Präsentation .NET
Teleseminar Web Services
Universität Karlsruhe / Universität Mannheim

Dominik Schmider, Dirk Mazur

Gliederung

- Einführung
- NET-Programmiersprachen
- Das .NET-Framework
- Bausteine des .NET-Framework
- Metadaten und Reflection
- Assemblies
- .NET und Webdienste

Umfrage!

- Wie weit ist .NET bekannt?

Was ist .NET?

- Entwicklungsplattform für Eigenentwicklung
- Ziel der .NET-Technologie:

  Programmierung von normalen, verteilten und Web-Anwendungen
- Neuheit: sprachunabhängige und plattformunabhängige Nutzung

Hintergrund

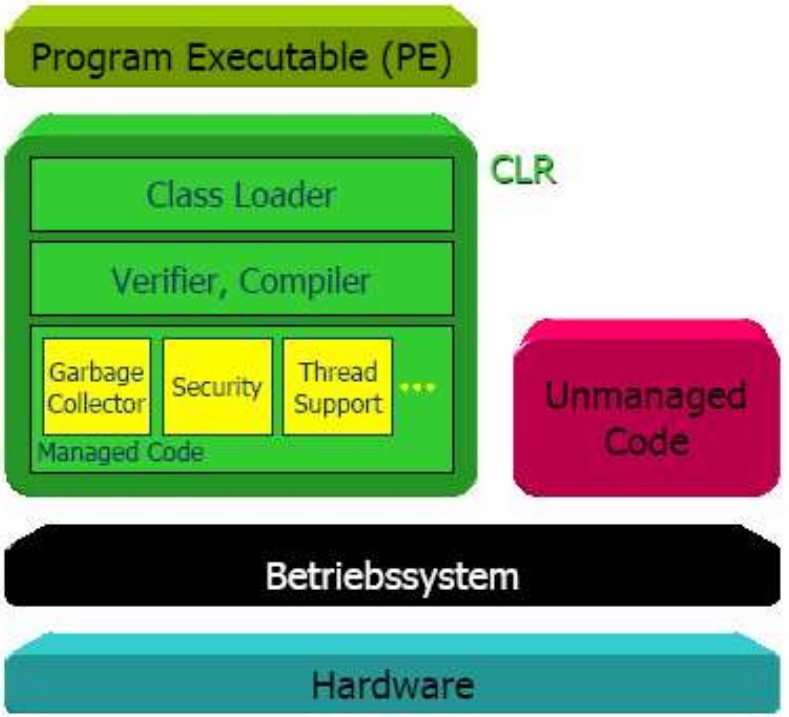
- .NET stammt von dem MS Entwicklungsteam
- Markteinführung durch Microsoft im Jahre 2002
- OO-Sprachen haben einen gemeinsamen Nenner
- Die Idee hinter .NET

.NET-Programmiersprachen

- Von Microsoft:
  - C++, C#, JScript, Visual Basic.NET, J#
- Von Drittanbietern (>20)
  - Smalltalk, COBOL, Delphi, Pascal, Eiffel, Perl, Python, usw.

.NET-Architektur

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



Common Language Runtime CLR

- Die CLR ist die Weiterentwicklung der bisherigen Konzepte zu einem einheitlichen Integrationsmodell
- Die CLR bietet ein:
  - Einheitliches und sprachübergreifendes Typsystem
  - Einheitliches und erweiterbares Format für Metadaten
  - Einheitliche und sprachübergreifende Laufzeitumgebung
  - Einheitliche und sprachübergreifende Klassenbibliothek

Common Intermediate Language CIL

- CIL = MSIL= Microsoft Intermediate Language
- Compiler erzeugen keinen Maschinencode sondern eine prozessorunabhängige Zwischensprache
- Übersetzung zur Laufzeit in Maschinencode
- Sprachintegration erfolgt auf Codeebene
- Sämtlicher Code wird unter Aufsicht der CLR ausgeführt (wird als Managed Code bezeichnet)
  - CLR führt Sicherheitsüberprüfungen aus
  - CLR übernimmt Speicherverwaltung und Fehlerbehandlung (Garbage Collector, Exceptions)
  - CLR führt Versionsüberprüfung aus

Just-In-Time Compiler

- IL-Code wird vor der Ausführung immer durch Compiler in echten Maschinencode übersetzt
  - Kompilierung methodenweise bei Bedarf (Häppchenweise)

    kein Performanceverlust (da mehrere MB pro Sekunde)
  - Gleichwertigkeit der Sprachen, da alle Compiler IL-Code erzeugen
  - Unabhängigkeit von Hardwareplattformen

Common Type System CTS

- Das Typsystem wandert vom Compiler in die CLR
- Typen werden eindeutig („Spielregeln“)
- Sprachen werden interoperabel,

  da das gleiche Typsystem (CTS/CLS) benutzt wird
- Einheitliche Fehlerbehandlung
- Sämtliche Klassen werden von System.Object abgeleitet
- Großer Satz an Datentypen,

  Unterscheidung in „Value Types“ und „Reference Types“

Value Type vs. Reference Type

- Value Types
  - „liegen“ auf dem Stack
  - Enthalten Daten (wie primitive Datentypen, Aufzählungstypen und benutzerdefinierte Werttypen)
  - Per Ableitung eines Managed Type können eigene Werttypen definiert werden
  - (Können nicht null sein)
- Reference Types
  - „liegen“ auf dem Heap
  - Enthalten Referenzen auf Objekte (wie Klassen, Felder, Zeiger)
  - (Können null sein)

C#

- C# ist eine objekt- und komponentenorientierte Programmiersprache
- C# ist an C++ und Java angelehnt
- C# setzt auf dem Common Type System auf und

  bringt kein eigenes Typsystem mit
- C# Typen sind schon Managed Types
- Wird als bevorzugte .NET-Sprache von Microsoft positioniert

Base Class Library BCL

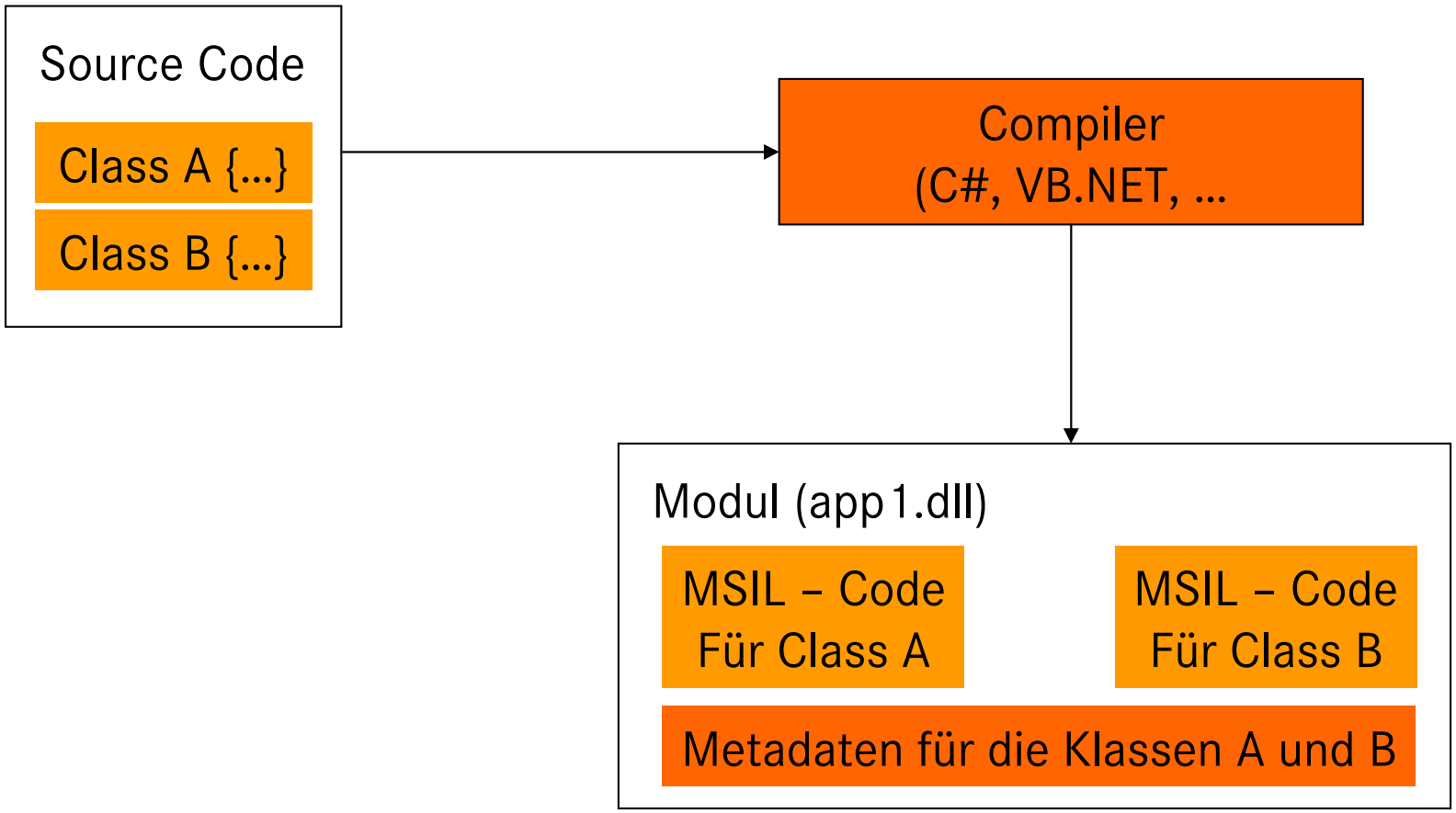
- Standardbibliothek (Teile sind ECMA-Standard)
- Alle Einträge liegen als Managed Code (MSIL) vor
- Sprachunabhängig
- Aufgeteilt durch Namespaces
- Größenordnung:
  - >7000 Typen
  - Ca. 100 Namespaces
- Alle gängigen Funktionalitäten vorhanden, wird immer wieder erweitert, Möglichkeit zur Selbsterweiterung

Metadaten und Reflection

- Ein .NET Programm besteht aus:
  - Metadaten
  - MSIL – Code
- Metadaten sind immer mit dem dazugehörigen Code verbunden
- Es gibt 3 Arten von Metadaten
  - Definitionstabellen
  - Referenztabellen
  - Manifesttabellen

Metadaten und Reflection(2)

- Übersetzen von Source Code / Metadaten



Metadaten und Reflection(3)

- Metadaten sind für alle Module auf die gleiche Art und Weise aufgebaut
  - Einheitliches Format
- Metadaten eines Moduls können zur Laufzeit ausgelesen werden und erweitert werden
  - Diesen Vorgang nennt man Reflection
  - Das .NET Framework stellt entsprechende Klassen über den Namespace System.Reflection bereit

Metadaten und Relfection Zusammenfassung

- Typen beschreiben sich selbst
- Entsprechende Metadaten werden beim Compilen erzeugt
- Beschreibungsinformation ist einheitlich über alle Sprachen

Assemblies und Module

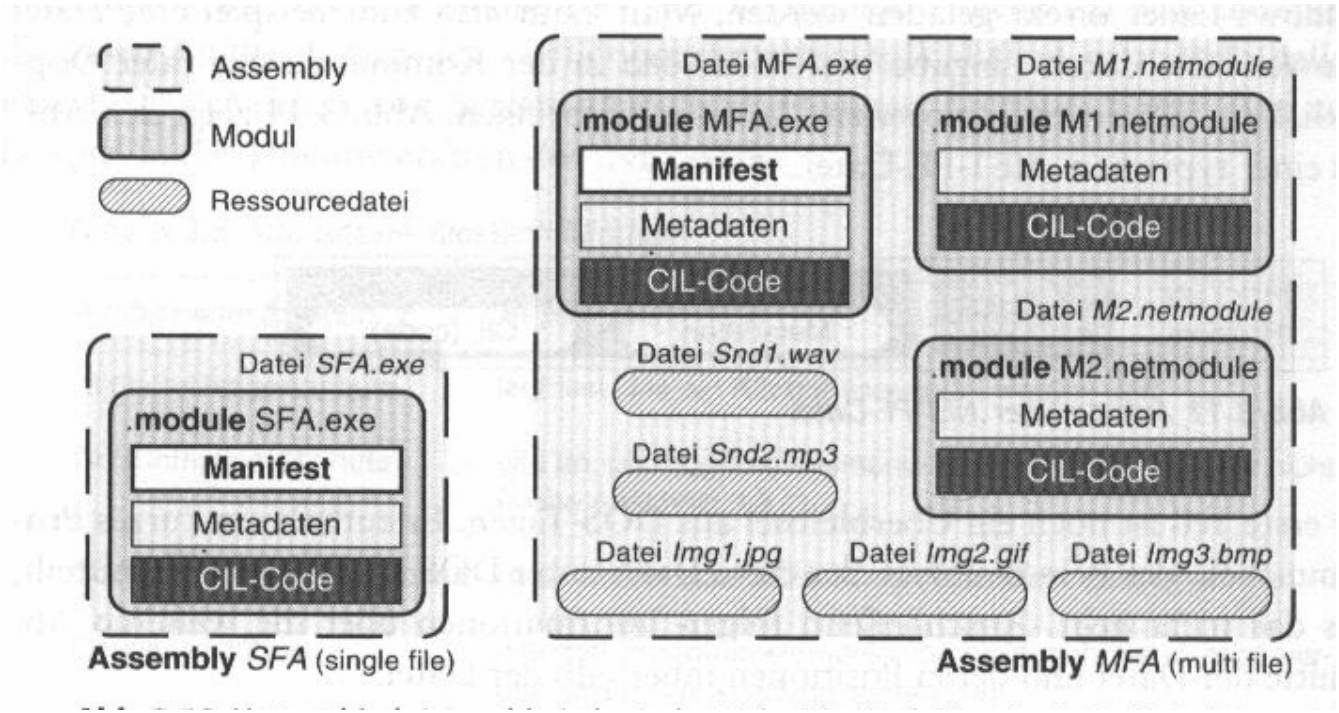
- Ein Modul dient als Container für Typen
- Ein Modul enthält
  - MSIL-Code der Typen
  - Beschreibungen der Typen (Metadaten)
- Jedes Modul enthält Metadaten
  - Compiler erstellt Metadaten „on the fly“
- Physische Einheit (dll/exe –Datei)

Assemblies und Module

- .NET Anwendungen bestehen aus Assemblies
- Eigenschaften von Assemblies:
  - Container für Module
  - Eine logische Einheit
- Aufgaben:
  - Auslieferung
  - Kapselung (Sichtbarkeit und Zugriffsrechte)
  - Versionierung

Assemblies und Module

- Grafische Abgrenzung der beiden:



Assemblies und Module

- Manifest (Metadaten für Assemblies) enthält:
  - Assembly–Identität = Name + Version + Ländercode
  - Assembly Tabelle (nur ein Modul)
  - File Tabelle (Dateiname / Attribute / Hashwert)
  - Exported-Type Tabelle

Assemblies und Module

- Kategorien von Assemblies:
  - Private Assembly
    - Innerhalb eines Anwendungsverzeichnisses bekannt
    - Identifikation anhand eines einfachen Namens, z.B. „App1“
    - Keine Versionsprüfung /versch. Versionen liegen in versch. Verzeichnissen
    - Installation per Filecopy → Zero-Impact-Installation
  - Shared Assembly
    - Assembly kann global von allen Anwendungen genutzt werden
      Stichwort: GAC

Assemblies und Module

- Shared Assembly – Strong Name
  - Identifikation über einen Strong Name
    - Eindeutigkeit per Public-Key-Verschlüsselung
    - Strongname = Identität + Public Key Token
  - Versionsprüfung durch die CLR
  - Installation im Global Assembly Cache (GAC)

Assemblies und Module

- Side-by-Side Execution:
  Unterschiedliche Versionen gleicher Komponenten können parallel betrieben werden (→ Ende der „DLL Hölle“) durch:
  - Verschiedene Versionen haben versch. Strong Names (mind. VersNr)
  - Jede Anwendung gibt an welche Version einer Komponente sie benötigt
    → man kann neuere Versionen installieren ohne alte zu überschreiben
- Sprachübergreifende Vererbung
- Sprachübergreifende Fehlerbehandlung

.NET und Webservices

- Überblick über die Webdienst-Namespaces
- Erstellen eines einfachen Web Service
- Lebenszyklus von Webservices
- SOAP Codierung von .NET Datentypen
- Werkzeuge und Ressourcen

Überblick über den Webdienstnamespace

- System.Web.Service
  - WebMethodAttribute
  - WebService
  - WebServiceAttribute
  - WebServiceBindingAttribute
- System.Web.Service.Description
- System.Web.Service.Discovery
- System.Web.ServiceProtocols

Erstellen eines einfachen Web Services

- Zwei Möglichkeiten
  - Mit Visual Studio .NET
  - Mit einem Texteditor (*.asmx)
- Klasse muss von der Basisklasse Webservice abstammen
- Metaattribut [WebMethod] für Webmethoden
- Datei in ein Verzeichnis auf dem Webserver (IIS) legen und aufrufen

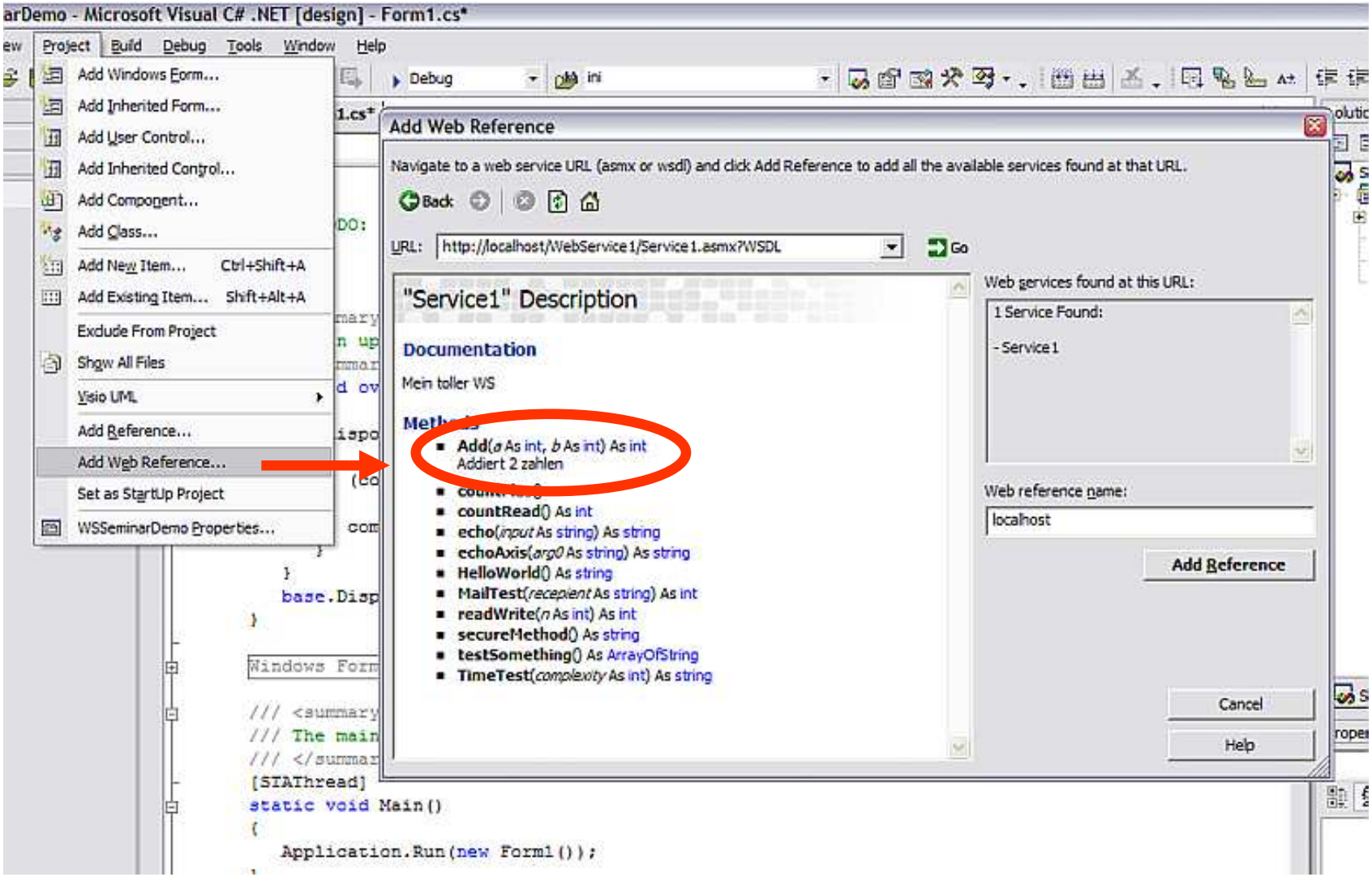
Erstellen eines einfachen Web Services

```
<%@ WebService Language="c#"  Class="Service1" %>

using System;
using System.Web.Services;

[WebService(Description="Mein toller WS")]
public class Service1 : WebService
{
    [WebMethod(Description="Addiert 2 zahlen")]
    public int Add(int a, int b)
    {
        return a + b;
    }
}
```

Client Erzeugung



Web Service-Aufruf

```
//proxy-Klasse deklarieren und instanziieren
localhost.Service1 myService = new localhost.Service1();

private void button1_Click(object sender, System.EventArgs e)
{
        //synchroner WS-Aufruf, Ergebnis speichern
        int result = myService.Add(47, 11);
        textBox1.Text = result.ToString();

        //asynchroner Aufruf
        AsyncCallback cb = new AsyncCallback (this.addIsComplete);
        IAsyncResult res = myService.BeginAdd (8, 15, cb, null);
}

//Methode für asynchronen Rückruf
private void addIsComplete(IAsyncResult res)
{
        int sum = myService.EndAdd (res);
        textBox2.Text = sum.ToString();
}
```

Lebenszyklus von Web Services

```
<%@ Webservice Language="C#" Class="StateDemo" %>

using System.Web.Services;
[WebService(Namespace="http://dotnet.jku.at/StateDemo/")]
public class StateDemo : WebService {
    [WebMethod()]
    public int IncApplication(){
         int hit = (Application["Hit"] == null) ? 0: (int) Application["Hit"];
         hit++;
         Application["Hit"] = hit;
         return hit;
    }

    [WebMethod(EnableSession = true)]
    public int IncSession(){
         int hit = (Session["Hit"] == null)? 0 : (int)Session["Hit"];
         hit++;
         Session["Hit"] = hit;
         return hit;
    }
}
```

SOAP Codierung von .NET Datentypen

- Durch .NET Attribute kann die Art der Codierung und Weiters angepasst werden
  - [SoapRpcMethod] / [SoapRpcService]
  - [SoapAttribute]
  - [SoapElement]
  - [SoapIgnoreAttribute]
  - [SoapIncludeAttribute]

SOAP Codierung von .NET Datentypen

```
Public struct TimeDesc {

[SoapAttribute] public string TimeLong;

[SoapAttribute] public string TimeShort;

[SoapAttribute(DataType=„nonNegativInteger“,
   AttributeName=„ZoneID“)] public string TimeZone;

}

Wird in Soap codiert als:

<types:TimeDesc id=„id1“ xsi:type=„types:TimeDesc“
   types:TimeLong=„10:00:25“ TimeShort=„10:00“
   types:ZoneID=„1“ />
```

SOAP Codierung von .NET Datentypen

```
Public struct TimeDesc {
public string TimeLong;
public string TimeShort;
[SoapElement(DataType=„nonNegativInteger“,
   ElementName=„ZoneID“)] public string TimeZone;
}
Wird in SOAP codiert als
<types:TimeDesc id=„id1“ xsi:type“types:TimeDesc“>
   <TimeLong xsi:type=„xsd:string“> 10:00.25</TimeLong>
   <TimeShort xsi:type=„xsd:string“> 10:00.25</TimeShort>
   <ZoneID xsi:type=„xsd:nonNegativeInteger“>1</ZoneID>
</types:TimeDesc>
```

Werkzeuge und Ressourcen

- wsdl.exe
- ildasm.exe
- .NET Web Service Studio
  - www.gotdotnet.com
  - Testumgebung für WS
  - GUI zur wsdl.exe
  - Einzelne Soap Nachrichten können sichtbar gemacht werden (Response und Request)
- ASP.NET Web Matrix
  - www.asp.net

Ausblick / Zusammenfassung

- Zukunftsweisend
- Zukunft der anwendungsentwicklung unter windows
- www.Go-mono.org

- Sichere Ausführung von Anwendungen durch managed code
- Aufräumen mit alten Problemen (Versionierung und Deployment)
- Detaillierte Rechteverwaltung (was darf der Code was nicht)
- Große Klassenbibliothek
- Webprogrammierung einfacher und schneller
- Durchgängige Unterstützung von Web Services und XML
- Trotz allem schnell

.NET

Geschafft !!!!!