Lazarus: Open source Delphi or Kylix

Michaël Van Canneyt

June 24, 2007

Abstract

Many windows programmers know Borland Delphi. Some of them even know that Delphi exists on Linux: Kylix. Probably even fewer know that there exists a true open source alternative to these: Lazarus. Maybe this article can change that.

1 Introduction

The number of applications for Linux written with Kylix is as yet not overwhelming. This is a pity, because Kylix is a good RAD environment wich makes developing GUI or even web applications a breeze. So why then aren't there more applications written in Kylix? Maybe part of the answer lies in the Linux community itself: it is by and large still an open source community. Kylix is not open source. The open edition is of course free for download, and can be used to develop free applications. But that is not the same as a true 'open source' application: one which comes with sources, and which can be tinkered with. What is more, the functionality available in the Open edition is not what makes Kylix so great to develop with: most things that would make the difference must be paid for.

Since quite some time, a real open source alternative is in the making: Lazarus. Born out of an even older project (named 'Megido'), its aim was and is to be a real alternative to Delphi/Kylix. Where possible 100% source-compatible to Delphi's VCL, and also independent of the various widget sets available in Linux or Windows. Since its initial implementation, the lazarus project has come quite a long way, and is usable: real-world applications can be developed with it in a manner which doesn't essentially differ from Delphi: true open source RAD.

2 Lazarus Architecture

To start with, lazarus uses the Free Pascal Compiler (FPC) as its underlying Object Pascal compiler. This open source compiler has been around for quite a while, is stable, and supports almost all known Object Pascal constructs: indeed, it was designed to be Turbo-Pascal and Delphi compatible from the very start. Furthermore it works on a ever-growing number of platforms: Linux, BSD, Windows, DOS, OS/2, Netware, QNX, Solaris, BeOS, Amiga. Support for various CPU types is also growing: Intel, Motorola 68000 are considered stable, PPC and SPARC are works in progress. The Run-Time library provides basic functionality (file handling, string manipulation etc.) in a system-independent way. Lastly, Free Pascal comes with a lot of units that provide access to various libraries commonly available on Unix or Windows systems: notably, GTK.

The Free Pascal compiler comes with the FCL (Free Component Library) which is essentially a set of low-level all-purpose classes, similar to the ones found in the VCL. They are non-visual (i.e. independent of any GUI system): streaming, parsers, XML support,

database support, process control, recently also XML-RPC support etc. All this functionality is available for use in Lazarus.

On top of the FCL, the Lazarus Class Library (LCL) is built: this is a set of classes for visual programming: checkboxes, Menus, Edit controls. Built to resemble the VCL (Visual Class Library, delivered with Delphi) as closely as possible . These classes obviously depend on the operating system, and more importantly, the graphical system available.

This system dependency is covered by the so-called 'system interface': a system of units that must provide a well-defined interface in pascal, hiding the system dependencies from the higher-level classes. It is not object oriented in nature, and works with a system closely resembling the windows messaging system, but is independent from the latter.

Currently, two such interfaces are developed:

1. A GTK interface. This one is the most complete, and works very well on Linux or various BSD flavours. It should also work on Windows, but some bugs in the GTK libraries still prevent the IDE from fully working.
2. A native Win32 interface. This is still under development, however several demo applications are reportedly working.
3. The foundations of a GNOME interface have also been laid.

In the distant past, a QT interface was started, but has been abandoned. Other layers could be implemented using Motif as the widget set, or FLTK, or even a native X layer could be made: all that is needed is (lots of) time and enthousiasm.

The object-oriented layer - the actual LCL classes - uses only functions from this abstraction layer, and does not use any direct GUI system calls. All visual classes are built using this abstraction layer, and so once the abstraction layer is built, the whole of lazarus can be ported.

Note that using the Free Pascal Compiler and LCL, one could easily create applications without needing to use the Lazarus IDE: everything can be done on the command-line. However, the IDE has some attractive features, which make it worth using.

3 The Lazarus IDE

On top of the LCL and FCL classes, the Lazarus IDE (Integrated Development Environment) is built. It consists of mainly two parts:

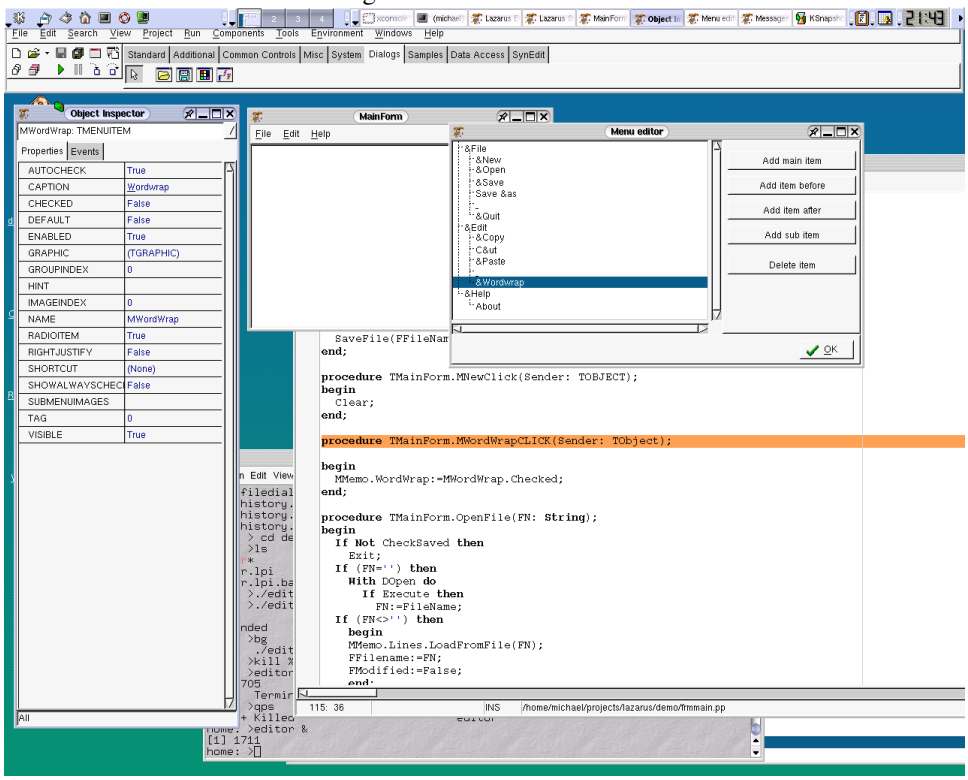
1. An advanced code editor.
2. A form designer.

In figure 1 on page 3 the IDE is shown, designing a simple editor application.

Both parts of the IDE interact heavily. Delphi developers will feel at home at once: the Lazarus code editor offers more than the standard Delphi functionality, a set of functionalities called the 'Code tools'. This functionality is offered by the excellent synedit components, which have been ported to Lazarus specially for this purpose. Among the features are:

- Syntax highlighting.
- Code completion: select the desired function call from a list.

Figure 1: The Lazarus IDE



- Tooltips showing required parameters.
- Class completion.
- Complete customizable keybinding.

And many more things, too much to be enumerated completely. The code editor by itself is enough reason to use lazarus as a development environment, even if none of its visual designing capacities are used: it offers all one can expect in a modern code editor.

The visual form designer works as in Delphi: forms ('Windows') are created, components can be selected from the component palette and dropped on the form. The 'Object Inspector' allows to set various properties using point-and-click techniques: select the desired property value from dropdowns, start property editors such as the menu editor (also shown in the above figure) etc.

Assigning code to run-time events, e.g. the user clicks a button, is also easy: By clicking on the 'OnClick' event in the object inspector, a method is created in the editor, and the cursor is positioned to start typing the code of the method. Nothing exciting for seasoned Delphi users, maybe more so for emacs users.

Press CTRL-F9 to compile the code, or F9 to compile and run: Lazarus does it all. As can be expected in an IDE, all FPC options can conveniently be set from various dialogs. Integrated debugging is also possible: breakpoints can be set by selecting the line where program execution should stop. (provided debug-info was compiled in the application, of course).

To demonstrate all this, a small demo application was created and compiled: a small notepad replacement. Its sources can be found on the CD-ROM accompanying this issue. Most of the code in this application (about 200 lines) is generated by the IDE: only

procedure bodies have to be typed explicitly.

4 Extensible

Lazarus can be extended. People familiar with Delphi know that they can dynamically load a package in the IDE which extends the functionality: either it extends the IDE itself, providing wizards to do things, or they put new components on the component palette, so they can be used in applications.

Lazarus does not yet provide this dynamically-loadable package functionality: the underlying Free Pascal Compiler does not (yet) allow such packages to be created, so Lazarus obviously cannot use them. An alternative packaging system is being worked on, which will be used till the time that the FPC compiler fully supports packages.

However, packages are not really needed: since the sources of lazarus are available, the IDE itself can be rebuilt at will. This means that functionality extending the IDE can be coded straight in the IDE, something which would be rather hard to do in Delphi.

Adding additional components to the component palette is a matter of adding an entry in a unit specially maintained for this purpose, and rebuilding the IDE. The IDE even contains special menu entries to facilitate recompilation of itself.

Apart from this, the IDE can be extended using external tools: an 'External tool' dialog allows to register extra programs. These extra programs then get a menu entry in the lazarus menu.

5 A work in progress

Lazarus can be used to design real-world applications. The *piece de resistance* at this moment is the checkbook tracker by Tony Maro: a kind of 'Microsoft Money' application to manage bank accounts (shown in figure 2 on page 5). It is designed completely in Lazarus, and hence - apart from its inherent usefulness as a checkbook tracker - shows that Lazarus can already be used as a complete development solution; As a testimony of this statement, it should be said that Lazarus will be included on a E-learning CD-ROM to be distributed in South Africa.

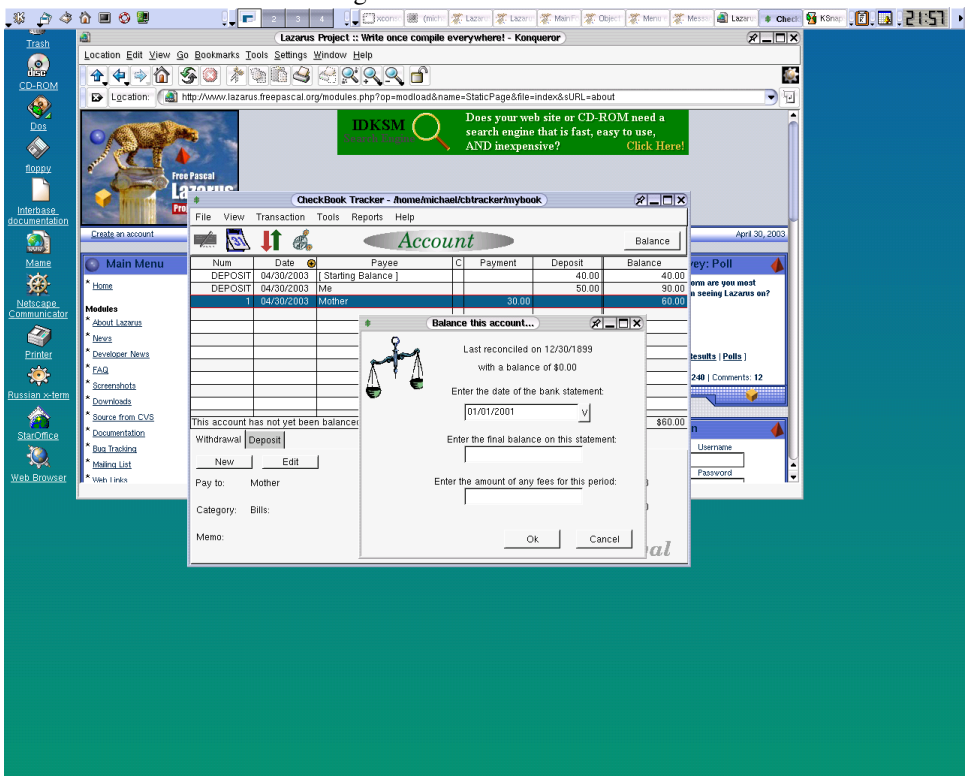
Despite this, Lazarus is far from finished. Its usefullness depends on the quality and quantity of the components available. While enough components are available to create good-looking and functional programs, there is still a lot to be done. The underlying FCL needs work, the LCL can be extended with much more visual controls, Database aware controls should be created, more web tools should be developed. The Win32 interface should be finished, maybe the Qt interface should be restarted.

The IDE itself is still not perfect: images for buttons and menu items must still be loaded in code because they cannot be streamed, more wizards could be introduced. Documentation is also needed: all these issues are things which have been started, but are progressing slowly. Lack of manpower is the main obstacle which prevents lazarus from becoming the killer environment which it could be.

6 Getting started

People wanting to give lazarus a try should start by installing the Free Pascal compiler, available from the Free Pascal website:

Figure 2: Checkbook tracker



`http://www.freepascal.org/`

Binary installs for various platforms are available. To make full use of the code tools in the Lazarus editor, the sources of the run-time library and FCL should also be installed.

After installing the compiler, Lazarus itself must be downloaded and installed. The Lazarus project has a separate website (with lots of screenshots):

`http://www.lazarus.freepascal.org/`

It also provides newer versions of the Free Pascal Compiler which can be used to compile Lazarus itself. Here also, the sources are best installed.

The lazarus IDE should be installed and configured: the location of all sources should be indicated, so the codetools can be used. A document describing the installation and configuration procedure in detail can be found on the Lazarus website. As with any self-respecting Open source project, help can be obtained from the lazarus forum or the mailing list. Bugs can be reported through the on-line bugtrackers.

7 Conclusion

Lazarus is a promising open-source project, definitely worth a look for people that know Delphi and are looking for an open-source alternative. Being short on manpower, help is more than welcome: the number of people actively developing Lazarus can be counted on the fingers of one hand. Looking at what has already been accomplished, Lazarus can definitely be called an open source gem.