Tcl .NET Integration

Artur Trzewik
mail@xdobry.de

Sixth European Tcl/Tk Users Meeting
Bergisch Gladbach - 2006

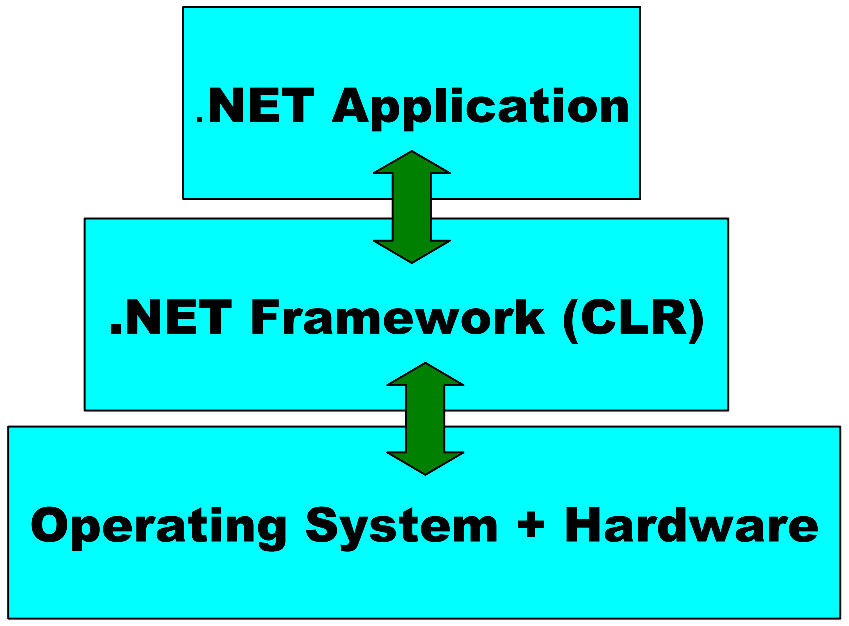
Agenda

- Short .NET Introduction
- Using Tcl-dll from .NET per Pinvoke
- Using .NET from Tcl per COM
- Tcl interpreter in C# - TclCsharp

.NET Introduction

.NET is platform for

- building
- deploying
- running application



Runtime view – CLR = Virtual Machine

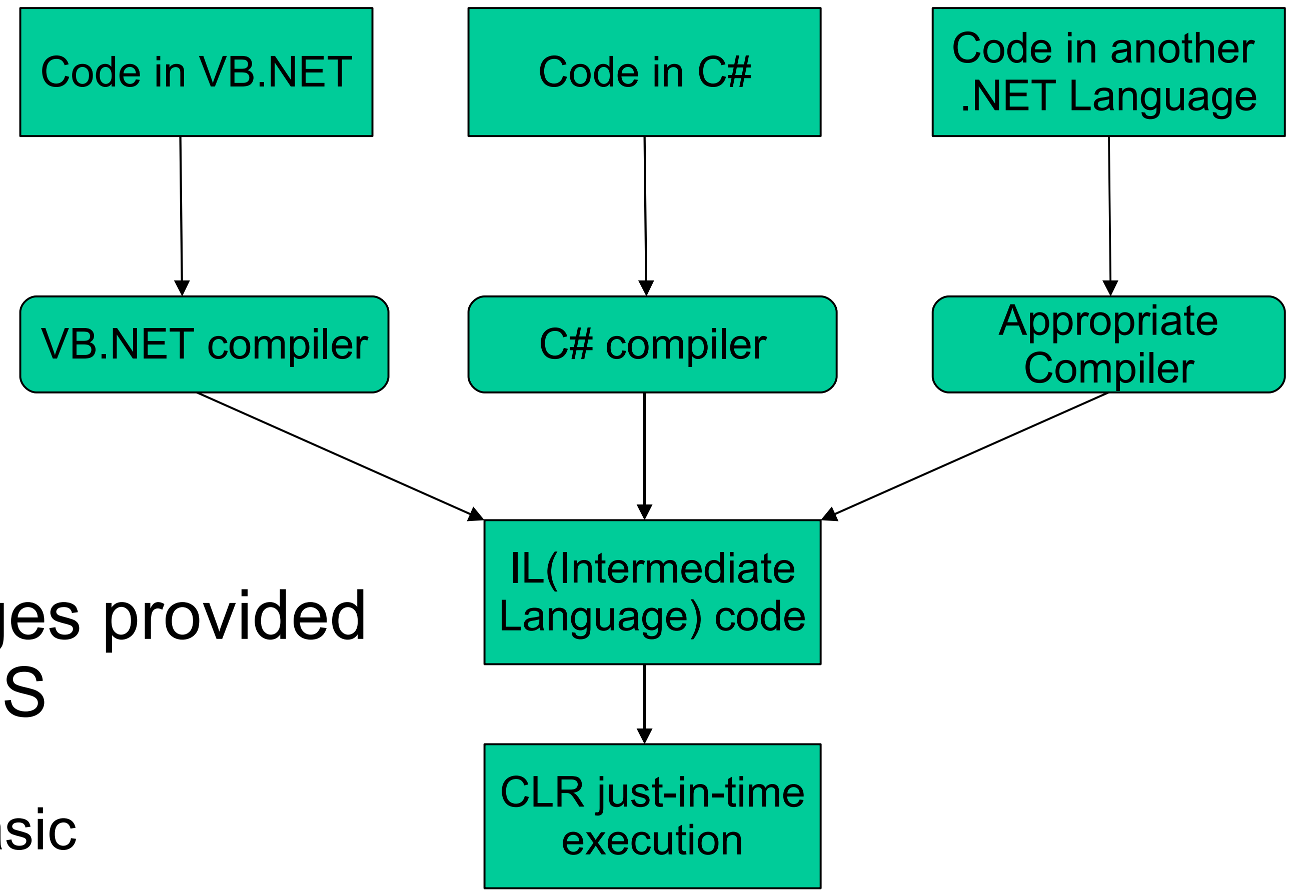
.NET focus

- RAD (Rapid Application Development)
- Language neutral
- Platform neutral (Hardware, .NET CE Compact Edition)
- Web-Application (ASP.NET)
- Web Services
- Distributed Application

.NET Elements

- CLR (Common Language Runtime – Virtual Machine)
- CLI (Common Language Infrastructure)
- Class Libraries (WinForm, XML, Remoting, Web, DB, ...)
- ASP.NET

.NET – language neutral


Code in VB.NET
Code in C#
Code in another .NET Language
VB.NET compiler
C# compiler
Appropriate Compiler
IL(Intermediate Language) code
CLR just-in-time execution

Languages provided by MS

- C#
- Visual Basic
- C++
- J#
- JScript

Common Language Infrastructure

- ECMA Specification for CLR
- Common Type System (int, char, ...)
- Language Integration
- Memory Management (Garbage Collector)
- Error Handling
- Threading
- CIL (Common Intermediate Language – Byte Code) MSIL (Microsoft intermediate language)

Compiling Tcl to .NET-Assembly

- Tcl does not fit in CLI (Common Language Infrastructure)
- Tcl – late distinguish between data and code (late compilation)
- Runtime Environment is needed (Tcl Interpreter)
- .NET provides libraries to produce byte code on the fly – Code Emit (used by regular expressions)

Scripting in .NET - IronPython

- Experimental implementation of script languages in .NET (Perl, Smalltalk, Python)
- IronPython – the most advanced, supports static and dynamic compilation (even faster than standard Python). Official supported by MS (download from Msdn)
- F# - OCalm (Functional language in .NET)
- .NET 3.0 offers Lambda-Expression (needed for RAD and OO-Mapper)

Using tcl.dll from .NET per Pinvoke

- Pinvoke easy direct access to old DLL
- No wrapper libs as by Java (JNI)
- Tcl is just C-Library (dll)

Base Example

```
public class TclAPI {
     [DllImport("tcl84.DLL")]
     public static extern IntPtr  Tcl_CreateInterp();
     [DllImport("tcl84.Dll")]
     public static extern int Tcl_Eval(IntPtr interp,string
   skript);
     [DllImport("tcl84.Dll")]
     public static extern IntPtr Tcl_GetObjResult(IntPtr interp);
     [DllImport("tcl84.Dll")]
     public static extern string    Tcl_GetStringFromObj(IntPtr
   tclObj,IntPtr length);
}
```

see http://wiki.tcl.tk/9563

Problems with PInvoke

- Memory management
- More integration
- Multithreading (EventLoop)
- Error management

Simple Tcl-Wrapper

- http://www.xdobry.de/tclinterop.zip
- Access to .NET objects from Tcl
- Callbacks
- Tcl Procs in .Net

Using .NET for Tcl per COM (tcom)

- Easy COM implementing per .NET
- Access to Windows internals
- Access to .NET libraries

Example .NET Com

```
using System;
using System.Runtime.InteropServices;
using System.Diagnostics;
using Microsoft.VisualBasic;
namespace ComTestAtk {
[ComClass]
public class ATKComm {
  public ATKComm() {}
  public void writeEventLog(string message)  {
      EventLog myLog = new EventLog();
      myLog.Source = "Anwendung";
      myLog.WriteEntry(message)
   }
}
}
```

COM registration

- Use regasm.exe (part of .NET SDK)
- No deploying by copy more
- Windows specific

Using Com from Tcl

```
package require tcom

set handle [tcom::ref createobject
  ComTestAtk.ATKComm]

$handle writeEventLog "Tcl is running"
```

TclCsharp

- Tcl interpreter in C# on http://sourceforge.net/projects/tclcsharp
- In Alpha-Phase
- Not more developed
- similar project in J# (Jacl.NET)
- based on Jacl (automatically migrated to C# by MS migration tool)

TclCSharp 2

- Pass Tcl tests from Jacl
- Good .NET integration
- Managed code
- Based on Tcl8.0 code
- No Tk
- No TCP/IP Sockets

TclCsharp Examples

```
package require java
java::load -gac System.Xml.dll
set doc [java::new System.Xml.XmlDocument]
set node [$doc CreateElement tclsharp]
$doc AppendChild $node
$node AppendChild [$doc CreateTextNode "is usable
  for many things"]
puts [[$doc get_DocumentElement] get_OuterXml]
```

Performance - Tclbench

	Tcl(ms)	Java(ms	Java-scale	C#(ms)	C#-scaled	C#/Java
list	92154	791301	8,58	582050	6,31	0,73
catch	46	875	19,02	1732	37,65	1,97
condit	99	13795	139,34	14004	141,45	1,01
data	3682	290110	78,79	142061	38,58	0,48
eval	4056	12353	3,04	8493	2,09	0,68
expr	132	9233	69,94	8788	66,57	0,95
heapsort	9696	975654	100,62	324873	33,5	0,33
loops	4168	740282	177,61	383107	91,91	0,51
proc	600	14593	24,32	19775	32,95	1,35
trace	35	517	14,77	624	17,82	1,2
unset	36	860	23,88	793	22,02	0,92

Performance Results

- Tcl in .NET und Java is about 10 times slower as in C
- Tclbench – no real world programs but make por compare different versions of Tcl C Interp
- .NET in most benches better than Java (but no .NET specific improvements)
- For improvements static compilation needed (generation of MSIL like IronPython)

That's all

Questions?

Artur Trzewik
http://www.xdobry.de