCanvasSize** (int *width, int *height,
double *mm_width, double *mm_height);

void **cdCanvas2Raster** (int *x, int *y);

int **cdClip** (int mode);
void **cdClipArea** (int xmin, int xmax, int ymin, int ymax);
int **cdGetClipArea** (int *xmin, int *xmax, int *ymin, int *ymax);