Comparing the free .NET implementations Mono and DotGNU

LOTS OF DOTS

The Open Source projects Mono and DotGNU open Microsoft's .NET platform to Linux developers. This article investigates how well-suited these alternative implementations are for building simple GUI applications. **BY MICHAEL TSCHATER**

In 2002, Microsoft launched a new system for platform-independent development that they dubbed .NET (pronounced "Dotnet"). The .NET project aims to simplify the development of network and Internet applications. It supports object oriented programming and comes with a unique class library suitable for use with multiple programming languages, such as C# and VB .NET for example. In the course of launching .NET, Microsoft has discontinued support for its previous favorite, the MFC library for Visual C++. All future Microsoft products will be based on the new technology.

Free .NET?

Despite the platform-independent approach of the .NET framework, Microsoft is highly unlikely to release a Linux implementation, and this lack of Linux support has prompted two projects to step into the gap: Novell's sponsored Mono project [1] and the open source DotGNU [2] project. Both Mono and DotGNU are aiming to provide as complete an implementation of the .NET platform as possible. The goal of both projects is to allow developers to exchange software across operating system boundaries.

Besides basic functionality such as string manipulation, most projects need widgets for graphical user interfaces to help them cope with day-to-day tasks. Microsoft offers the *System.Windows.Forms* library for this purpose. The library has GUI objects, such as *MainMenu*, *ToolBar*, *Combobox*, or *Button*. Listing 1 has a short code snippet as an example.

To compare the Mono and DotGNU implementations, we will be using a program written on Windows. This short example has all the major GUI elements, see Figure 1. The source code is available on the Linux Magazine website at [3]. The only functionality this code actually provides is that it logs user input in a window specially designed for that purpose.

The Original by Microsoft

One way of obtaining the original Microsoft .NET is the free .NET runtime environment and SDK (Software Development Kit) download. The download gives you all the command line tools needed for software development on Windows, just like the Java SDK. The *csc* command compiles an application. The results can be executed directly on Windows.

The Integrated Developer Environment (IDE) Visual Studio .NET is a com-

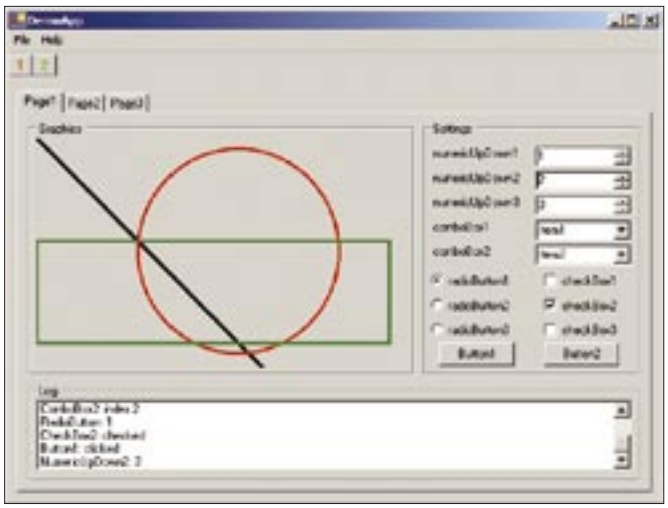

Figure 1: The demo application includes all major GUI elements from the .NET System.Windows.Forms library. This figure shows the Windows version.

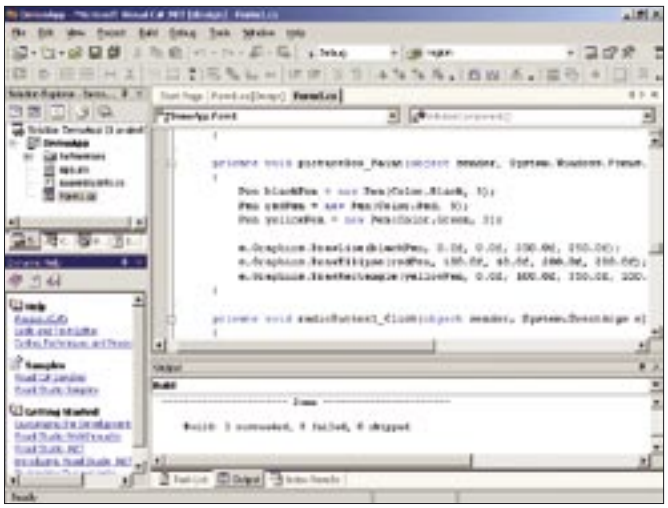

Figure 2: Visual Studio .NET displaying the code from the sample project described in this article. The code in the figure draws graphical elements.

mercial alternative for .NET development that includes a debugger and dialog editor (Figure 2). This package automatically installs the .NET SDK. The free Sharpdevelop developer environment [4] is an excellent alternative to the Visual Studio .NET package. Sharpdevelop gives you a powerful IDE that only lacks an integrated debugger. The features include the ability to toggle between the Microsoft and Mono runtime environments and to import Visual Studio .NET projects.

The Alternative: Mono

The Mono environment [5] is available for Linux, Windows, and Mac OS. Besides the .NET standard class libraries, Mono has a few extensions, such as a variety of Gnome libraries. The virtual machine adds just-in-time and ahead-of-time build techniques for improved speed. Mono uses GTK# (which is also available for Windows) for graphical user interfaces. GTK# is based on GTK+, which forms the basis of the Gnome Desktop on Linux.

GUI with Windows.Forms

The *System.Windows.Forms* library is mainly implemented in C# and draws its own controls [6]. The library requires a driver for each operating system: drivers for X11 and Windows are available at this time of writing. The DotGNU project has adopted this design after ditching a Wine-based implementation as impracticable.

Besides the *mcs* compiler and the *mono* runtime environment, the Linux version also has the *monodevelop* developer environment, which is based on a Linux port of Sharpdevelop. The *monodoc* help browser, which allows programmers to update documentation online from the Mono repository, is another useful feature. Besides distribution specific binaries, there is an executable installer, which sets up all the required packages.

After you download the 50MByte package, a wizard walks you through the installation procedure, which just involves a few steps. Thanks to the wizard, the installation problems that once affected explicitly supported, but non-current, distributions (such as Fedora) are a thing of the past. Some distributions – such as Suse 9.3 – include Mono by default, but distros such as Red Hat and Fedora tend to place their emphasis instead on the Java platform.

There are a number of websites, such as the one at [5], that document the progress of the Mono project in detail. For example, you will find an overview of the controls in the *System.Windows.Forms* library at [6].

Listing 1: GUI with System.Windows.Forms

```
01 #using System;
02 #using System.Windows.Forms;
03 #
04 #namespace LinuxMagazin.
   FirstApp
05 #{
06 #  public class FirstApp :
   System.Windows.Forms.Form
07 #  {
08 #    public static void Main()
09 #    {
10 #      Application.Run(new
   FirstApp());
11 #    }
12 #
13 #    public FirstApp() :
   base()
14 #    {
15 #      this.Text = "First
   Application with GUI";
16 #    }
17 #  }
18 #}
```

A Sample Application with Mono

Existing Visual Studio .NET projects such as our sample application can be imported directly into Monodevelop. But don't forget to adjust the references for the new project. The context-based help when editing source code is one thing that you immediately notice about the developer environment, but the environment lacks a GUI builder at this time of writing.

When we tried to build the code by giving the *mcs -r:System.Windows.*

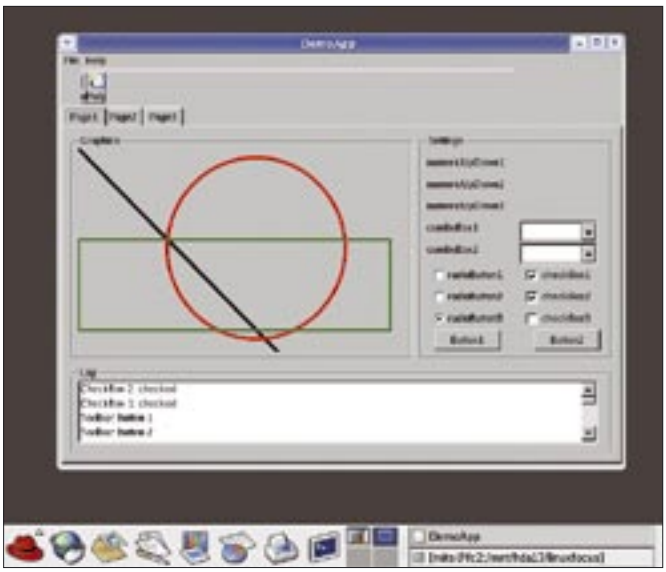

Figure 3: The sample application running in Mono on Linux has difficulty drawing the buttons at the top and the selection boxes on the right.

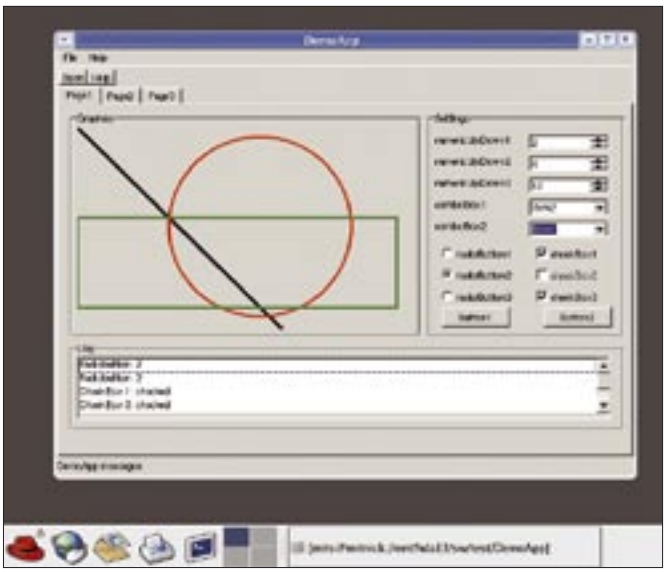

Figure 4: The GUI-based .NET application has a few issues running in DotGNU; for example, the radio buttons falsely trigger events multiple times.

Forms.dll sourcefile command, the standard *System.Windows.Forms.NumericUpDown* control caused a problem, as it is not completely implemented in Mono as yet. After removing the offending code, we were able to build the application and launch it in the *mono* runtime environment (Figure 3).

A number of errors occurred: for example, the update mechanism, which handles redrawing in the application, did not work properly when hidden by another application. Additionally, the event mechanism had a some trouble with the *RadioButton* and *ComboBox* controls.

The Contender: DotGNU

Based on the 0.7 version number assigned by the project, DotGNU is less mature than Mono. The core component of DotGNU is titled Portable .NET. It comprises three major components: *treecc* (an aspect-oriented programming tool), *pnet* (runtime environment, C# compiler, programming tools), and *pnetlib* (C# system library with critical C# class libraries).

Source code and RPM packages are available from the DotGNU website. The RPM variant did not require any tweaking to give us a fully functional environment; the tools all worked at first asking. Current distributions such as Debian 3.1 make things even easier by actually including the DotGNU environment by default (in the *pnet* package). The following commands build and launch the HelloWorld application:

```
cscc -o HelloWorld.exe ➥
HelloWorld.cs
ilrun HelloWorld.exe
```

The following command sequence builds a more complex project with a graphical interface. The source code files are located in the *src* subdirectory:

```
cscc -winforms -o ➥
Application.exe src/*.cs
ilrun Application.exe
```

The implementation of the *System.Windows.Forms* library is not based on a graphical library such as GTK, but instead it draws the controls itself, just like the Java Swing library. DotGNU does not implement an IDE of its own.

The Sample Project in DotGNU

DotGNU runs the bytecode generated on Windows without any modifications. However, clicking on the *RadioButton* controls initiated the event routine multiple times. This did not change when we rebuilt the source using DotGNU: *cscc --winforms -o DemoApp.exe *.cs*. Additionally, we had to remove the *ImageList* handling code to get the application to build.

Our tests managed to uncover another bug in the variant compiled with DotGNU (Figure 4). Our tests indictate that

Table 1: Comparing .NET Implementations

Project	DotGNU	Mono
Current Version	0.7.0	1.1.8
Copyright	LGPL	GPL or proprietary (selectable)
Operating systems	Linux, Mac OS X, Windows	Linux, Mac OS X, Windows
Advantages		
	Standard package with Debian	Standard package with Suse 9.3
	Original Windows bytecode can be launched without modifications	Supported by full-time developers at Novell
Disadvantages		
	System.Windows.Forms not completely implemented	System.Windows.Forms not completely implemented
	Event handling error	Event handling error
	No developer environment	Installation packages (Wizard) not adequately tested; for example, the developer environment is missing in Mono 1.1.7

the event mechanism does not work properly: the GUI element *RadioButton* triggered duplicate events. *Numeric UpDown* doesn't work at all and is not displayed correctly.

The *GroupBox* control changes color to indicate the current focus. This behavior does not conform to the standards. The *ImageList* control prevents the application from being displayed. DotGNU will display the executable created using Mono, but the icons in the control bar buttons are faulty.

A clause in the Microsoft .NET license prohibits the publication of comparative benchmarks for .NET implementations. However, the DotGNU project provides the PNetmark tool [7] to help those of you interested in benchmark testing to create your own benchmarks. For more information on performing your own benchmarks, check out Section 1.4 of the DotGNU FAQ.

Summary

Microsoft's .NET framework provides an interesting approach to platform-independent programming. The C# programming language is very similar to the Java and C++ programming languages, and that makes it easy for developers who are familiar with these programming languages to learn C#. Another advantage that C# gives you is ECMA standardization.

The two open source implementations of the .NET framework, Mono and DotGNU, both have impressive feature sets, but they do have some rough edges too. The developers of these tools still have a long way to go before they achieve complete compatibility. This said, both .NET and Mono succeed in providing support for .NET software development on Linux.

We mainly experienced issues using the *System.Windows.Forms* library, which provides graphical applications on Windows. In other words, it is not trivially possible to migrate GUI-based Windows .NET software to Linux right now.

As an alternative to direct migration of GUI-based Windows .NET software to Linux, you can use GUI libraries such as GTK#, Qt#, or WX#. Because all of these libraries are also available for the Windows platform, cross platform applications are possible within limits, although these applications will not be .NET compatible. ■

INFO

[1] Mono: *http://www.mono-project.com*

[2] DotGNU: *http://www.dotgnu.org*

[3] Listings online: *http://www.linux-magazine.com/Magazine/Downloads/2005/12/dotNET/*

[4] Sharpdevelop homepage: *http://www.icsharpcode.net*

[5] Mono development status: *http://www.go-mono.com/mono-roadmap.html*

[6] System.Windows.Forms development status at Mono: *http://svn.myrealbox.com/mwf/owners.html*

[7] PNetmark: *http://www.southern-storm.com.au/pnet_faq.html#q1_4*