

Mono - die freie Implementierung der .Net Plattform

7. Juni 2004

Rechtlicher Hinweis

Dieser Beitrag ist lizenziert unter der Creative Commons License.

Zusammenfassung

Mono ist eine freie Implementierung der von Microsoft im Rahmen des .Net Development Frameworks vorgeschlagenen ECMA- und ISO-Standards.

Der Vortrag gibt einen Überblick über den aktuellen Stand der Entwicklung der im Rahmen des Mono-Projektes entwickelten Komponenten C#- und VisualBasic.Net-Compiler, Runtime und Klassenbibliothek. Ferner werden Werkzeuge wie Debugger und Profiler sowie Tools für Build Management und Unit Testing vorgestellt. Darüber hinaus wird die Integration von Mono mit bereits etablierten Open-Source-Projekten wie dem Apache-Webserver oder dem Gnome-Desktop behandelt.

1 Einleitung

Mit .Net ist es Microsoft ¹ das erste Mal gelungen, Open-Source-Entwickler von den technischen Vorzügen eines Produkts zu überzeugen. Unter Leitung des Gnome-Initiators Miguel de Icaza ² läuft seit Juli 2001 die Arbeit an Mono ³, einer freien Implementierung der von Microsoft im Rahmen des .Net Development Frameworks vorgeschlagenen ECMA- und ISO-Standards.

Mit den Worten "Bitte vergessen Sie alles, was sie bislang über .Net gehört haben" wandte sich de Icaza im Februar 2002 an die Gnome-Benutzer und -Entwickler. Sie sollten sein Kind ohne Vorurteile in Augenschein nehmen. Damals war das von seiner Firma Ximian ⁴ (die mittlerweile von Novell ⁵ gekauft wurde) initiierte Mono-Projekt ein gutes halbes Jahr alt. Nach dieser relativ kurzen Zeit hatten die Mono-Entwickler bereits einige Erfolge vorzuweisen. So war der in C# geschriebene C#-Compiler *mcs* in der Lage, sich selbst zu übersetzen. Im Folgenden wird es um die Frage gehen, wie weit die Mono-Plattform in den letzten Jahren gekommen ist. Gegenstand der Betrachtung ist die erste Beta-Version von Mono 1.0.

Warum interessiert sich die Open-Source-Gemeinde für eine Portierung der Microsoft.Net-Technologie auf freie Betriebssysteme? Microsoft verfolgt mit seiner .Net-Strategie die Konsolidierung der Programmierschnittstellen der Windows-Plattform. Diese Programmierschnittstellen sollen einheitlich von möglichst vielen Programmiersprachen aus genutzt werden.

Die Open-Source-Gemeinde hat ähnliche Probleme mit "gewachsenen" Programmierschnittstellen. So nutzt das GNOME-Projekt ⁶ derzeit beispielsweise eine CORBA ⁷-basierte Lösung, um mehreren Programmiersprachen den Zugriff auf die GNOME-Schnittstellen zu ermöglichen. Die von Microsoft.Net propagierte Sprachunabhängigkeit ist eine feine Sache, die man sich (ebenso wie die von Java ermöglichte Plattformunabhängigkeit) auch für die UNIX-Welt wünscht. Darüber hinaus ermöglicht eine Portierung der Microsoft.Net-Technologie die Ausführung von für Microsoft.Net entwickelten "Windows-Anwendungen" unter UNIX.

Bereits heute gibt es eine Vielzahl von Anwendungen für den GNOME-Desktop, die beispielsweise in C# programmiert werden und die GNOME-Schnittstellen über die entsprechenden Mono-Bindings wie beispielsweise Gtk# ⁸ verwenden. Obwohl diesbezüglich noch keine offizielle Entscheidung getroffen wurde, ist abzu-

¹<http://www.microsoft.de/> <<http://www.microsoft.de/>>

²<http://primates.ximian.com/~miguel/> <<http://primates.ximian.com/~miguel/>>

³<http://www.go-mono.com/> <<http://www.go-mono.com/>>

⁴<http://www.ximian.com/> <<http://www.ximian.com/>>

⁵<http://www.novell.com/> <<http://www.novell.com/>>

⁶<http://www.gnome.org/> <<http://www.gnome.org/>>

⁷<http://www.corba.org/> <<http://www.corba.org/>>

⁸<http://gtk-sharp.sourceforge.net/> <<http://gtk-sharp.sourceforge.net/>>

sehen, dass Mono mittelfristig ein Kernbestandteil des GNOME-Projektes werden wird. Wichtigstes Argument hierfür dürfte die leichtere Entwicklung von Anwendungen in C# gegenüber C oder C++ sein.

2 Die Mono-Plattform

Die Mono-Plattform besteht, ebenso wie die Microsoft.Net-Plattform, aus mehreren Bestandteilen. Die wichtigsten sind

- *mcs* , ein Compiler für die neue Programmiersprache C#, die in vielen Aspekten Ähnlichkeit mit Java und C++ hat.
Der Compiler *mcs* "versteh" Programme gemäß Version 1.0 der C# Sprachspezifikation, sowie mit den Iteratoren eines der neuen Sprachmerkmale von C# 2.0, das Microsoft bereits angekündigt hat.
- *mbas* , ein Compiler für VisualBasic.Net.
- *ilasm* , ein Assembler, der aus Microsoft Intermediate Language (MSIL) Quellen so genannte Portable Executables (PE) generiert.
- Eine *Common Language Runtime (CLR)* konforme virtuelle Maschine, die Bytecode der *Common Intermediate Language (CIL)* ausführt.
Diese Laufzeitumgebung ist in Version 1.0 der Mono-Plattform konform mit Version 1.1, einem Bugfix Release, das vor allem der Konsistenz der Programmierschnittstellen dient, der Microsoft.Net-Plattform.
- Eine Klassenbibliothek, die ein breites Spektrum an Funktionalität von Dateisystemoperationen und Datenbankschnittstellen bis zu Remoting und Verarbeitung von XML abdeckt.

Die Mono-Plattform bietet zwei verschiedene Implementierungen der .Net-Laufzeitumgebung. Die eine arbeitet nach dem Just-in-Time (JIT) Prinzip und übersetzt die Opcodes des .Net-Bytecodes in die Maschinensprache des Rechners, auf dem Mono ausgeführt wird. Im Gegensatz hierzu interpretiert der Mono Interpreter (*mint*) den .Net-Bytecode und kann so nicht die Performanz der JIT-Variante erreichen, lässt sich im Gegenzug aber wesentlich einfacher auf neue Plattformen portieren.

Seit Version 0.24 bietet Mono die Ausführung in einem Ahead-of-Time (AOT) getauften Modus an. Hierbei werden die kostenintensive Übersetzung und Optimierung der Opcodes nur einmalig durchgeführt. Aber Achtung: Die in diesem Schritt erzeugten Binaries sind nicht mehr plattform-unabhängig, da sie Prozessor-spezifischen Maschinencode enthalten.

Mono 1.0 unterstützt die folgenden Plattformen:

- *Just-in-Time* : PowerPC, (S390), Sparc und X86.
- *Interpreter* : AMD64, HPPA, IA64, S390, StrongARM.

3 Von Produzenten und Konsumenten

Compiler, die .Net-Bytecode für Programmiersprachen wie Basic, Cobol, Fortran, Eiffel oder Smalltalk erzeugen, bieten Entwicklern Unabhängigkeit von der Programmiersprache: In einer Sprache entwickelte Komponenten können von einer anderen aus benutzt werden. Jene Idiome, für die ein geeigneter Compiler .Net-Bytecode erzeugen kann, heißen ".Net-Producer"; solche, die .Net-Komponenten verwenden können, nennt man ".Net-Consumer".

Bettet eine Anwendung die Mono-Laufzeitumgebung ein, verwandelt sie sich in einen .Net-Consumer. Sie kann einfach und direkt mit Komponenten interagieren, die im .Net-Bytecode vorliegen. So kann eine Applikation zum einen eine einheitliche Programmierschnittstelle (API) bekommen, über die sie sich in jeder Sprache, die als .Net-Producer fungieren kann, erweitern lässt.

Zum anderen erweitert eine eingebettete Mono-Laufzeitumgebungen bestehende Applikationen so, dass sie als .Net-Consumer in Aktion treten können. Für PHP 5⁹ existiert bereits eine Mono-Integration, die es erlaubt, .Net-Komponenten durch Überladung wie "normale" PHP-Objekte zu benutzen.

Im IKVM-Projekt¹⁰ arbeiten Open-Source-Entwickler an einer Java Virtual Machine (JVM)¹¹ für die .Net-Plattform. Diese Implementierung ist bereits so weit gediehen, dass selbst komplexe Java-Applikationen wie Eclipse¹², JBoss¹³ oder Jython¹⁴ in einer von der Mono-Laufzeitumgebung ausgeführten JVM ausgeführt werden können.

4 ASP.Net-Webanwendungen mit Apache und Mono

Bei der Einführung von .Net aktualisierte Microsoft seine Active Server Pages, die nunmehr ASP.Net¹⁵ heißen und nicht mehr auf interpretiertes VisualBasic beziehungsweise JScript beschränkt sind. In der neuen Variante kann man sie in C#, VisualBasic.Net und JScript.Net sowie jeder Programmiersprache entwickeln, die als .Net-Producer agiert. Microsoft selbst empfiehlt zwar als Webserver eine für die .Net-Plattform ausgelegte Version des hauseigenen IIS, doch dank Mono und einem entsprechenden Modul für den Apache 2.0 Webserver¹⁶ lassen sich Active Server Pages auch unter UNIX ausführen.

Beispiel 4.1 zeigt einen in C# geschriebenen dynamischen Kalender für ASP.Net, **Abbildung 1** die entsprechende Ausgabe im Webbrowser.

Beispiel 4.1: ASP.Net-Beispiel

```
<%@ Page Language = "C#" %>
<html>
  <head>
    <title>Testing properties in inner tags</title>
  </head>
  <body>
    <form runat="server">
      <h3>Calendar and properties</h3>
      <asp:calendar id="Calendar1" Font-Name="Arial" showtitle="true" runat="server">
        <SelectedDayStyle BackColor="Blue" ForeColor="Red" />
        <TodayDayStyle BackColor="#CCAACC" ForeColor="Black" />
      </asp:Calendar>
    </form>
  </body>
</html>
```

⁹<http://www.php.net/> <<http://www.php.net/>>

¹⁰<http://www.ikvm.net/> <<http://www.ikvm.net/>>

¹¹<http://java.sun.com/docs/books/vmspec/> <<http://java.sun.com/docs/books/vmspec/>>

¹²<http://www.eclipse.org/> <<http://www.eclipse.org/>>

¹³<http://www.jboss.org/> <<http://www.jboss.org/>>

¹⁴<http://www.jython.org/> <<http://www.jython.org/>>

¹⁵<http://www.asp.net/> <<http://www.asp.net/>>

¹⁶<http://httpd.apache.org/> <<http://httpd.apache.org/>>



Abbildung 1: ASP.Net-Beispiel

5 GUI-Toolkits

Für die Entwicklung von grafischen Benutzeroberflächen (GUIs) bietet sich das vom GNOME-Projekt bekannte Gimp-Toolkit (Gtk+) ¹⁷ an. Es steht dank der Gtk#-Bibliothek für Mono-Anwendungen zur Verfügung. Für die GUI-Erstellung mit der .Net-Entwicklungsplattform sieht Microsoft die Klassen des *Windows.Forms*-Namespaces vor. Da diese jedoch auf den Windows-APIs basieren, gestaltet sich eine Portierung damit entwickelter Software schwierig. Eine Implementierung des *Windows.Forms*-Namespaces auf Basis des Wine-Projektes ¹⁸, welches eine Portierung der Windows-APIs für X11 ¹⁹ zum Ziel hat, wird in Mono 1.0 nur in einer „unstable“ Version enthalten sein. Diese soll für Mono 1.2, das für das vierte Quartal 2004 angekündigt ist, komplettiert werden.

Wie eine in C# geschriebene Gtk#-Anwendung aussieht, zeigt *HelloWorld.cs* ([Beispiel 5.1](#)). Es wird mit dem C#-Compiler von Mono wie folgt übersetzt:

```
mcs /r:gtk-sharp.dll /r:gdk-sharp.dll /r:glib-sharp.dll HelloWorld.cs
mono HelloWorld.exe
```

¹⁷<http://www.gtk.org/> <<http://www.gtk.org/>>

¹⁸<http://www.winehq.org/> <<http://www.winehq.org/>>

¹⁹<http://www.x.org/> <<http://www.x.org/>>

Beispiel 5.1: Gtk#-Beispiel

```
using Gtk;
using GtkSharp;
using System;

class HelloWorld {
    static void Main() {
        Application.Init();

        Window window = new Window("Hello World!");
        window.Show();

        Application.Run();
    }
}
```

Mono treibt hier die Nähe zum Vorbild so weit, dass es die Windows-übliche Konvention übernimmt, Parameter mit / einzuleiten. Wichtig ist, bei der Übersetzung von Gtk#-Anwendungen stets die Gtk#, Gdk#- und Glib#-Assemblies über den /r Parameter einzubinden. *mono HelloWorld.exe* führt das gerade übersetzte Gtk#-Beispiel aus; es sollte sich ein leeres Fenster (**Abbildung 2**) öffnen.

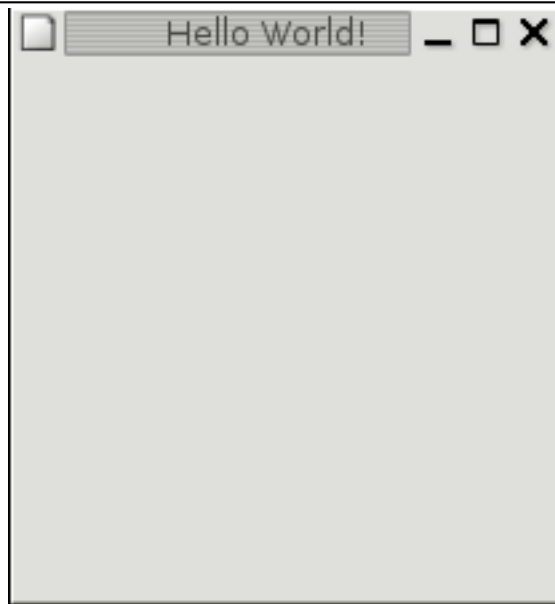


Abbildung 2: Gtk#-Beispiel

Neben Gtk# und Windows.Forms gibt es mit Sharp WT ²⁰ (basierend auf dem Standard Widget Toolkit

²⁰<http://sourceforge.net/projects/sharpdevelop/> <<http://sourceforge.net/projects/sharpdevelop/>>

(SWT)²¹, das unter anderem von Eclipse genutzt wird), Qt#²² (basierend auf Qt²³) und wxNet²⁴ (basierend auf wxWidgets²⁵) weitere .Net-Bindings für Open Source GUI-Toolkits, die aus Mono heraus genutzt werden können.

6 Werkzeuge

Für die Entwicklung von Software ist ein Compiler – und im Falle von Software für die .Net-Plattform eine entsprechende Laufzeitumgebung – zwar notwendig, jedoch noch lange nicht ausreichend. Entwickler benötigen weitere Werkzeuge, um beispielsweise die Projektdateien zu übersetzen (Build Management), die korrekte Ausführung von Klassen und Methoden zu testen (Unit Testing), Fehlern in der Codebasis auf den Grund zu gehen (Debugger) oder .Net Assemblies zu disassemblieren.

Mit dem ebenfalls in C# geschriebenen *Mono-Debugger* lassen sich Fehler in .Net-Anwendungen sowohl auf der Kommandozeile als auch mit einem Gtk#-basierten Frontend lokalisieren. Das Build Management kann man getrost *NAnt*²⁶, einer Portierung der Java-Standardlösung *Ant* anvertrauen. Ebenfalls eine Entlehnung aus der Java-Welt stellt der JUnit-Nachbau *NUnit*²⁷ dar, dessen Kommandozeilenversion beispielsweise in das von NAnt verwaltete Build Management integrierbar ist. Es wartet ebenfalls mit einem Gtk#-basierten Frontend auf.

Mit MonoDevelop²⁸ (**Abbildung 3**) steht eine in C# geschriebene, auf Gtk#, Mono-Debugger und Monodoc basierende Entwicklungsumgebung zur Verfügung, die bereits in der aktuellen Entwicklerversion 0.2 recht brauchbar erscheint.

²¹<http://www.eclipse.org/articles/Article-SWT-Design-1/SWT-Design-1.html>

<<http://www.eclipse.org/articles/>

[Article-SWT-Design-1/SWT-Design-1.html](http://www.eclipse.org/articles/Article-SWT-Design-1/SWT-Design-1.html)>

²²<http://qtsharp.sourceforge.net/> <<http://qtsharp.sourceforge.net/>>

²³<http://www.trolltech.com/products/qt/> <<http://www.trolltech.com/products/qt/>>

²⁴<http://wxnet.sourceforge.net/> <<http://wxnet.sourceforge.net/>>

²⁵<http://www.wxwidgets.org/> <<http://www.wxwidgets.org/>>

²⁶<http://nant.sourceforge.net/> <<http://nant.sourceforge.net/>>

²⁷<http://www.nunit.org/> <<http://www.nunit.org/>>

²⁸<http://www.monodevelop.com/> <<http://www.monodevelop.com/>>

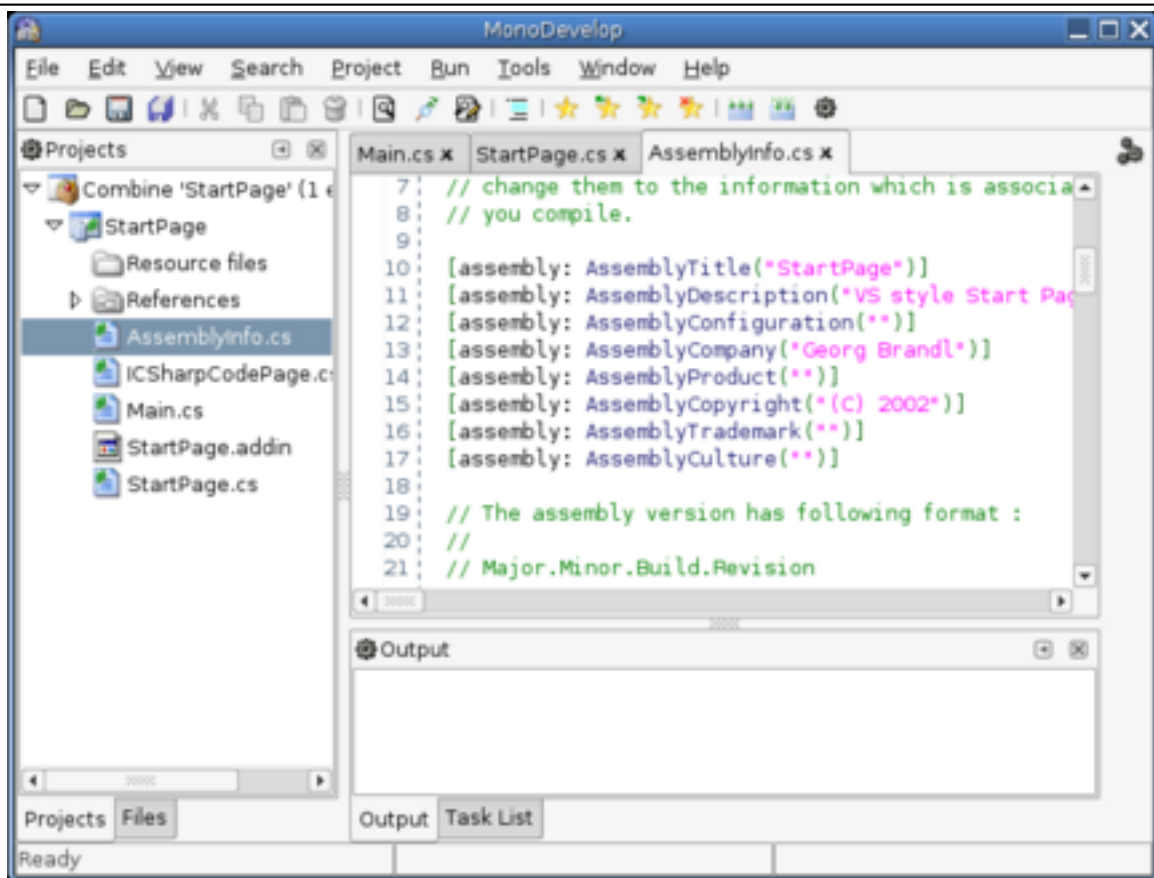


Abbildung 3: MonoDevelop