Microsoft .net

Softwareentwicklung mit MS.NET und C#

Was ist .NET? Die .NET Common Language Runtime

Robert Bruckner

06.11.2002

Agenda

- Vision
- .NET Übersicht
- CLR Prinzipien
- CLR Features
- Demos
- Zusammenfassung

Microsoft's Vision Statements

Empower people
through great software,
any time, any place, and on any device.

The .NET Framework is the
programming-model substrate
for the .NET vision.

.NET auf einer Folie

Verteilte Applikationen

- Basierend auf Standards
- Allgegenwärtige Services
- Security und Privacy direkt integriert
- Device-unabhängig
- Programmiersprachen-unabhängig
- Windows-Plattform-unabhängig
- Sehr guter Entwicklersupport (Visual Studio, MSDN)

Was ist .NET?

Mit dem .NET Framework können verteilte, XML basierte Web Applikationen erstellt werden.

Zu einer .NET Plattform gehört ein geeignetes Betriebssystem und Serversoftware.

.NET Plattform

.NET Framework & Tools	*CLR (Common Language Runtime)* Einheitliche Klassenbibliothek, ASP.NET Command Line Tools, Visual Studio.NET
Building Block Services	Ständig verfügbare Internet-Dienste (Code-Updates, Suchdienste, Messenger, .NET My Services)
Infrastruktur	Heutige „2000-Produktfamilie“ (zukünftig .NET Enterprise Servers)
Devices	Clients und mobile Geräte, auf denen .NET Anwendungen laufen (Handy, PDA, Tablet PC)

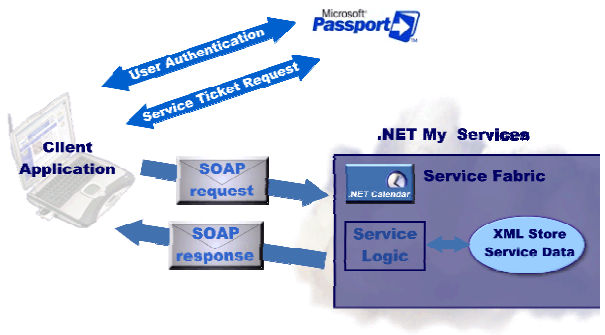
.NET Designziele

- Vereinfachung von Entwicklung und Deployment
- Einheitliches Programmiermodel
- Robuste und sichere Laufzeitumgebung
- Support für „beliebige“ Sprachen

.NET My Services

- Microsoft Passport
  - Authentifizierungsservice mit single sign-on für jede beliebige Web Site
  - Offen für Unternehmensnetzwerke
- .NET My Services
  - Menge von benutzerorientierten Web Services, um persönliche Informationen zu verwalten und zu beschützen.
  - Aufbau von persönlichen Netzwerken
  - Jede Applikation und jedes Device kann die Services verwenden.

.NET My Services



Robert Bruckner I/S TU 9

.NET Evolution

- Ursprünglich:
  - Enge Verbindung zwischen Code und Daten
  - Kaum Integration zwischen Applikationen
- COM:
  - Integration von Komponenten
  - Jede Komponente muss sich um „Plumbing“ selbst kümmern
  - Keine direkte Interaktion
- .NET Common Language Runtime (CLR):
  - Direkte Interaktion zwischen Objekten

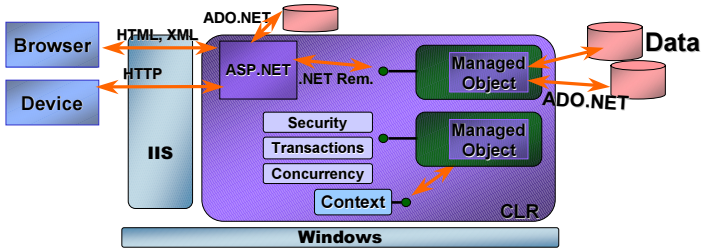
Robert Bruckner I/S TU 10

.NET Framework

- Common Language Runtime (CLR)
  - Common type system für alle Sprachen
  - Mächtige Runtime Umgebung
- Klassenbibliotheken (.NET Framework)
  - Base class libraries, ADO.NET und XML
  - Windows Forms
- Web Applikationsplattform ASP.NET
  - Interaktive Seiten
  - Web Services

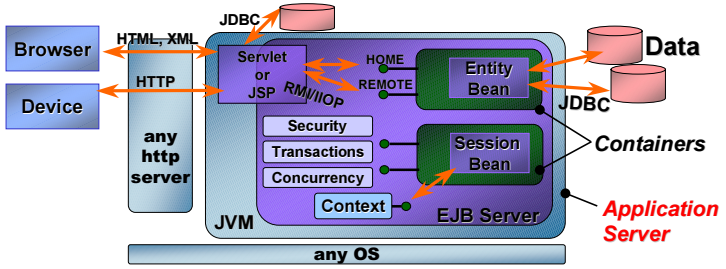
Robert Bruckner I/S TU 11

.NET Framework



Robert Bruckner I/S TU 12

J2EE Plattform

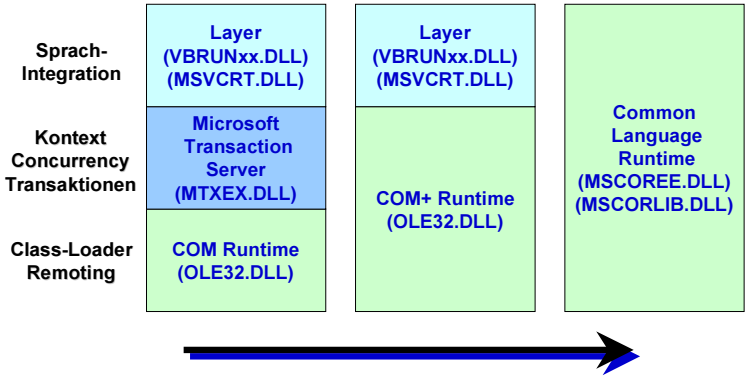


Historie

	1996	1998	2001
Microsoft	Windows DNA - MTS (heute: COM+) - ASP - ADO		.NET Framework - CLR - C#, VB.NET - Nächste Generation von COM+, ASP, ADO - Web Services
Sun/Java	Java - Java (Sprache) - Java VM - J2SE	J2EE - EJB - JSP - JDBC	

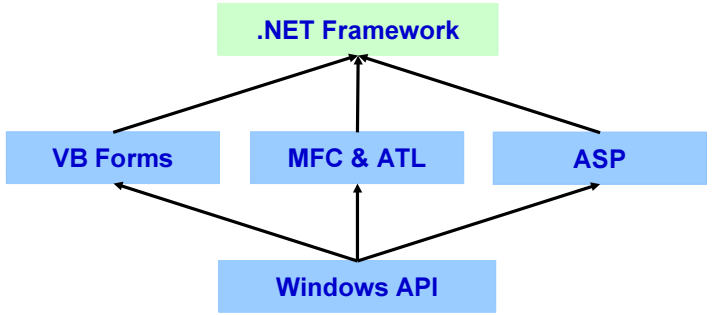
Warum eine Runtime?

Einheitliches Integrationsmodell



Warum ein Framework?

Einheitliches Programmiermodell



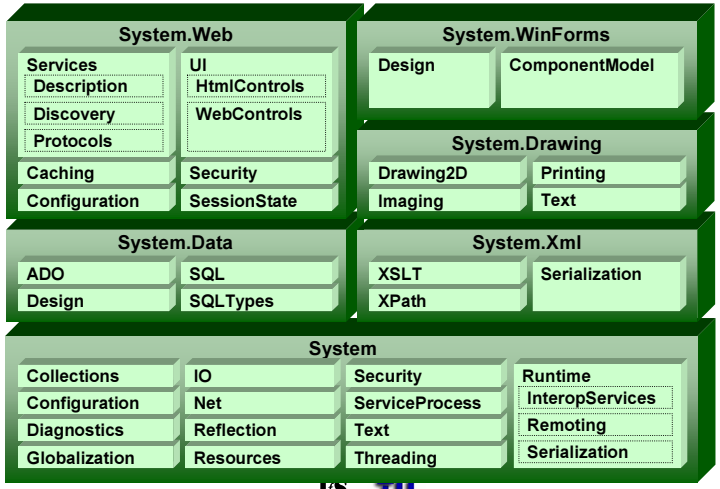
Demo

Beispiel 1: „Hello World“ unter .NET

Demo

Beispiel 2: Vererbung in .NET / CLR

Das .NET Framework



Das .NET Framework

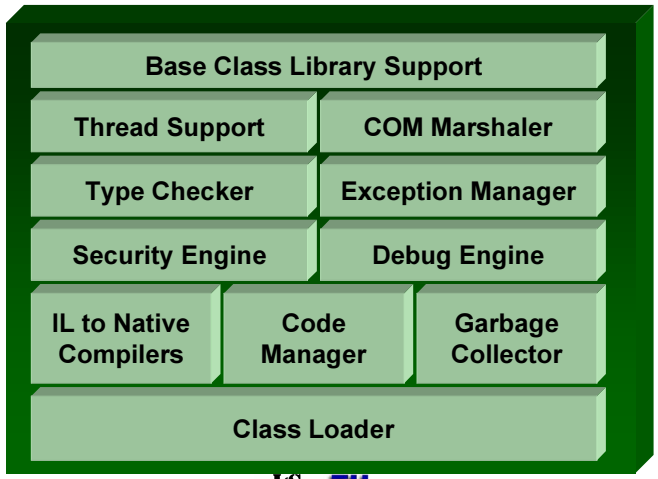
Namespace	Purpose	Assembly DLL
`System`	Core type system	`mscorlib`
`System.Reflection`	Runtime type info	`mscorlib`
`System.Security`	Access control	`mscorlib`
`System.Text`	Text encoding/munging	`mscorlib`
`System.Collections`	Collection types	`mscorlib`
`System.IO`	Binary and text I/O	`mscorlib`
`System.Threading`	Threading/locking	`mscorlib`
`System.Runtime.Serialization`	Object persistence	`mscorlib`
`System.Runtime.Remoting`	SOAP/AOP/Proxies	`mscorlib`
`System.Runtime.InteropServices`	Native code support	`mscorlib`
`System.Data`	Access to DBMS	`System.Data`
`System.Xml`	Access to arbitrary data	`System.Xml`
`System.Web`	Server-side HTTP	`System.Web`
`System.Diagnostics`	Assertion/tracing	`System[.Diagnostics]`

CLR (Common Language Runtime)

- Verwaltet laufende Programme
  - Sicherstellung Typsicherheit
  - Garbage collection, Fehlerbehandlung
  - Securitysystem für Code
- Common Type System (CTS)
  - Value types (integer, float, …)
  - Objekte, Interfaces
  - Delegates, Events, Properties, Pointers
- Zugriff auf System Ressourcen
- CLS (Common Language Specification)
  - Subset der CLR, Standardisierung läuft

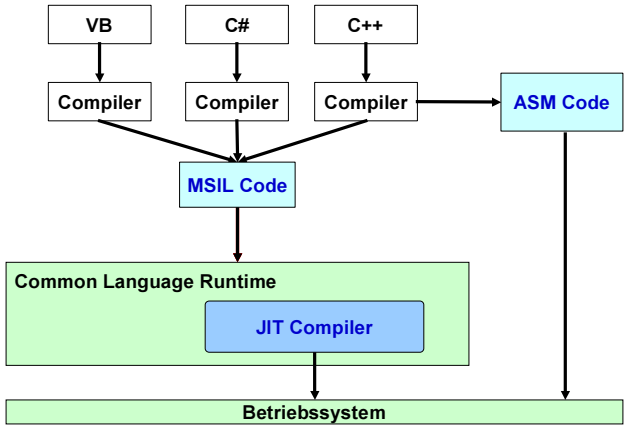
Robert Bruckner I/S TU 21

CLR Architektur



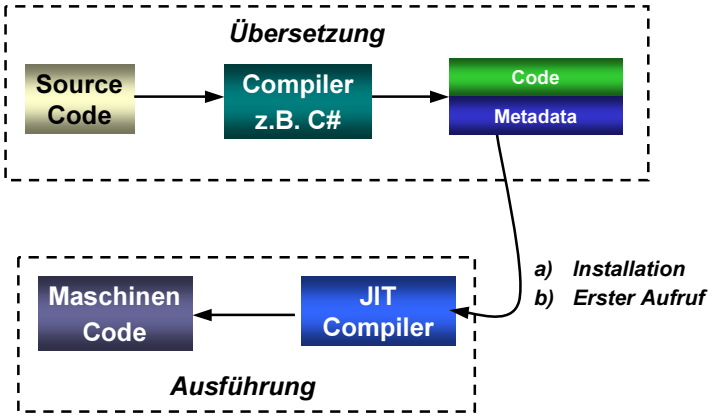
Robert Bruckner I/S TU 22

Übersicht



Robert Bruckner I/S TU 23

Übersetzung und Ausführung



Robert Bruckner I/S TU 24

CLR

Managed Code

- Sämtlicher Code wird unter Aufsicht der Common Language Runtime ausgeführt
  - Runtime führt Sicherheitsüberprüfungen aus
  - Runtime übernimmt Speicherverwaltung und Fehlerbehandlung (→ GarbageCollection, Exceptions)
  - Runtime führt Versionsprüfungen aus
  - Dieser Code wird mit Managed Code bezeichnet

Robert Bruckner I/S TU 25

CLR

Microsoft Intermediate Language (MSIL)

- Compiler erzeugen keinen native Code, sondern eine prozessorunabhängige Zwischensprache
  - Sprachintegration erfolgt auf Codeebene
  - IL-Code wird niemals interpretiert
- C# IL durchschnittlich 5x schneller als Visual Basic 6.0 Maschinencode
- C# IL max. 50% langsamer als C++ Maschinencode (allerdings stark abhängig von COM und Libraries)

Robert Bruckner I/S TU 26

CLR

Code wird kompiliert

- IL-Code wird vor der Ausführung immer (!) durch Compiler in echten Maschinencode übersetzt
  - Unabhängigkeit von Hardwareplattformen
  - unter Windows CE bereits mit einem IL-Vorläufer im Einsatz
- Sicherheit geht weit über Java Sandboxing hinaus.

Robert Bruckner I/S TU 27

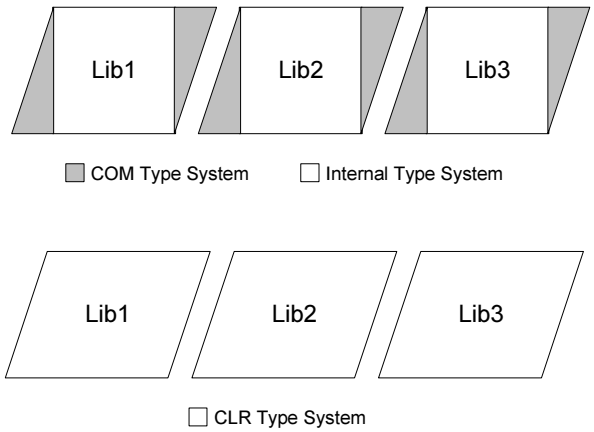
CLR

Common Type System

- Das Typsystem wandert vom Compiler in die Runtime
  - Typen werden eindeutig
    - „ein String unter C# und ein String unter VB.NET sind identisch“
  - Sprachen werden interoperabel, da sie das gleiche Typsystem benutzen
  - CTS – Common Type System

Robert Bruckner I/S TU 28

CLR / CTS vs. COM



CLR

Implikationen

- Sprachen werden gleichwertig, da alle Compiler MSIL-Code erzeugen
  - „eine C# Klasse kann von einer VB.NET Klasse abgeleitet sein“ → Demo 2, Demo 3
  - einheitliche Fehlerbehandlung
- Compilerbau wird einfacher
  - kein Typsystem
  - Sprachen sind per „Definition“ interoperabel

CLR

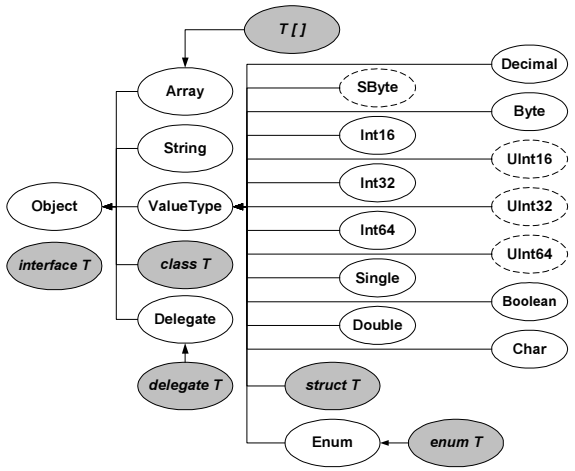
Implikationen

- MSIL unterscheidet sich von „reinen“ Assemblersprachen
  - komplexe Datentypen und Objekte sind fester Bestandteil
  - Konzepte wie Vererbung und Polymorphie werden a priori unterstützt

CLR

Beispiel 3: Integration auf Codeebene

Common Type System



Common Type System

Beispiel 4: Boxing und Unboxing

Common Type System

Gleichheit und Identität von Objekten

- Zwei Objekte sind gleich, wenn deren Inhalte gleich sind
- Zwei Objekte sind identisch, wenn sie die gleiche Instanz referenzieren
- Gleichheit definiert sich über die virtuelle Methode System.Object.Equals
  - identisch: System.Object.Equals = true
  - gleich: System.Object.Equals.Value = true

Common Type System

Beispiel 5: Delegates – Typisierte Funktionszeiger

Common Type System

Attribute

- Klassen und Methoden können über Attribute mit Metadaten versehen werden
- Der Wert eines Attributs kann zur Laufzeit ausgelesen werden
- Attribute werden durch Ableitungen von der Klasse System.Attribute definiert
- Konsequente Weiterentwicklung des Attribut-Gedankens von COM+
  - Aber: COM+ Attribut ≠ CLR Attribut !!!

Common Type System

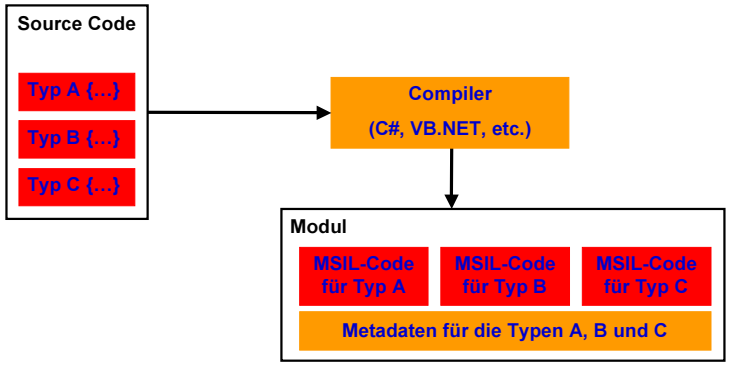
Beispiel 6: Attribute

Bestandsaufnahme

- Die Common Language Runtime ermöglicht unabhängig von Programmiersprachen eine durchgängig objekt- und komponentenorientierte Programmierung
- .NET Sprachen sollten sich auf die Typen beschränken, die über das Common Type System definiert sind.
  Attribute:
  - Assembly: [assembly: CLSCompliant(true)]
  - Im Code: [CLSCompliant(true)]

Metadaten und Reflection

Übersetzen von Sourcen



Metadaten und Reflection

- Ein Modul dient als Container für Typen
- Ein Modul enthält
  - den IL-Code der Typen
  - Beschreibung der Typen
- Die Beschreibung der Typen wird mit Metadaten bezeichnet
- Jedes Modul enthält Metadaten
  - Compiler erstellt Metadaten „on the fly“

Metadaten und Reflection

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

Metadaten und Reflection

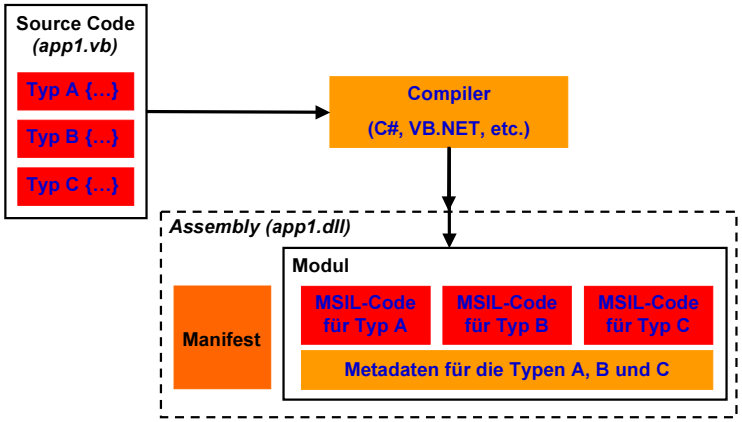
Beispiel 7: Reflection

Assemblies

- .NET Anwendungen bestehen aus Assemblies
- Ein Assembly ist ein Container für Module
- Sämtliche Sicherheits- und Versionsüberprüfungen durch die CLR erfolgen auf der Basis von Assemblies !!!

Assemblies

Übersetzen von Sourcen

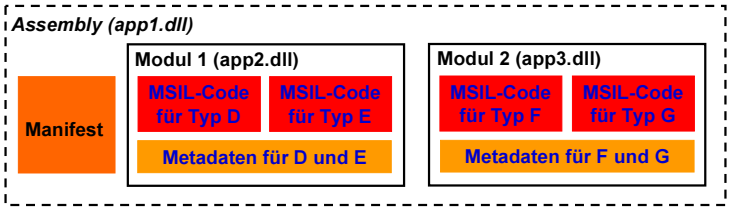


Assemblies

- Sobald ein Modul kompiliert ist, gehört es zu einem Assembly
  - Compiler erstellt Assembly „on the fly“
  - .NET Framework stellt entsprechende Klassen über den Namespace System.Reflection.Emit bereit
- Die im Modul vorhandenen Typen sind nur innerhalb des Assemblies bekannt

Assemblies

Container für mehrere Module



Assemblies

Manifest

- Jedes Assembly enthält genau ein Manifest
- Das Manifest enthält
  - Assembly-Identität
    - Name + Version + Ländercode
  - Liste der Module, aus denen das Assembly besteht
  - Referenzierte Assemblies
  - Exportierte Typen und Resourcen
  - Attribute

Assemblies

Kategorien

- Private Assembly
  - Assembly kann nur von genau einer Anwendung benutzt werden
- Shared Assemby
  - Assembly kann global von allen Anwendungen benutzt werden

Assemblies

Private Assembly

- Identifikation anhand eines einfachen Namens, z.B. “Reverse”
- Keine Versionsüberprüfung
- Installation per Filecopy
  - Standardmäßig befinden sich Assembly und Anwendung im gleichen Verzeichnis
  - Verzeichnis kann per *.config* - Datei definiert werden

Assemblies

Beispiel 8: Private Assembly

Assemblies

Shared Assembly

- Identifikation über einen Strong Name
- Versionsüberprüfung durch die Runtime
- Installation im Global Assembly Cache (→ SDK-Tool al.exe oder gacutil.exe)
  - systemweiter “Speicherbereich”
  - normale Dateien
  - keine Registry-Einträge, o. ä.

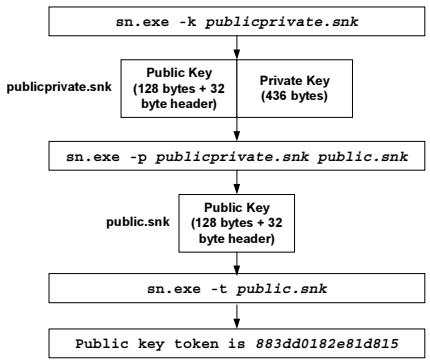
Assemblies

Shared Assembly - Strong Name

- **Eindeutigkeit des Names wird mit Hilfe der Public-Key-Verschlüsselung hergestellt**
  - **Strong Name = Identität + Public Key**
  - **Attribut Originator im Manifest**

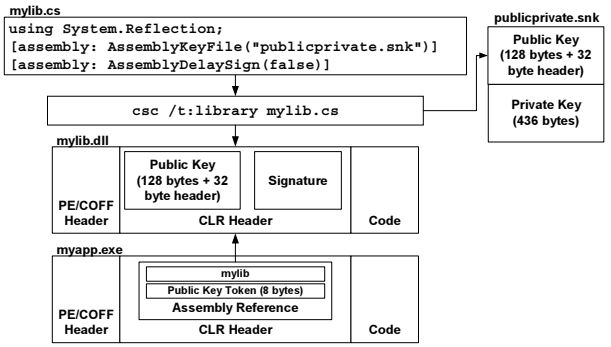
Assemblies

Sign-Verfahren für Shared Assemblies



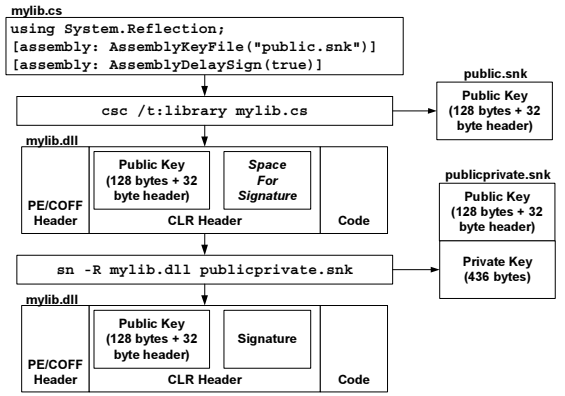
Assemblies

Sign-Verfahren für Shared Assemblies



Assemblies

Delayed Sign-Verfahren für Shared Assemblies



Assemblies

Shared Assembly zur Laufzeit laden

- Client wird standardmäßig an die Version gebunden, die in seinem Manifest eingetragen ist
- Dieses Verhalten kann per *.config*-Datei überschrieben werden

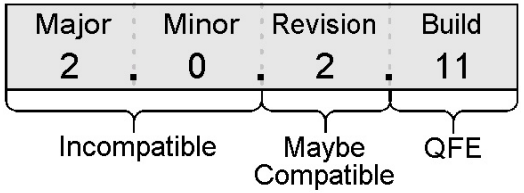
```
<?xml version="1.0" ?>
<configuration
    xmlns:asm="urn:schemas-microsoft-com:asm.v1">
  <runtime>
    <asm:assemblyBinding>
      <asm:probing privatePath="dll"/>
    </asm:assemblyBinding>
  </runtime>
</configuration>
```


Assemblies

Beispiel 9: Shared Assembly

Versionierung

Aufbau der Versionsnummer




```
[assembly: AssemblyVersion("2.0.2.11")]
```


Versionierung

Incompatible

- Ein Shared Assembly ist grundsätzlich inkompatibel zum Client, wenn sich die Major- oder Minor-Version ändert
  - Beispiel: neues Produktrelease
  - Runtime wirft eine Type Load Exception

Versionierung

Maybe Compatible

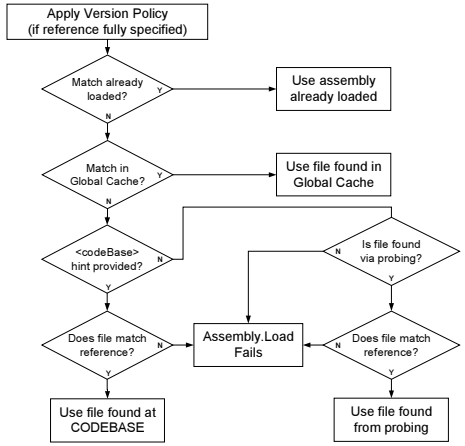
- Ein Shared Assembly kann kompatibel zum Client sein, wenn sich die Revision bei gleichbleibender Major- und Minor-Version ändert
  - Beispiel: Servicepack
  - Runtime versucht, das Assembly mit der höchsten Revisions- und Buildnummer zu laden

Versionierung

QFE – Quick Fix Engineering

- Ein Shared Assembly ist grundsätzlich kompatibel zum Client, wenn sich nur die Buildnummer ändert
  - In diesem Fall liegt ein sogenannter Quick Fix Engineering (QFE) vor
  - Beispiel: Security Hotfix
  - Runtime versucht, das Assembly mit der höchsten Revisions- und Buildnummer zu laden

Assembly Load



Zusammenfassung

- Sprachübergreifende Integration und einheitliche Fehlerbehandung über ein gemeinsames Typsystem
- Unterschiedliche Versionen gleicher Komponenten können parallel betrieben werden (→ Ende der “DLL Hölle”)
- Deployment von Anwendungen wird einfacher (→ Filecopy, keine Registry)

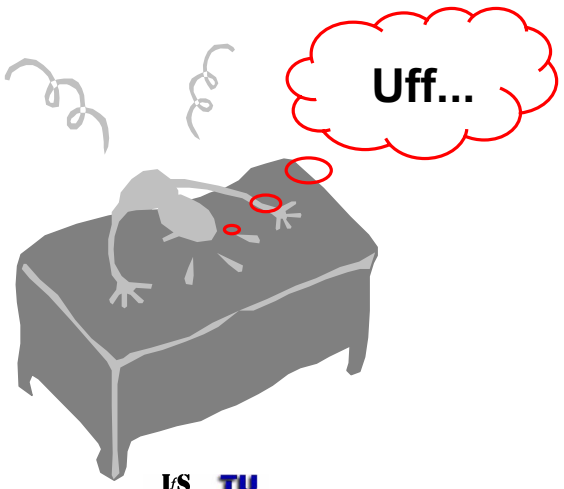
.NET auf einer Folie

Verteilte Applikationen

- Basierend auf Standards
- Allgegenwärtige Services
- Security und Privacy direkt integriert
- Device-unabhängig
- Programmiersprachen-unabhängig
- Windows-Plattform-unabhängig
- Sehr guter Entwicklersupport (Visual Studio, MSDN)

Robert Bruckner I/S TU 65

Fragen?



Robert Bruckner I/S TU 66