Lazarus: Open Source statt Delphi und Kylix

Michaël Van Canneyt, Entwickler von Free Pascal

Fast alle Windows-Programmierer kennen Borland Delphi. Manche davon wissen sogar, daß es Delphi als »Kylix« auch auf Linux gibt. Aber nicht allen ist bekannt, daß es eine Alternative zu diesen Systemen als Open Source gibt: Lazarus. Dieser Beitrag soll dem abhelfen.

Die Zahl der mit Kylix entwickelten Anwendungen ist nicht gerade beeindruckend. Das ist eigentlich schade, denn Kylix ist ein gutes RAD-Tool, mit dem das Programmieren von GUI- oder Webapplikationen Spaß macht. Woran liegt es also? Vielleicht ist ein Teil der Antwort in der Linux-Gemeinde begründet: sie ist zum größten Teil eine Open-Source-Community. Kylix ist nicht Open Source.

Die Open Edition gibt es zwar frei zum Download und sie kann auch zum Entwicklen freier Applikationen eingesetzt werden. Aber das ist nicht das gleiche wie eine echte Open-Source-Anwendung, die mit den Quelltexten ausgeliefert wird und auch modifiziert werden kann. Was schwerer wiegt, ist die mit der Open Edition angebotene Funktionalität von Kylix: das, was Kylix so attraktiv für Entwickler macht, muß extra bezahlt werden.

Die Architektur von Lazarus

Seit einiger Zeit ist eine echte Open-Source-Alternative im Entstehen: Lazarus. Es ist die Weiterführung des Megido-Projekts und soll die Alternative zu Delphi oder Kylix sein.

Dort, wo es realisierbar ist, ist es hundertprozentig quelltextkompatibel mit der VCL von Delphi, und genauso unabhängig von den in Linux oder Windows verfügbaren verschiedenen Widget-Sets. Das Lazarus-Projekt befindet sich in einem recht fortgeschrittenem Stadium, die Programmierung von Anwendungen unterscheidet sich nicht besonders von der in Delphi: echtes Open-Source-RAD.

Lazarus setzt auf den Free-Pascal-Compiler (FPC) als darunterliegenden ObjectPascal-Compiler. Ihn gibt es schon recht lange, er ist stabil und er unterstützt fast alle bekannten ObjectPascal-Konstrukte – er war nämlich von Anfang an als Turbo-Pascal- und Delphi-kompatibel geplant. Außerdem funktioniert er auf einer wachsenden Zahl von Plattformen: Linux, BSD, Windows, DOS, OS/2, Netware, QNX, Solaris, BeOS und Amiga. Das gleiche gilt für die unterstützten CPU-Typen: Intel und Motorola 68k gelten als stabil, an PowerPC und SPARC wird noch gearbeitet.

Die Laufzeitbibliothek bietet systemunabhängige grundlegende Funktionalität (Dateibehandlung, Stringmanipulationen). Zu guter Letzt enthält Free Pascal eine Menge von Units, die den Zugriff auf verschiedene auf Unix oder Windows verfügbare Bibliotheken, insbesondere das GTK, erlauben.

Free Pascal wird mit der FCL (Free Component Library) ausgeliefert, die aus Low-Level-Klassen für alle Zwecke besteht und ähnlich zu denen der VCL ist. Sie ist nicht visuell, was bedeutet, daß sie unabhängig von einem GUI-System ist: Streaming, Parser, XML-Unterstützung, Datenbankunterstützung, Prozeßkontrolle, seit neuestem auch XML-RPC-Unterstützung. Alle diese Funktionaliäten können in Lazarus verwendet werden.

Oben auf der FCL sitzt die Lazarus Class Library (LCL). Sie besteht aus Klassen für visuelle Programmierung: Checkboxen, Menüs, Edit-Controls. Sie wurde mit dem Ziel gebaut, Delphi so ähnlich wie möglich zu sein. Diese Klassen hängen vom Betriebssystem ab und – noch wichtiger – dem verfügbaren grafischen System. Diese Systemabhängigkeit wird vom sogenannten System-Interface abgedeckt: einem System von Units, die in Pascal ein wohldefiniertes Interface zur Verfügung stellen und die Systemabhängigkeiten vor den höheren Klassen verstecken. Es ist nicht von Natur aus objektorientiert und arbeitet mit einem System, das dem Message-System von Windows ähnelt, aber von ihm unabhängig ist.

Momentan werden die folgenden Interfaces entwickelt:

- Ein GTK-Schnittstelle. Sie ist sehr komplex und funktioniert auf Linux und verschiedenen BSDs einwandfrei. Sie sollte zwar auch auf Windows funktionieren, aber ein paar Fehler in der GTK-Bibliothek hindern die IDE momentan daran.
- Ein natives Win32-Interface. Es ist noch in der Entwicklung, mehrere Demoapplikationen laufen aber schon.

Die Grundlagen für ein Gnome-Interface wurden bereits gelegt. Vor längerer Zeit wurde mit einem QT-Interface begonnen, dies wurde aber wieder aufgegeben. Andere Layer,

die Motif als Widget-Set verwenden, oder FLTK oder sogar ein nativer X-Layer sind denkbar, dies alles braucht aber Zeit und Enthusiasmus. Die objektorientierte Layer – die LCL-Klassen – verwenden nur Funktionen dieser Abstraction Layers und keine direkten GUI-Systemaufrufe. Alle visuellen Klassen sind mit dieser Abstraktions-Layer gebaut, wenn einmal der ganze Abstraktions-Layer steht, kann das ganze Lazarus portiert werden.

Beachten Sie, daß man zum Programmieren von Anwendungen mit dem Free-Pascal-Compiler und LCL nicht unbedingt die Lazarus-IDE braucht, alles kann auch auf der Kommandozeile erledigt werden. Aber die IDE besitzt viele sehr attraktive Features, die die Arbeit damit zum Vergnügen werden lassen.

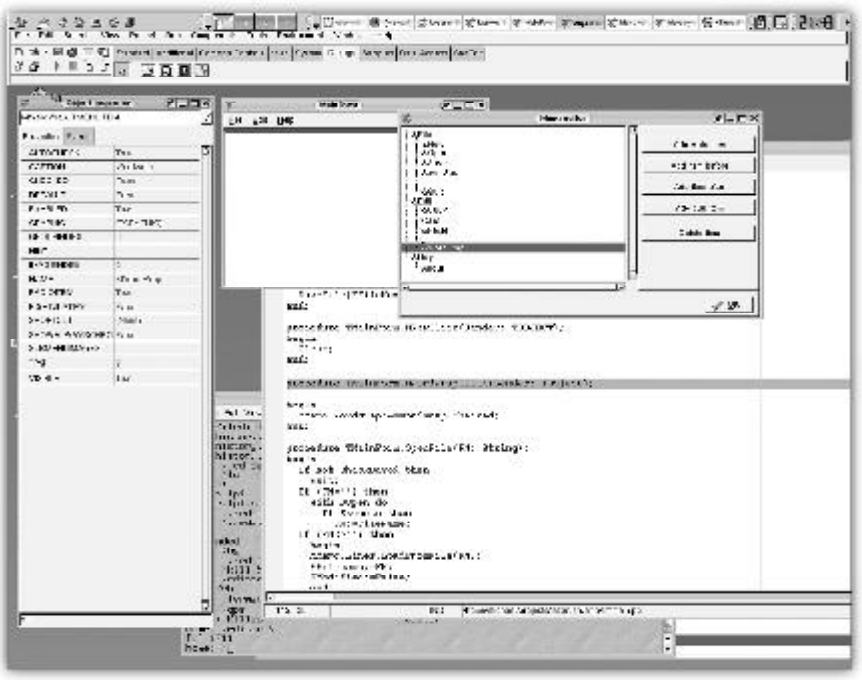

Bild 1: Die Lazarus-IDE

Die Lazarus-IDE

Auf den LCL- und FCL-Klassen sitzt die Lazarus-IDE (Integrated Development Environment). Sie besteht aus zwei Teilen:

- Einem fortgeschrittenen Code-Editor.
- Einem Form-Designer. In Bild 1 wird die IDE gezeigt, in der gerade ein einfacher Editor entworfen wird.

Beide Teile der IDE interagieren stark. Delphi-Entwickler fühlen sich sofort zuhause: Der Code-Editor von Lazarus bietet mehr als die Standard-Funktionalität von Delphi, nämlich eine Menge von Funktionen, die »Code Tools« genannt werden. Diese Funktionalität ist durch die hervorragenden SynEdit-Komponenten gegeben, die speziell für diesen Zweck auf Lazarus portiert wurden. Unter anderem gibt es folgende Features:

- Syntax Highlighting
- Code Completion: Man wählt die gewünschte Funktion aus einer Liste aus.
- Tooltips zeigen erforderliche Parameter.
- Class Completion.
- Komplett anpaßbare Tastenbelegung und noch viel mehr, das nicht alles aufgezählt werden kann.

Allein der Code-Editor bietet genug Gründe, um Lazarus als Entwicklungsumgebung einzusetzen, selbst wenn keine der visuellen Design-Fähigkeiten gebraucht wird: er bietet alles, was ein moderner Code-Editor braucht.

Der visuelle Form-Designer arbeitet wie Delphi: Forms (Fenster) werden erzeugt, Komponenten können aus der Komponentenpalette ausgewählt und auf die Fom gezogen werden. Der Object Inspector erlaubt das Setzen verschiedener Eigenschaften mittels Point-und-Klick-Techniken: aus einem Drop-down die gewünschten Eigenschaftswerte auswählen und die Eigenschaftseditoren wie den Menü-Editor starten (auch in Bild 1 zu sehen).

Das Zuweisen von Code an Laufzeit-Ereignisse (Runtime-Events, der Anwender klickt beispielsweise auf einen Button) ist auch einfach: Klickt man auf das *OnClick*-Event im Object Inspector, wird im Editor eine Methode erzeugt und der Cursor auf die Stelle positioniert, an der der Code eingetippt werden soll. Das ist zwar nichts Aufregendes für Delphi-Anwender, wer aber nur den Emacs gewohnt ist, wird sich freuen! Mit [Ctrl][F9] wird kompiliert, mit [F9] kompiliert und anschließend sofort gestartet: Lazarus übernimmt das alles. Wie es in einer IDE guter Stil ist, können alle FPC-Optionen bequem in verschiedenen Dialogen eingestellt werden. Die Integration von Debugging ist auch möglich: Breakpoints werden gesetzt, indem die Zeile ausgewählt wird, in der die Programmausführung anhalten soll (vorausgesetzt natürlich, daß Debug-Infos in die Anwendung einkompiliert wurden).

Zur Demonstration wurde eine kleine Applikation – der Ersatz für ein Notepad – erzeugt und kompiliert. Die Quellen befinden sich auf der Heft-CD. Der meiste Code für diese Anwendung (über 200 Zeilen) wurde von der IDE erzeugt, nur Prozedurrümpfe müssen explizit geschrieben werden.

Lazarus kann erweitert werden. Wenn man sich mit Delphi auskennt, weiß man, daß man in die IDE dynamisch ein Package laden kann, das die Funktionalität erweitert: Entweder wird die IDE an sich erweitert, oder es werden Wizards für bestimmte Aufgaben eingesetzt, oder es werden neue Komponenten in die Komponentenpalette integriert, die in verschiedenen Anwendungen verwendet werden können.

Lazarus besitzt die Funktionalität der dynamisch ladbaren Packages noch nicht: der darunterliegende Free-Pas-

cal-Compiler erlaubt das Erzeugen solcher Packages (noch) nicht, so daß Lazarus sie also auch nicht verwenden kann. Es wird an einem alternativen Package-System gearbeitet, das so lange eingesetzt werden wird, bis FPC Packages voll unterstützt.
Jedoch sind Packages nicht eigentlich erforderlich: da die Quelltexte von Lazarus verfügbar sind, kann die IDE individuell angepaßt und neu übersetztwerden. Das bedeutet, daß die Funktionalität zum Erweitern der IDE direkt in die IDE eingebaut werden kann. Das ist etwas, was in Delphi schwer vorstellbar ist.

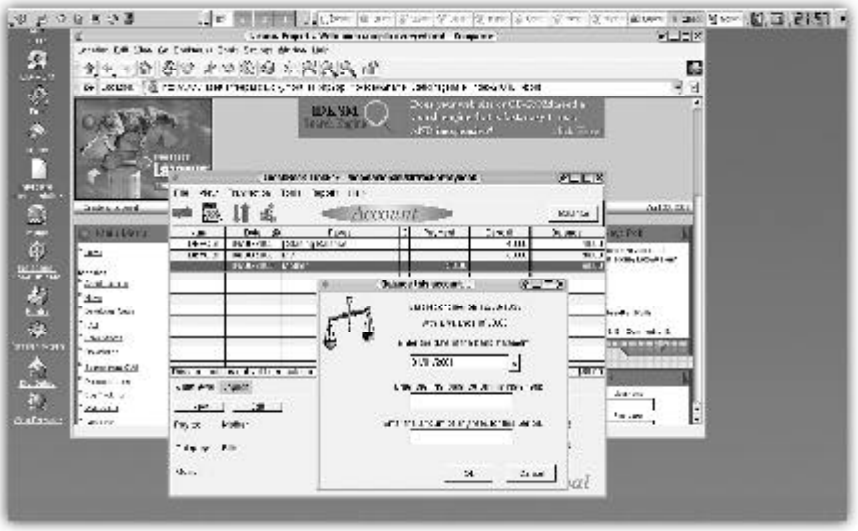

Bild 2: Der Checkbook Tracker

In Bau

Das Hinzufügen zusätzlicher Komponenten auf der Komponentenpalette findet statt, indem ein Eintrag in eine Unit erfolgt, die speziell zu diesem Zweck gehalten wird, und dann wird die IDE neu gebaut. Die IDE enthält sogar besondere Menüeinträge, um die Rekompilation zu erleichtern.
Abgesehen davon, kann die IDE mit externen Tools erweitert werden: Mit einem Dialog »External Tool« werden gesondert Programme registriert. Diese Programme bekommen dann im Lazarus-Menü einen Menüeintrag.
Mit Lazarus können Real-World-Applikationen geschrieben werden. Interessant ist der Checkbook Tracker von Tony Maro: eine Anwendung ähnlich »Microsoft Money«, die Bankkonten verwaltet (siehe Bild 2).
Die Software wurde komplett in Lazarus entwickelt und zeigt – abgesehen von ihrer Nützlichkeit als Kontoverwaltung –, daß Lazarus bereits als eine komplette Entwicklungslösung verwendet werden kann. Dies wird auch durch die Tatsache untermauert, daß Lazarus auf einer E-Learning-CD-ROM in Südafrika verteilt werden wird.
Trotzdem ist Lazarus noch weit davon entfernt, fertig zu sein. Seine Nützlichkeit hängt von der Qualität und Quantität der verfügbaren Komponenten ab. Zwar gibt es genug Komponenten, um gut aussehende und funktionelle Software zu entwickeln, dennoch ist noch genug Arbeit zu erledigen. Die unterliegende FCL ist noch nicht angeschlossen, die LCL könnte mit mehr visuellen Controls ausgestattet sein, es sollten Datenbank-Komponenten integriert werden und es könnte ruhig mehr Web-Tools geben. Das Win32-Interface sollte abgeschlossen werden, und das QT-Interface sollte vielleicht auch wieder angepackt werden.
Die IDE selbst ist noch nicht perfekt: Bilder für Buttons und Menüs müssen noch in den Code geladen werden, weil sie nicht gestreamt werden können, es könnten mehr Wizards enthalten sein. Bedarf besteht auch für eine Dokumentation – all dies wurde bereits begonnen, aber der Fortschritt ist langsam. Es gibt nicht genügend Mitarbeiter, um Lazarus zu dem zu machen, was es eines Tages sein wird: das Killer-Environment!

Getting started

Will man Lazarus testen, muß man zuerst den Free-Pascal-Compiler installieren. Er steht auf der Free-Pascal-Webseite zum Download (*http://www.freepascal.org/*). Binärversionen für verschiedene Plattformen sind verfügbar. Um die Code-Tools in Lazarus voll auszunutzen, sollten auch die Sourcen der Laufzeitbibliothek und der FCL installiert werden.
Nach der Installation des Compilers muß Lazarus selbst downgeloaded und installiert werden. Das Lazarus-Projekt hat eine separate Webseite (mit einer Menge Screenshots) unter *http://www.lazarus.freepascal.org/*. Dort gibt es auch neuere Versionen von FPC, mit denen Lazarus selbst kompiliert werden kann. Auch hier sollten die Quellen installiert werden.
Dann wird die Lazarus-IDE installiert und konfiguriert. Es muß angegeben werden, wo sich die Quellen befinden, so daß die Code-Tools verwendet werden können. Ein Dokument, das die Installation und Konfiguration detailliert beschreibt, ist auf der Webseite von Lazarus zu finden. Wie bei jedem guten Open-Source-Projekt gibt es Hilfe im Lazarus-Forum oder auf Mailinglisten. Bugs können über Online-Bugtrackers berichtet werden.
Lazarus ist ein vielversprechendes Open-Source-Projekt, das sich Entwickler, die Delphi kennen und sich nach einer Open-Source-Alternative umsehen, unbedingt anschauen sollten.
Da die Entwicklertruppe über viel zu wenig Manpower verfügt – es sind gerade einmal eine Handvoll Personen, die aktiv Lazarus entwickeln – ist Hilfe immer willkommen! Betrachtet man das, was bereits vorliegt, kann man Lazarus mit Fug und Recht als eine Perle bezeichnen. ◆