Cross-Plattform-Entwicklung mit WX-Windows

Grenzen überwinden

Plattformübergreifendes Programmieren ist eine Kunst: Um die großen Unterschieden von Linux, Windows und Mac zu überwinden, müssen Cross-Plattform-Toolkits viel leisten. Dieser Artikel stellt WX-Windows vor, eine GUI-Bibliothek, die seit über zehn Jahren Entwicklern diese Aufgabe erleichtert. Steffen Panning

Welches ist das beste Cross-Plattform-Toolkit? Diese Frage führt in einem Entwicklerforum schnell zu einem kleinen Religionskrieg. Eine der möglichen Antworten lautet WX-Windows [1]. Neben dem persönlichen Geschmack entscheiden auch Einsatzgebiet oder Firmenpolitik über eine sinnvolle Auswahl. Vergleichen lässt sich WX unter anderem mit QT, der Cross-Plattform-Bibliothek der norwegischen Firma Trolltech [2].

Spieglein, Spieglein ...

Beim Vergleich von WX und QT fallen drei Unterschiede besonders auf: interne Realisierung, Lizenz und verfügbare Plattformen. Der erste Unterschied betrifft das technische Prinzip: QT zeichnet die grafische Oberfläche selbst, ohne dabei auf Widget-Bibliotheken der Zielplattform zuzugreifen. WX beschränkt sich dagegen meist auf die Rolle einer Vermittlerin, die dem Programmierer ein einheitliches API anbietet. Für die Darstellung sind plattformspezifische Bibliotheken zuständig. Unter Linux stehen sogar zwei Widget-Bibliotheken zur Auswahl: Motif/X11 und GTK, siehe Abbildungen 1 und 2.

Die meisten Linux-Benutzer erwarten nicht unbedingt, dass verschiedene GUI-Programme dasselbe Look & Feel benutzen. Auf anderen Plattformen, etwa Microsoft Windows, führt aber selbst ein geringfügig anderes GUI-Verhalten zu Irritationen, da der Benutzer dieses Verhalten nicht gewohnt ist. WX-Applikationen passen sich automatisch an die Gegebenheiten der Plattform an.

Der zweite, oft wichtigere Unterschied ist die Lizenz. Die QT-Bibliothek ist auf Linux-Systemen und Mac OS X durch die GPL lizenziert. Wer sein Programm auch Windows-Nutzern zur Verfügung stellen will, muss eine recht teure Lizenz kaufen. WX-Windows unterliegt dagegen auf allen Plattformen der weniger restriktiven LGPL-Lizenz. Drittens unterstützt WX einige Plattformen, die QT nicht erreicht, etwa MGL und OS/2.

Geschmacksrichtungen

WX-Windows unterstützt alle Windows-Betriebssysteme ab Version 3.1, Mac OS 9 und X, Unix-artige Betriebssysteme, OS/2 sowie verschiedene Embedded-Systeme. Die grafisch orientierte WX-Windows-Bibliothek hat noch eine interessante Schwester: WX-Base. Dieser Untermenge von WX-Windows fehlen die grafischen Klassen. Mit WX-Base lassen sich plattformunabhängige Kommandozeilenprogramme für Systeme schreiben, auf denen keine grafische Oberfläche installiert ist.

Der Programmierer darf nicht nur die Zielplattform recht frei wählen, auch bei der Programmiersprache hat ihm WX einiges zu bieten: Neben dem Default-API in C++ existieren Bindings zu vielen anderen Sprachen (Tabelle 1). Die Unterstützung der einzelnen Sprachen ist allerdings unterschiedlich weit entwickelt. Während beispielsweise die Python- und Perl-Bindungen einen recht ausgereiften Eindruck machen, steckt die Dotnet-Unterstützung noch in den Kinderschuhen.

Lang gereift

Durch die lange Entwicklungszeit ist WX zu einem robusten Werkzeug gereift, das den GUI-Entwickler auf vielfältige Weise unterstützt. Das wichtigste Feature ist das einheitliche API, WX unterstützt zudem mehrsprachige Programme, lässt

die Software auch Original-Konfigurationsformate der jeweiligen Zielplattform benutzen und verwendet ein XML-basiertes Ressourcensystem.
Darüber hinaus erklären rund 70 mitgelieferte Beispiele den Umgang mit der Bibliothek und helfen beim Einstieg in komplexere Techniken wie dem Document-View-Framework, Debugging und Logging, der Datenbankunterstützung sowie dem Printing-API.

Eigenheiten

Seine lange Vorgeschichte führt allerdings auch dazu, dass das Framework heute mit einigen Altlasten zu kämpfen hat. So stellt WX eigene Datencontainer zur Verfügung, die vom C++-Standard abweichen. Diese Container haben eine ähnliche Funktionalität wie die entsprechenden STL-Klassen (Standard Template Library), sie sind jedoch nicht miteinander kompatibel.
Die Dokumentation weist als Hauptgrund für diesen Sonderweg neben der langen Geschichte die unterschiedlichen Voraussetzungen der unterstützten Systeme aus. Zu Beginn des WX-Projekts waren Compiler, die mit C++-Templates umgehen konnten, noch rar. Die STL trägt schon im Namen, dass sie diese Technologie intensiv nutzt, und kam somit als Basis des Frameworks nicht in Frage. Seit kurzem wollen einige WX-Entwickler aber die STL in ihr Framework einbinden.
Eine weitere Schwäche ist die fehlende Ausnahmefestigkeit, sie ist auf dieselben Gründe zurückzuführen. Das Entwicklerteam will zwar in einer der nächsten Versionen eine ausnahmefeste Bibliothek anbieten, das erweist sich jedoch laut Mailingliste wegen der gewachsenen Codestrukturen als recht schwierige Aufgabe.

Tabelle 1: WX-Bindings

Sprache	URL
Basic	http://wxbasic.sourceforge.net
Dotnet	http://wxnet.sourceforge.net
Eiffel	http://elj.sourceforge.net/projects/gui/ewxw/
Haskell	http://wxhaskell.sourceforge.net
Java	http://www.wx4j.org
Javascript	http://wxjs.sourceforge.net
Lua	http://www.luascript.thersgb.net
Perl	http://wxperl.sourceforge.net
Python	http://wxpython.org
Ruby	http://wxruby.rubyforge.org

Schrittweise durch die Quellen

Der in den Listings 1 und 2 abgedruckte Sourcecode zeigt die Grundlagen der WX-Programmierung anhand eines sehr einfachen Texteditors (siehe Abbildungen 1 und 2). Der Header in Listing 1 deklariert die drei Klassen »HelloApp«, »HelloFrame« und »AboutDialog«. Als Kindklasse von »wxApp« ist »HelloApp« die Basisklasse der ganzen Applikation, »HelloFrame« zeigt Einzelheiten zum Eventhandling und »AboutDialog« führt in die Grundlagen der WX-eigenen Layoutstrategie ein.
Das Ableiten einer Subklasse von »wxApp« ist der übliche Weg, um eine eigene Applikation zu erstellen. Meist genügt es, die »OnInit()«-Methode zu überschreiben, um die Anwendung zu initialisieren. Die Implementation in Listing 2 (Zeilen 9 bis 15) hat nicht viel zu tun. Sie erzeugt lediglich eine Instanz von »HelloFrame«, erklärt sie zum obersten Fenster und zeigt sie an.
Das Makro »DECLARE_APP(HelloApp)« in Listing 1, Zeile 13, fügt eine globale Funktion »wxGetApp()« ein, die eine Referenz auf die Instanz von »HelloApp« zurückgibt. Der »AboutDialog« in Zeile 15 ist eine einfache Kindklasse von »wxDialog« und benötigt lediglich einen Konstruktor. Die Zeilen 21 bis 42 beschreiben »HelloFrame«, das Hauptfenster des Editors. Das Makro »IMPLEMENT_APP(HelloApp)« in Listing 2, Zeile 7, ergänzt die Implementierungsdatei um eine Main-Funktion mit allen Aufrufen, die zum Start der Applikation erforderlich sind.
Der Konstruktor von »HelloFrame« erzeugt in den Zeilen 27 bis 38 (Listing 2) die Menüleiste der Anwendung. Hier fällt auf, dass der Code zwar Elemente im Freispeicher anlegt (»new«), die Klasse »HelloFrame« aber keinen Destruktor besitzt, der diese Elemente wieder zerstört würde (»delete«). Des Rätsels Lösung: WX übernimmt die fehleranfällige Destruktion selbst. Welche Menü-Elemente WX freigeben soll, weiß es, sobald das Programm sie mit »Append()« in das übergeordnete Element eingehängt hat (Zeilen 32 und 36).

Widgets einbinden

Das in Zeile 39 angelegte Textfeld zeigt die übliche Strategie, um Widgets zu erzeugen, deren Klasse von »wxWindow« erbt. Der erste Parameter des Konstruktors gibt das übergeordnete Element an. Wenn es ungleich »NULL« ist, zerstört WX die Kindelemente rekursiv, bevor es

Listing 1: Header des Editors

```
01 /**********
02  hellowx.h
03  **********/
04
05 #include <wx/wx.h>
06
07 class HelloApp:public wxApp
08 {
09 public:
10   virtual bool OnInit();
11 };
12
13 DECLARE_APP(HelloApp);
14
15 class AboutDialog:public wxDialog
16 {
17 public:
18   AboutDialog();
19 };
20
21 class HelloFrame:public wxFrame
22 {
23 public:
24   HelloFrame();
25   void OnSave(wxCommandEvent &event);
26   void OnOpen(wxCommandEvent &event);
27   void OnQuit(wxCommandEvent &event);
28   void OnAbout(wxCommandEvent &event);
29 private:
30   wxMenuBar *m_ptrMenBar;
31   wxTextCtrl *m_ptrText;
32   wxMenu *m_ptrFileMenu, *m_ptrHelpMenu;
33
34   enum {
35     OPEN_EVT_ID = wxID_HIGHEST + 1,
36     SAVE_EVT_ID,
37     NEW_EVT_ID,
38     QUIT_EVT_ID,
39     ABOUT_EVT_ID
40   };
41   DECLARE_EVENT_TABLE();
42 };
```

das Vaterelement zerstört. Es ist daher eine gute Strategie, bei WX-Windows alle Elemente im Freispeicher zu erzeugen und es der Bibliothek zu überlassen, Instanzen freizugeben.

Die Ereignisverarbeitung besteht bei WX-Programmen aus mehreren Teilen. Die Enum-Deklaration im Header (Listing 1 ab Zeile 35) erzeugt IDs, mit denen WX ein Ereignis einer verarbeitenden Methode zuordnet. Ein ID-Bereich ist für WX-interne Aufgaben reserviert. Um nicht versehentlich eine intern genutzte ID zu verwenden, beginnen die selbst definierten Event-IDs beim Wert »wxID_HIGHEST + 1«. Die Makros »wxID_LOWEST« und »wxID_HIGHEST« bezeichnen die untere und obere Grenze der internen IDs. In Zeile 42 fügt das Makro »DECLARE_EVENT_TABLE()« einige Variablen und Methodendeklarationen ein, um die Ereignisverarbeitung der Klasse vorzubereiten.

Ereignisse verarbeiten

In der Implementierungsdatei (Listing 2) beschreibt eine Event-Tabelle, wie eine Klasse die Ereignisse verarbeiten soll. Das WX-Makro »BEGIN_EVENT_

Listing 2: Implementierung des Editors

```
01 /**********
02  hellowx.cc
03  **********/
04
05 #include "hellowx.h"
06
07 IMPLEMENT_APP(HelloApp);
08
09 bool HelloApp::OnInit()
10 {
11   HelloFrame *ptr_main = new HelloFrame();
12   SetTopWindow(ptr_main);
13   ptr_main->Show(TRUE);
14   return TRUE;
15 }
16
17 BEGIN_EVENT_TABLE(HelloFrame, wxFrame)
18   EVT_MENU(OPEN_EVT_ID, HelloFrame::OnOpen)
19   EVT_MENU(SAVE_EVT_ID, HelloFrame::OnSave)
20   EVT_MENU(QUIT_EVT_ID, HelloFrame::OnQuit)
21   EVT_MENU(ABOUT_EVT_ID, HelloFrame::OnAbout)
22 END_EVENT_TABLE()
23
24 HelloFrame::HelloFrame()
25   :wxFrame(static_cast<wxFrame*>(NULL), -1, "Hello wxWindows")
26 {
27   m_ptrMenBar = new wxMenuBar();
28   m_ptrFileMenu = new wxMenu();
29   m_ptrFileMenu->Append(OPEN_EVT_ID, "&Oeffnen");
30   m_ptrFileMenu->Append(SAVE_EVT_ID, "&Speichern");
31   m_ptrFileMenu->Append(QUIT_EVT_ID, "&Beenden");
32   m_ptrMenBar->Append(m_ptrFileMenu, "&Datei");
33
34   m_ptrHelpMenu = new wxMenu();
35   m_ptrHelpMenu->Append(ABOUT_EVT_ID, "&About");
36   m_ptrMenBar->Append(m_ptrHelpMenu, "&Hilfe");
37
38   SetMenuBar(m_ptrMenBar);
39   m_ptrText = new wxTextCtrl(this, -1, "",
40     wxDefaultPosition, wxDefaultSize, wxTE_MULTILINE);
41 }
42
43 void HelloFrame::OnSave(wxCommandEvent &event)
44 {
45   wxFileDialog saveDialog(static_cast<wxWindow*>(NULL),
46     "Save", "", "", "*.txt", wxSAVE);
47   if(saveDialog.ShowModal() == wxID_OK) {
48     m_ptrText->SaveFile(saveDialog.GetPath());
49   }
50 }
51
52 void HelloFrame::OnOpen(wxCommandEvent &event)
53 {
54   wxFileDialog openDialog(static_cast<wxWindow*>(NULL),
55     "Open", "", "", "*.txt", wxOPEN);
56   if(openDialog.ShowModal() == wxID_OK) {
57     m_ptrText->LoadFile(openDialog.GetPath());
58   }
59 }
60
61 void HelloFrame::OnQuit(wxCommandEvent &event)
62 {
63   if(!m_ptrText->IsModified()) {
64     Close(TRUE);
65     return;
66   }
67
68   wxMessageDialog msgDialog(static_cast<wxWindow*>(NULL),
69     "Datei wurde geaendert. Wirklich beenden?",
70     "Wirklich beenden?",
71     wxOK | wxCANCEL | wxICON_QUESTION);
72   if(msgDialog.ShowModal() == wxID_OK) {
73     Close(TRUE);
74   }
75 }
76
77 void HelloFrame::OnAbout(wxCommandEvent &event)
78 {
79   AboutDialog aboutdlg;
80   aboutdlg.ShowModal();
81 }
82
83 AboutDialog::AboutDialog()
84   :wxDialog(static_cast<wxWindow*>(NULL), -1, "About")
85 {
86   wxButton *ok_button = new wxButton(this, wxID_OK, "OK");
87
88   wxTextCtrl *about_txt = new wxTextCtrl(this, -1,
89     "Nur eine nichtssagende Aboutbox :-)",
90     wxDefaultPosition, wxSize(200, 100),
91     wxTE_MULTILINE | wxTE_READONLY);
92
93   wxBoxSizer *ptr_sizer = new wxBoxSizer(wxVERTICAL);
94   ptr_sizer->Add(about_txt, 0, wxALL, 5);
95   ptr_sizer->Add(ok_button, 0, wxALIGN_CENTER, 10);
96   SetSizer(ptr_sizer);
97   ptr_sizer->SetSizeHints(this);
98 }
```

TABLE()« legt fest, dass »HelloFrame« eine Kindklasse von »wxFrame« ist und die bis »END_EVENT_TABLE()« genannten Ereignisse selbst behandeln möchte. Die Zeilen 18 bis 21 beschreiben mit dem Makro »EVT_MENU(*Event_ID*, *Ereignis_Methode*)«, mit welcher Methode die Anwendung auf die Ereignis-IDs reagieren soll.

In diesem Beispiel sind es die Menü-Einträge, die zu den Ereignissen führen: Die Aufrufe in den Zeilen 29 bis 31 und 35 verknüpfen je eine Ereignis-ID mit einem Menü-Eintrag. Klickt ein Anwender auf den Eintrag, löst WX einen »wxCommandEvent« mit dieser ID aus. Die Bibliothek durchsucht nun die Event-Tabelle nach der entsprechenden ID und führt die damit verknüpfte Methode aus.

Callback-Funktionen

Die Zeilen 43 bis 81 zeigen die Callback-Funktionen. »OnAbout()« öffnet die selbst erstellte About-Box, »OnSave()« öffnet einen Speichern-Dialog. Mit der If-Anweisung in Zeile 47 bringt das Programm in Erfahrung, ob der Anwender den OK-Button des Speichern-Dialogs gedrückt hat. Ist dies der Fall, speichert die Callback-Funktion den eingegebenen Text unter dem Namen, den der User im Dialogfenster angegeben hat. Die Methode »OnOpen()« funktioniert analog. »OnQuit()« prüft, ob der Text der angezeigten Datei geändert wurde. Ist dies der Fall, erzeugt sie in Zeile 68 einen Standard-Systemdialog, der den User auf die Änderung aufmerksam macht.

Eine Besonderheit der Eventbehandlung steckt in dem Konstruktor von »AboutDialog«: Er erstellt einen Button, erzeugt aber keine Event-Tabelle. Die Ereignisverarbeitung funktioniert, weil die Vaterklasse »wxDialog« eine Methode »OnOK()« besitzt. Um das automatische Verhalten der Dialogklasse auszunutzen, muss nur die ID des Buttons mit der vordefinierten ID »wxID_OK« belegt sein.

Das Layoutmanagement verstehen

Bei WX-Windows ist, wie bei den meisten Widget-Bibliotheken, das Pixel-genaue Positionieren der GUI-Elemente verpönt. Für GUI-Builder wären Pixelgenaue Platzierungen zwar einfacher, auch manch Programmierer zieht diese anschauliche Variante vor. Sie führt aber zu vielen Problemen, wenn die Applikation auf mehreren Plattformen laufen soll, die Beschriftungen der Widgets mehrsprachig sind oder der Benutzer die Fontgröße ändern will.

Bei WX-Windows sind die Sizer-Klassen für das Layout verantwortlich. Sie erfordern zwar eine gewisse Einarbeitungszeit, ermöglichen aber ein sehr flexibles Layout der Applikationsoberfläche. Statt genauer Pixel-Positionen verwenden die Sizer-Klassen Layoutregeln, die der Programmierer festlegt. In Zeile 93 (Listing 2) wird ein »BoxSizer« erstellt, der ein Textfeld und einen Button erhält. Der Sizer hat ein recht einfaches Layoutkonzept, er ordnet Elemente nebeneinander oder untereinander an.

Das Flag »wxVERTICAL« gibt an, dass der Box-Sizer die Komponenten untereinander positionieren soll. Per »Add()«-Methode erfährt der Sizer, um welche Elemente er sich kümmern muss. Der Aufruf in Zeile 94 fügt dem Sizer das Textfeld »about_txt« hinzu. Der zweite Parameter bestimmt, wie sich das eingefügte Element verhalten soll, wenn sich die Größe in Richtung Hauptorientierung ändert. Im Beispiel ist das die vertikale Ausrichtung. Der dritte Parameter bestimmt unterschiedliche Eigenschaften, die der Programmierer bei Bedarf verordnet. Der hier verwendete Parameter »wxALL« legt fest, dass das Element auf jeder Seite einen Rahmen haben soll. Der letzte Parameter bestimmt den Abstand um jedes Element in Pixeln.

Der Aufruf »SetSizer()« in Zeile 96 macht »ptr_sizer« zum Sizer von »AboutDialog«, die Sizer-Methode »SetSizeHints()« in der nächsten Zeile bestimmt, dass sich der Sizer an der Größe des About-Dialogs orientieren soll.

Alles zusammenbauen

Weil das Programm nur aus zwei Dateien besteht, lässt es sich auch ohne Makefile leicht übersetzen. Das Shellskript »wx-config«, das der WX-Bibliothek beiliegt, erleichtert den Umgang mit dem Toolkit und seinen Abhängigkeiten von anderen Bibliotheken. Um das Programm zu erstellen, genügt der folgende Aufruf:

```
g++ -Wall -g `wx-config --cflags --libs` ➥
  hellowx.cc -o hellowx
```

WX-Windows ist ein modernes Werkzeug, das eine lange Entwicklungszeit hinter sich hat. Es wird von einer treuen Fangemeinde genutzt und gepflegt, die auch jedem Neuling gerne mit Rat und Tat zur Seite steht. Neben der Dokumentation stehen auf der Projektseite [1] auch noch hilfreiche Tutorials zum Download bereit.

WX-Windows ist nicht perfekt, die Entwicklergemeinde arbeitet jedoch permanent an seiner Verbesserung. Wer mit den leichten Einschränkungen leben kann, hat ein gut getestetes Werkzeug zur Hand, das seine Alltagstauglichkeit seit über zehn Jahren beweist. *(fjl)* ■

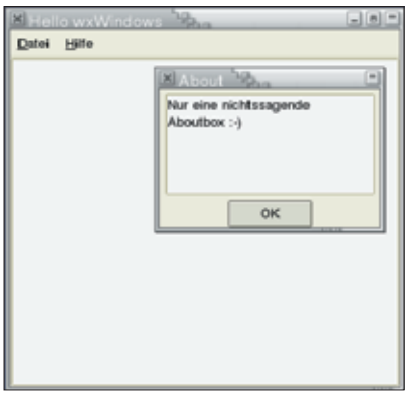

Abbildung 1: Der Beispiel-Editor (140 Zeilen Quellcode, siehe Listings 1 und 2) ist hier mit der GTK-Fassung von WX-Windows gelinkt.

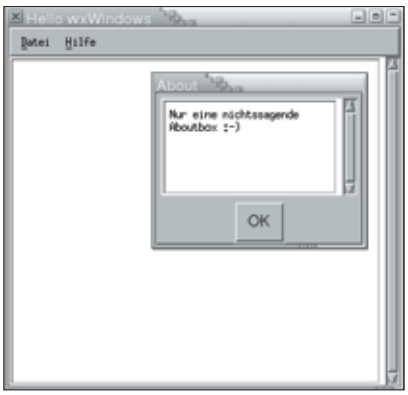

Abbildung 2: Dasselbe Programm wie in Abbildung 1, hier aber im Motif-Stil: WX-Windows ist ein einheitliches API zu verschiedenen Widget-Bibliotheken.

Infos

[1] WX-Windows: [http://www.wxWindows.org]
[2] QT: [http://www.troll.no]