Grafische Benutzeroberflächen mit Tcl/Tk

Christian Gollwitzer

Experimentalphysik V, Universität Bayreuth, D-95440 Bayreuth

16. Oktober 2006

Wozu GUIs?

- Aufwendig zu programmieren
- Schwieriger zu debuggen
- Weniger portabel
- Grafische Eingabe von Startdaten
- Beispiel Kantenerkennung

Toolkits im Vergleich: Ein Knopf

- Grafisches Hello-World-Programm: Ein Fenster
- Ein Knopf, der beim Betätigen eine Aktion auslöst, z. B. Beenden
- WinAPI
- MFC
- QT
- Tcl/Tk

Toolkits im Vergleich: Ein Knopf

Christian Gollwitzer

Windows API

```
1   #include <windows.h>
2
3   LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
4
5   int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
6                      PSTR szCmdLine, int iCmdShow)
7   {
8      MSG msg;
9      HWND hWnd;
10     WNDCLASS wc;
11
12     const char szAppName[] = "Windows Buttons";
13
14     wc.cbClsExtra = 0;
15     wc.cbWndExtra = 0;
16     wc.hbrBackground = (HBRUSH) GetStockObject(LTGRAY_BRUSH);
17     wc.hCursor = LoadCursor(NULL, IDC_ARROW);
18     wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
19     wc.hInstance = hInstance;
20     wc.lpfnWndProc = WndProc;
21     wc.lpszClassName = szAppName;
22     wc.lpszMenuName = NULL;
23     wc.style = CS_HREDRAW | CS_VREDRAW;
24
25     RegisterClass(&wc);
26
27     hWnd = CreateWindow( szAppName,
28                          szAppName,
29                          WS_OVERLAPPEDWINDOW,
30                          CW_USEDEFAULT,
31                          CW_USEDEFAULT,
32                          CW_USEDEFAULT,
33                          CW_USEDEFAULT,
34                          NULL,
35                          NULL,
36                          hInstance,
37                          NULL);
38
39     ShowWindow(hWnd, iCmdShow);
40     UpdateWindow(hWnd);
41
42     while (GetMessage(&msg, NULL, 0, 0))
43     {
44        TranslateMessage(&msg);
45        DispatchMessage(&msg);
46     }
47
48     return msg.wParam;
49  }
50
51  LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
52  {
53     static HWND hButton;
54
55     switch (message)
56     {
57     case WM_CREATE:
58        {
59              hButton = CreateWindow( "button",
60                                      "Beenden WinAPI",
61                                      WS_CHILD | WS_VISIBLE,
62                                      0, 0, 0, 0,
63                                      hWnd,
64                                      NULL,
65                                      ((LPCREATESTRUCT) lParam) -> hInstance,
66                                      NULL);
67           return 0;
68        }
69     case WM_SIZE:
70        {
71              MoveWindow(hButton, LOWORD(lParam) / 2 - 80, HIWORD(lParam) - 30,
72                                                          160, 22, TRUE);
73           return 0;
74        }
75
76        case WM_COMMAND:
77        {
78           if (lParam == (LPARAM)hButton)
79           {
80              if (HIWORD(wParam) == BN_CLICKED)
81                 SendMessage(hWnd, WM_CLOSE, 0, 0);
82           }
83           return 0;
84        }
85     case WM_DESTROY:
86        {
87           PostQuitMessage(0);
88           return 0;
89        }
90     }
91     return DefWindowProc(hWnd, message, wParam, lParam);
92  }
```

Toolkits im Vergleich: Ein Knopf

Christian Gollwitzer

Microsoft Foundation Classes MFC

```
1   #include "stdafx.h"
2
3   class CHello : public CFrameWnd
4   { CButton * endbutton;
5   public:
6       CHello()
7       {
8           Create(NULL,_T("Hello World!"),WS_OVERLAPPEDWINDOW,rectDefault);
9           endbutton=new CButton();
10          endbutton->Create("Beenden MFC", WS_VISIBLE, CRect(50, 50, 210, 72), this, 10);
11      }
12  protected:
13      afx_msg void OnSize(UINT n, int cx, int cy);
14      afx_msg void ButtonClicked(UINT n);
15      DECLARE_MESSAGE_MAP( )
16  };
17
18  BEGIN_MESSAGE_MAP(CHello, CFrameWnd)
19    ON_WM_SIZE()
20    ON_CONTROL_RANGE(BN_CLICKED, 10, 10, ButtonClicked)
21  END_MESSAGE_MAP( )
22
23  void CHello::ButtonClicked(UINT n) {
24         PostQuitMessage(0);
25  }
26
27  void CHello::OnSize(UINT n, int cx, int cy) {;
28            endbutton->MoveWindow( cx/ 2 - 80, cy- 30, 160, 22, TRUE);
29            CFrameWnd::OnSize(n, cx, cy);
30  }
31
32  class CHelloApp : public CWinApp
33  {
34  public:
35      virtual BOOL InitInstance();
36  };
37
38  BOOL CHelloApp::InitInstance()
39  {
40      m_pMainWnd = new CHello();
41      m_pMainWnd->ShowWindow(m_nCmdShow);
42      m_pMainWnd->UpdateWindow();
43      return TRUE;
44  }
45
46  CHelloApp theApp;
```

Toolkits im Vergleich: Ein Knopf

Modernes C++ — QT

```
1  #include <QApplication>
2  #include <QHBoxLayout>
3  #include <QPushButton>
4
5
6  int main(int argc, char *argv[])
7  {
8      QApplication app(argc, argv);
9
10     QWidget window;
11     window.resize(200, 120);
12
13     QPushButton *quit= new QPushButton("Beenden QT");
14     QObject::connect(quit, SIGNAL(clicked()), &app, SLOT(quit()));
15
16     QVBoxLayout *layout=new QVBoxLayout;
17     layout->addStretch();
18     layout->addWidget(quit);
19     window.setLayout(layout);
20
21     window.show();
22     return app.exec();
23 }
```

Toolkits im Vergleich: Ein Knopf

Tcl/Tk

```
1 #!/usr/bin/wish
2 button .b -text "Beenden TK" -command exit
3 pack .b -side bottom
```

GUI-Konzept – Inversion of Control

- Ablauf in seriellen Programmen hartkodiert: *Fordere A*→*Fordere B*→*Rechne*→*Ausgabe*
- Ablauf in GUIs vom Benutzer gesteuert, z.B.: *Eingabe B*→*Eingabe A*→*Korrektur B*→*Rechne* →*Korrektur A*→*Nochmal Rechnen...*
- Realisierung: Eventloop. Benutzer sendet *Events*, Programm reagiert (Aufruf einer Funktion)
- Programmreaktion muss *schnell* sein ($< 0.5\,\mathrm{s}$).
- Verschiedenartige Events: *low-level* (Maus bewegt, Klick links, Leertaste losgelassen, Neuzeichnen, Fenstergröße geändert) *high-level* (OK geklickt, Zahl geändert, Menüpunkt gewählt)

TCL in Kürze

- Shellartige Skriptsprache von John Ousterhout
- Anfänge als C-Erweiterung
- Minimalsprachenkonzept, nur 7(!) Syntaxregeln
- *kommando arg1 arg2 ...*
  `puts "Hallo Welt"`
- Kommandoersetzung: `puts [expr 3+7]`
- Variablenersetzung ($), Verkettung (;), Kommentar (#):
  `set a 7; puts $a; #dies ist ein Kommentar`
- Quoting mit "" oder {}:

```
1    set a 5+7; set b [expr 5+7]
2    puts a=b
3    puts {$a=$b}
4    puts "$a=$b"
```

TK in Kürze

- Hochmodernes Konzept von 1989(!) John Ousterhout
- Anfänge als Tcl-Erweiterung
- Widgetnamen: `.urahn.grossmutter.vater.kind`

```
1 package require Tk
2 button .b1 -text "Servus" -command {
3    puts "Servus $name" }
4 entry .e -textvariable name
5 label .l -text "Dein Name:"
6 pack .l .e .b1 -side top
```

Aufgaben

- Knopf zum Löschen der Eingabe

Aufgaben

- Knopf zum Löschen der Eingabe

```
1   pack [button .b2 -text "Loeschen" -command {
2     set name "" }]
```

Aufgaben

- Knopf zum Löschen der Eingabe

```
1   pack [button .b2 -text "Loeschen" -command {
2     set name "" }]
```

- Anordnen der 2 Knöpfe nebeneinander (frame)

Aufgaben

- Knopf zum Löschen der Eingabe

```
1   pack [button .b2 -text "Loeschen" -command {
2     set name "" }]
```

- Anordnen der 2 Knöpfe nebeneinander (frame)

```
1   frame .buttons
2   button .buttons.b1 ...
3   button .buttons.b2 ...
4   pack .l .e .buttons -side top
5   pack .buttons.b1 .buttons.b2 -side left
```

Eventmodell in Tk

- High level meist durch Optionen:

```
1 button .b -command {}
2 scale .s -variable var
```

- Low-level durch Bindings:

```
1 label .l -text "Streichle mich"
2 bind .l <Enter> { .l configure -background pink }
3 bind .l <Leave> { .l conf -background lightgray}
4 bind .l <1> { puts "You klicked me at %x %y" }
5 pack .l
```

Aufgabe – GUI für Kantenerkennung

- Dateiauswahlbox `set filename [tk_getOpenFile]`
- Bild laden und anzeigen

```
1 image create photo kantenbild -file $filename
2 canvas .c -width 400 -height 300
3 .c create image 0 0 -anchor nw -image kantenbild
```

- Programm ausführen und Ergebnis speichern

```
1 set ergebnis [exec programm arg1 arg2 ]
```

- Rote Linie zeichnen

```
1 .c create line $coords -fill red -width 3
```

Kantenerkennung - Lösung


```
1  set filename [tk_getOpenFile]
2
3  image create photo kantenbild -file $filename
4  canvas .c -width 400 -height 300
5
6  .c create image 0 0 -anchor nw -image kantenbild
7
8  pack .c
9  bind .c <1> {
10   set coords [exec ./sobellaeufer $filename %x %y 3]
11   .c create line $coords -fill red -width 3
12 }
```