Vorlesung "Application Frameworks and Componentware"

Peter Sturm
Universität Trier

.NET

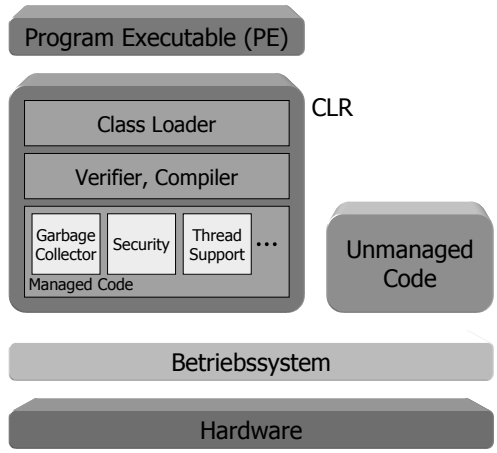
- Völlig neue Systemstruktur als Antwort auf Java
  - Assembly = Komponente
  - Zwischensprache (IL = Intermediate Language)
  - Übersetzung einer Methoden vor erstem Aufruf in nativen und für Zielmaschine optimierten Code (JIT)
  - Einfach Verteilbar

Keine virtuelle Maschine

- Weitgehender Wegfall der Windows Registry
  - Assembly enthält alle notwendigen Metadaten
  - Systemweiter Cache benutzter Assemblies
  - Unterschiedliche Versionen einer Assembly auf einem System möglich

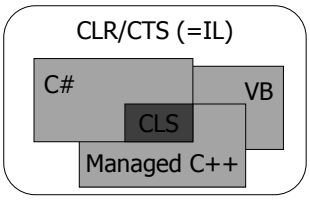
Architektur

- Managed Code
  - Wird von virtueller Maschine ausgeführt
  - Setzt auf einem gemeinsamen Typsystem auf (CTS)
- Unmanaged Code
  - Win32-Code o.ä.
- .NET Framework
  - Laufzeitumgebung
  - Tools
  - Klassenbibliothek



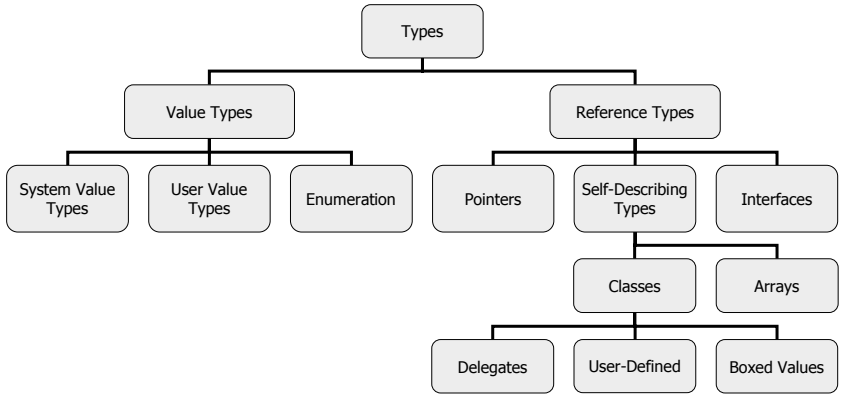
Common Language Runtime (CLR)

- Tools
  - Laden und Verwalten von Assemblies
  - Compiler
  - Diverse Tools
  - Caching von Assemblies (GAC)
- Standard-Assemblies
- Sprachübergreifendes Typsystem (CTS)
  - Vorhandene Typen in allen .NET-Sprachen nutzbar
  - C#-Ableitung von VB-Basisklasse



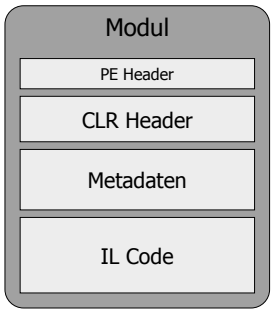
Verfügbare Datentypen

Common Type System (CTS)



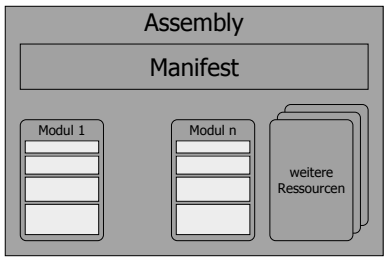
Managed Modul

- Übersetzungseinheit in .NET
- Bestandteile
  - PE Header
    - Dateityp (GUI, CUI, DLL, …)
    - Zeitstempel
    - Primär für unmanaged Code
  - CLR Header
    - Benötigte CLR Version
    - Ort der Metadaten etc.
    - Einstiegspunkt
    - …
  - Metadaten
    - Definierte Typen und Member
    - Referenzierte Typen und Member
  - IL
    - Der übersetzte Code



Assembly

- Zusammenfassung von
  - 1 oder mehreren managed Modulen
  - Weitere Ressourcen (Graphiken, HTML-Dateien, etc.)
- Struktur wird durch ein Manifest beschrieben
  - Name
  - Shared Name
  - Version
  - Hash
  - Referenzierte Assemblies
  - …
- Assembly Linker (AL.exe)



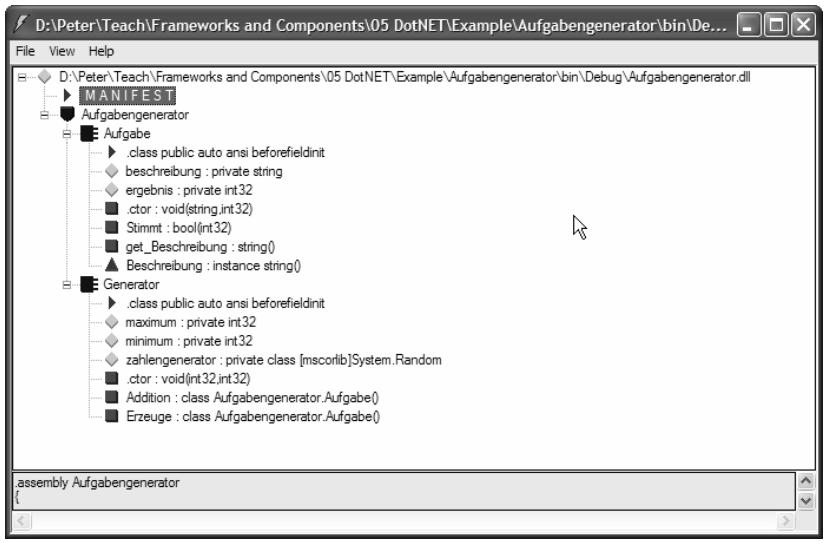
Beispiel

- Generatorklasse
  - Erzeugt Rechenaufgaben
  - Frontend gibt Aufgaben aus und prüft Ergebnis
- Zwei Klassen
  - Aufgabe
  - Generator

```
public class Aufgabe
{
    internal Aufgabe ( string a_beschreibung, int a_ergebnis )
    {
        beschreibung = a_beschreibung;
        ergebnis = a_ergebnis;
    }

    private string beschreibung;
    private int ergebnis;

    public string Beschreibung
    {
        get
        {
            return beschreibung;
        }
    }

    public bool Stimmt ( int wert )
    {
        return ergebnis == wert;
    }
}
```

```
public class Generator
{
    public Generator ( int a_minimum, int a_maximum )
    {
        minimum = a_minimum;
        maximum = a_maximum;
        zahlengenerator = new Random();
    }

    public Aufgabe Erzeuge ()
    {
        return Addition ();
    }

    private Aufgabe Addition ()
    {
        int ergebnis = zahlengenerator.Next(minimum,maximum+1);
        int x = zahlengenerator.Next(minimum,ergebnis+1);
        int y = ergebnis - x;
        string b = x.ToString() + " + " + y.ToString();
        return new Aufgabe(b,ergebnis);
    }

    private int minimum;
    private int maximum;
    private Random zahlengenerator;
}
```

Struktur der Assembly



Manifest

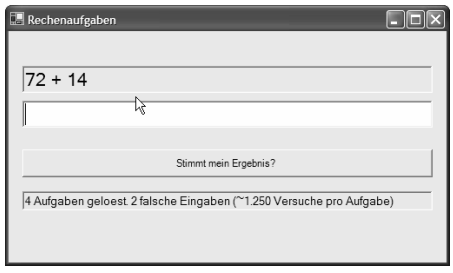
Generator::Addition : class Aufgabengenerator.Aufgabe()

```
.method private hidebysig instance class Aufgabengenerator.Aufgabe
        Addition() cil managed
{
  // Code size       90 (0x5a)
  .maxstack  4
  .locals init ([0] int32 ergebnis,
           [1] int32 x,
           [2] int32 y,
           [3] string b,
           [4] class Aufgabengenerator.Aufgabe CS$00000003$00000000)
  IL_0000:  ldarg.0
  IL_0001:  ldfld      class [mscorlib]System.Random Aufgabengenerator.Generator::zahlengenerator
  IL_0006:  ldarg.0
  IL_0007:  ldfld      int32 Aufgabengenerator.Generator::minimum
  IL_000c:  ldarg.0
  IL_000d:  ldfld      int32 Aufgabengenerator.Generator::maximum
  IL_0012:  ldc.i4.1
  IL_0013:  add
  IL_0014:  callvirt   instance int32 [mscorlib]System.Random::Next(int32,
                                                                    int32)
  IL_0019:  stloc.0
  IL_001a:  ldarg.0
  IL_001b:  ldfld      class [mscorlib]System.Random Aufgabengenerator.Generator::zahlengenerator
  IL_0020:  ldarg.0
  IL_0021:  ldfld      int32 Aufgabengenerator.Generator::minimum
  IL_0026:  ldloc.0
  IL_0027:  ldc.i4.1
  IL_0028:  add
  IL_0029:  callvirt   instance int32 [mscorlib]System.Random::Next(int32,
                                                                    int32)
  IL_002e:  stloc.1
  IL_002f:  ldloc.0
  IL_0030:  ldloc.1
  IL_0031:  sub
  IL_0032:  stloc.2
  IL_0033:  ldloca.s   x
  IL_0035:  call       instance string [mscorlib]System.Int32::ToString()
  IL_003a:  ldstr      " + "
  IL_003f:  ldloca.s   y
  IL_0041:  call       instance string [mscorlib]System.Int32::ToString()
  IL_0046:  call       string [mscorlib]System.String::Concat(string,
                                                              string,
                                                              string)
  IL_004b:  stloc.3
  IL_004c:  ldloc.3
  IL_004d:  ldloc.0
  IL_004e:  newobj     instance void Aufgabengenerator.Aufgabe::.ctor(string,
                                                                      int32)
  IL_0053:  stloc.s    CS$00000003$00000000
  IL_0055:  br.s       IL_0057
  IL_0057:  ldloc.s    CS$00000003$00000000
  IL_0059:  ret
} // end of method Generator::Addition
```

Beispiel: IL Code

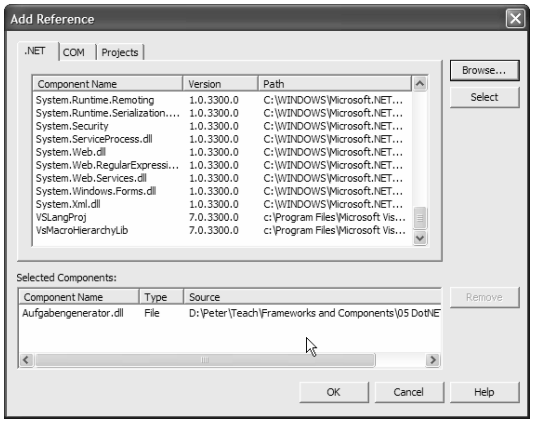
Frontend

- Windows Forms
  - Einfacher Windows-Client
- GUI Builder
  - Textfeld: Aufgabe
  - Textfeld: Ergebnis
  - Textfeld: Status
  - Button: Stimmt
- Reaktion implementieren
  - Doppelklick auf Button im GUI Builder



Zugriff auf Aufgabengenerator

- Project → Add Reference
- Funktion
  - Einlesen der Metadaten aus referenzierter Assembly
  - Integration der Typ-information in IDE
  - Verweise auf referenzierte Assembly



Initialisierung des GUI

```
public class Rechenaufgabe : System.Windows.Forms.Form
{
    private System.Windows.Forms.TextBox Aufgabe;
    private System.Windows.Forms.Button Stimmt;
    private System.Windows.Forms.TextBox Ergebnis;
    private System.Windows.Forms.TextBox Status;
    /// <summary>
    /// Required designer variable.
    /// </summary>
    private System.ComponentModel.Container components = null;

    private Generator agen;
    private Aufgabe a;

    public Rechenaufgabe ()
    {
        // Required for Windows Form Designer support
        InitializeComponent();

        agen = new Generator(0,100);
        a = agen.Erzeuge();
        Aufgabe.Text = a.Beschreibung;
        Ergebnis.Focus();

    }
```

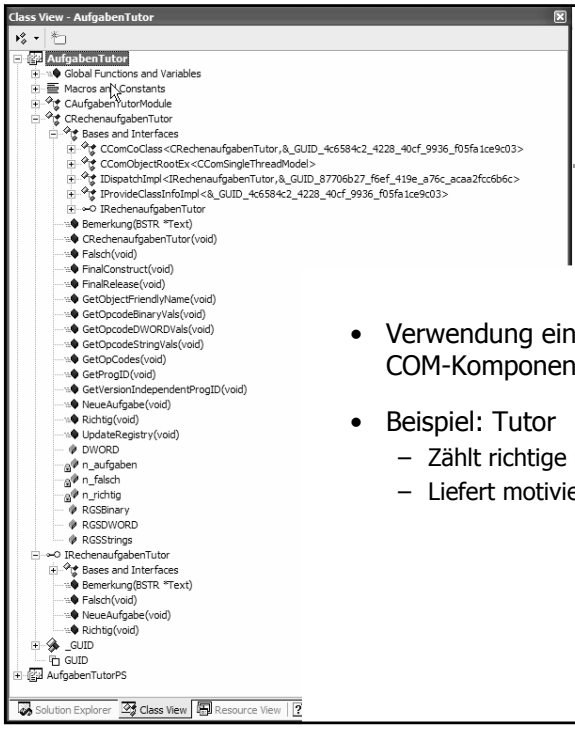
```
private void Stimmt_Click(object sender, System.EventArgs e)
{
    int erg;
    string text;

    if (Ergebnis.Text.Length == 0)
        erg = 0;
    else
        erg = Convert.ToInt32(Ergebnis.Text);

    if (a.Stimmt(erg))
    {
        Status.Text = "Richtig :-)";
        a = agen.
        Aufgabe.T                    eibung;
    }
    else
    {
        Status.Text = "Falsch :-(";
    }
    Ergebnis.Text = "";
    Ergebnis.Focus();
}
```

Equals
Erzeuge
GetHashCode
GetType
ToString

MANIFEST

```
.assembly extern System.Windows.Forms
{
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )                         // .z\V.4..
  .ver 1:0:3300:0
}
.assembly extern System
{
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )                         // .z\V.4..
  .ver 1:0:3300:0
}
.assembly extern Aufgabengenerator
{
  .ver 1:0:1075:30027
}
.assembly extern mscorlib
{
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )                         // .z\V.4..
  .ver 1:0:3300:0
}
.assembly extern System.Drawing
{
  .publickeytoken = (B0 3F 5F 7F 11 D5 0A 3A )                         // .?_....:
  .ver 1:0:3300:0
}
.assembly AufgabenFrontend
{
  .custom instance void [mscorlib]System.Reflection.AssemblyCopyrightAttribute::.ctor(string) = ( 01 00 00 00 00 )
  .custom instance void [mscorlib]System.Reflection.AssemblyKeyNameAttribute::.ctor(string) = ( 01 00 00 00 00 )
  .custom instance void [mscorlib]System.Reflection.AssemblyKeyFileAttribute::.ctor(string) = ( 01 00 00 00 00 )
  .custom instance void [mscorlib]System.Reflection.AssemblyDelaySignAttribute::.ctor(bool) = ( 01 00 00 00 00 )
  .custom instance void [mscorlib]System.Reflection.AssemblyTrademarkAttribute::.ctor(string) = ( 01 00 00 00 00 )
  .custom instance void [mscorlib]System.Reflection.AssemblyConfigurationAttribute::.ctor(string) = ( 01 00 00 00 00 )
  // --- The following custom attribute is added automatically, do not uncomment -------
  //  .custom instance void [mscorlib]System.Diagnostics.DebuggableAttribute::.ctor(bool,
  //                                                                                 bool) = ( 01 00 01 01 00 00 )
  .custom instance void [mscorlib]System.Reflection.AssemblyCompanyAttribute::.ctor(string) = ( 01 00 00 00 00 )
  .custom instance void [mscorlib]System.Reflection.AssemblyProductAttribute::.ctor(string) = ( 01 00 00 00 00 )
  .custom instance void [mscorlib]System.Reflection.AssemblyDescriptionAttribute::.ctor(string) = ( 01 00 00 00 00 )
  .custom instance void [mscorlib]System.Reflection.AssemblyTitleAttribute::.ctor(string) = ( 01 00 00 00 00 )
  .hash algorithm 0x00008004
  .ver 1:0:1075:37446
}
.mresource public AufgabenFrontend.Rechenaufgabe.resources
{
}
.module AufgabenFrontend.exe
// MVID: {7F94EF8C-B1AC-446E-8F9F-5ADA0BE01971}
.imagebase 0x00400000
.subsystem 0x00000002
.file alignment 512
.corflags 0x00000001
// Image base: 0x031b0000
```

Manifest Frontend

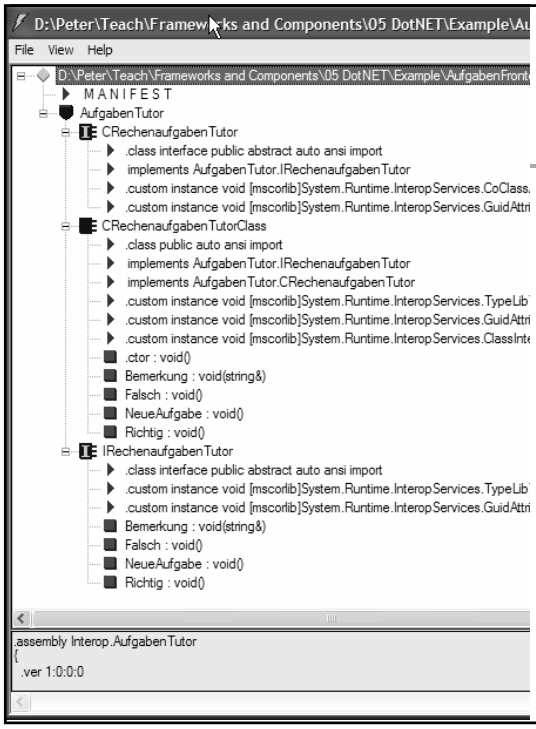


COM-Integration

- Verwendung einer existierenden COM-Komponente
- Beispiel: Tutor
  - Zählt richtige und falsche Antworten
  - Liefert motivierenden Text

Tutor-Code

```
STDMETHODIMP CRechenaufgabenTutor::NeueAufgabe(void)
{
    n_aufgaben++;
    return S_OK;
}

STDMETHODIMP CRechenaufgabenTutor::Richtig(void)
{
    n_richtig++;
    return S_OK;
}

STDMETHODIMP CRechenaufgabenTutor::Falsch(void)
{
    n_falsch++;
    return S_OK;
}

STDMETHODIMP CRechenaufgabenTutor::Bemerkung(BSTR* Text)
{
    char buffer[128]; // Ich weiß, so soll man das nicht machen :-)
    double mittelwert = ((double) n_richtig+n_falsch) / n_aufgaben;
    sprintf(buffer,"%d Aufgaben geloest. %d falsche Eingaben (~%.3lf Ve
    CComBSTR t = buffer;
    t.CopyTo(Text);
    return S_OK;
}
```



Referenz auf COM-Objekt

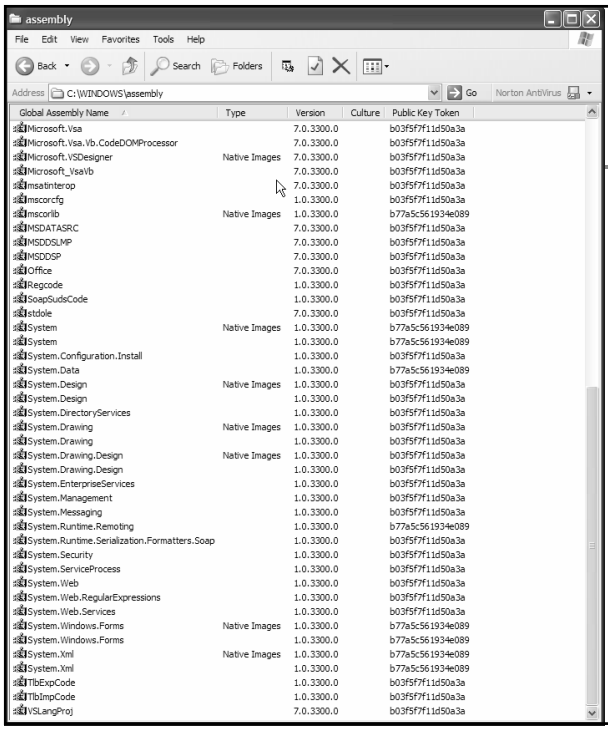
- Im GUI-Projekt
  - Project → Reference
    - Typbibliothek (tlb) der gewünschten Komponente oder
    - Bibliothek selbst (dll)
- Automatische Generierung eines Wrappers

Tutor erzeugen

```
private System.Windows.Forms.TextBox Aufgabe;
private System.Windows.Forms.Button Stimmt;
private System.Windows.Forms.TextBox Ergebnis;
private System.Windows.Forms.TextBox Status;
/// <summary>
/// Required designer variable.
/// </summary>
private System.ComponentModel.Container components = null;

private Generator agen;
private Aufgabe a;
private AufgabenTutor.CRechenaufgabenTutorClass tutor;

public Rechenaufgabe ()
{
    // Required for Windows Form Designer support
    InitializeComponent();

    agen = new Generator(0,100);
    a = agen.Erzeuge();
    Aufgabe.Text = a.Beschreibung;
    Ergebnis.Focus();

    // Creating a new tutor
    tutor = new AufgabenTutor.CRechenaufgabenTutorClass();
    tutor.NeueAufgabe();
}
```

Tutor verwenden

```
private void Stimmt_Click(object sender, System.EventArgs e)
{
    int erg;
    string text;

    if (Ergebnis.Text.Length == 0)
        erg = 0;
    else
        erg = Convert.ToInt32(Ergebnis.Text);

    if (a.Stimmt(erg))
    {
        tutor.Richtig();
        tutor.Bemerkung(out text);
        Status.Text = text;
        a = agen.Erzeuge();
        Aufgabe.Text = a.Beschreibung;
        tutor.NeueAufgabe();
    }
    else
    {
        tutor.Falsch();
        tutor.Bemerkung(out text);
        Status.Text = text;
    }
    Ergebnis.Text = "";
    Ergebnis.Focus();
}
```



Assembly Cache

- Über Explorer
  - siehe Bild
- Tool
  - gacutil

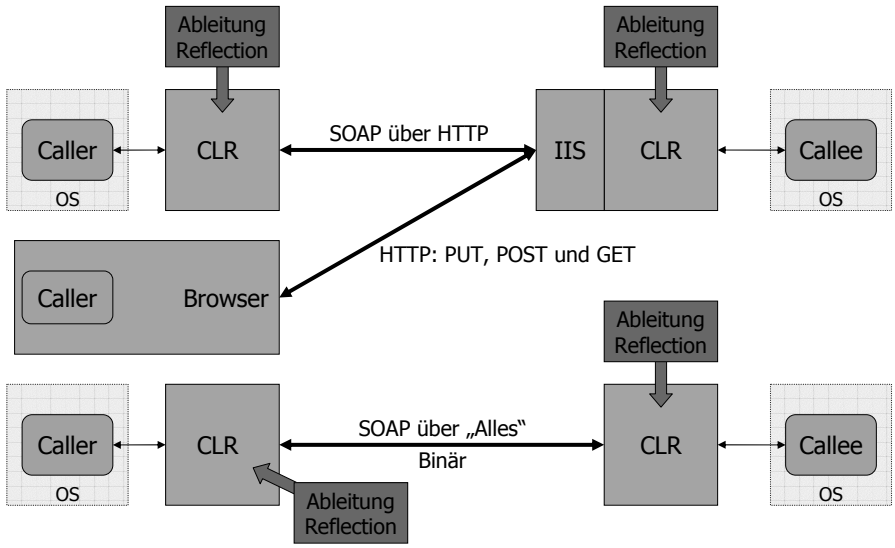
.NET FCL

- .NET Framework Class Library
  - Teile sind ECMA-Standard
- Größenordnung
  - mehr als 7000 Typen
  - ca. 100 Namespaces
- Alle gängige Funktionalität vorhanden
  - … zumindest bisher nichts vermißt
- Windows Forms
  - „MFC.NET"

Interaktionsvarianten in .NET

- Web Services
  - HTTP als Transportprotokoll
  - XML-basierter RPC SOAP
  - .NET-Komponenten hinter Web-Server (IIS)
- Web Forms
  - „Reiche" WWW Forms
  - Klassisches HTTP-Protokoll (PUT, POST und GET)
- Remoting
- Windows Forms
  - Traditionelle Windows-Programme
  - Nachfolger der MFC

Remoting, Web Form und Web Service



.NET und Open Source

- Microsoft läßt Teile von .NET standardisieren
  - C# und CLR sind ECMA-Standard
  - C# und CLR sollen ISO-Standard werden
- Außen vor bleiben wesentliche Teile der .NET FCL
- Bekannte "Open Source"-Ansätze
  - Mono: http://www.go-mono.com/
  - DotGNU: http://www.dotgnu.org/
  - Rotor: http://msdn.microsoft.com/msdnmag/issues/02/07/sharedsourcecli/default.aspx

- Initiator Miguel de Icaza, Ximian
- "Open Source"-Implementierung
  - C#-Compiler
  - CLR
  - Class Libraries
- Class Libraries
  - ECMA-definierten Klassen sind "kein" Problem
  - ASP.NET: Web Forms und Web Services, Controls
  - ADO.NET: Providers für verschiedene DBs fertig
  - Windows Forms: Win32-Implementierung gegen WinLib

DotGNU

- Meta Projekt in der GNU Tradition
- DotGNU Portable.NET (ECMA-basiert)
  - C# Compiler
  - CLR
  - Class Libraries
- Mehr Weltanschauung als bei Mono
- Aktuell geringer Reifegrad
  - Arbeiten an C#-Compiler und CLR
  - Ziele insgesamt etwas diffus

Rotor

- Rotor = Common Language Infrastructure (CLI)
- Microsoft-kontrollierte Öffnung zu Forschungszwecken
  - Shared Source License
  - http://msdn.microsoft.com/msdnmag/issues/02/07/sharedsourcecli/default.aspx
- Inhalt:
  - Implementierung der CLI-Laufzeitplattform (ECMA-335) für Windows XP und FreeBSD
  - Compiler für C# (ECMA-334) und Jscript
  - Werkzeuge: assembler/disassemblers (ilasm, ildasm), debugger (cordbg), metadata introspection (metainfo), …
  - Platform Adaptation Layer (PAL) für die Portierung der CLI von Windows XP nach FreeBSD
  - Entwicklungswerkzeuge: nmake, build, …
  - Dokumentation und Testsuiten