

D Überblick über die 13. Übung

D Überblick über die 13. Übung

- .NET Anwendungen bauen und binden
 - ◆ Assemblies
 - ◆ Anwendungskonfigurationsdatei
 - ◆ Signieren, Versionierung
 - ◆ Binden
- Advanced .NET Remoting
 - ◆ Remotekonfigurationsdatei
 - ◆ Metadaten-Assemblies
 - ◆ Contexts

1 Assemblies

D.1 .NET-Anwendungen bauen und binden

- private und gemeinsam benutzte Assemblies
- Metadaten werden im *Manifest* dem Assembly beigelegt
 - Assembly ist selbstbeschreibend
- Manifest enthält Metadaten der enthaltenen Typen und folgenden Informationen:
 - ◆ Version
 - ◆ Umgebungsinformationen (Sprache)
 - ◆ referenzierte Typen
 - ◆ Abhängigkeiten
- *multifile Assembly*: besteht aus mehreren Dateien

D.1 .NET-Anwendungen bauen und binden

D.1 .NET-Anwendungen bauen und binden

1 Assemblies

- Anwendung oder Bibliothek
 - z.B. unter Windows: ausführbare Datei!
- entspricht einer Sammlung von Klassen, Strukturen und Typen
- Assembly enthält plattformunabhängigen Code
 - ◆ *Common Intermediate Language* (CIL) oder *MSIL*
 - ◆ wird von JIT-Compiler übersetzt
 - ◆ → *portable execution file* (PE)

2 Manifest

D.1 .NET-Anwendungen bauen und binden

- enthält Metainformationen und Referenzen auf andere Assemblies
- Manifest eines Assemblies betrachten:

```
Mono> monodis Customer.exe
Win> ildasm Customer.exe
```

- Ausgabe:

```
.assembly extern mscorlib
{
    .ver 1:0:3300:0
}
.assembly extern Bank
{
    .ver 1:0:0:1
}
.assembly 'Customer'
{
    .hash algorithm 0x00008004
    .ver 0:0:0:0
}
.namespace Customer
{ ...
```

3 Auflösen der Abhängigkeiten

- Suchen der referenzierten Assemblies (*probing*)
- Teil des Bindungsprozesses
- beeinflussbar durch Anwendungsconfigurationsdatei

3 Der Probing Vorgang

- in Pseudocode:

```
function ProbeForAssembly
    ( AsmName, AppBase, Culture, PrivatePath)

    for each EXT in {"dll", "exe"}
        search (AppBase\AsmName.EXT);

        if (Culture == "neutral")
            search(AppBase\AsmName\AsmName.EXT);
        else
            search(AppBase\Culture\AsmName.EXT);
            search(AppBase\Culture\AsmName\AsmName.EXT);
        end if

        for each PATH in privatePath
            if (Culture == "neutral")
                search(AppBase\PATH\AsmName\AsmName.EXT);
            else
                search(AppBase\PATH\Culture\AsmName.EXT);
                search(AppBase\PATH\Culture\AsmName\AsmName.EXT);
            end if
        next PATH
    next EXT
end function
```

3 Die Anwendungsconfigurationsdatei

- Ort: Verzeichnis der Anwendung
- Name: <Name der Anwendung>.config
 - ◆ Beispiel.: Customer.exe.config
- Format: XML
- Aufgabe: enthält z.B. Informationen für den Bindevorgang
- Beispiel: Customer.exe.config

```
<configuration>
  <runtime>
    <assemblyBinding
      xmlns="urn:schemas-microsoft-com:asm.v1">
      <probing privatePath="libs;libs;morelibs" />
    </assemblyBinding>
  </runtime>
</configuration>
```

4 Gemeinsam genutzte Assemblies

- Identifikation: *starker Name* bestehend aus:
 - ◆ einfacher Textname (*friendly name*)
 - ◆ Umgebungsinformationen (*culture*)
 - ◆ Versionsnummer
 - ◆ öffentlicher Schlüssel
 - ◆ digitale Signatur
- Umgebungsinformation: "English", "German", meist aber "neutral" nur bei *satellite assemblies* ist die Sprache relevant.
- Vorteile eines starken Namens:
 - ◆ eindeutiger Name
 - ◆ mehrere Versionen möglich
 - ◆ Integritätsprüfung möglich
 - ◆ Herkunft überprüfbar

4 Gemeinsam genutzte Assemblies

- Position: *Global Assembly Cache (GAC)*
z.B.: `c:/windows/assembly`

- Assembly in GAC installieren:

- ◆ mit dem Windows Explorer (drag & drop)

- ◆ Mit `gacutil`

```
Win> gacutil /i Bank.dll
```

- ◆ von mono (noch) nicht unterstützt

5 Signieren

- Manifest der Bibliothek vor dem Signieren:

```
.assembly Bank
{
    .hash algorithm 0x00008004
    .ver 1:0:0:1
}
```

- ... und danach:

```
.assembly Bank
{
    .publickey = (00 24 00 00 04 80 00 00 94 00 00 00 06 02 00 00
00 24 00 00 52 53 41 31 00 04 00 00 01 00 01 00
6F 9A D3 D0 71 02 DA C8 AB B1 56 30 E7 30 2D 72
3F 89 01 86 9A BB 7A 14 9D D6 C2 E4 17 D4 73 7F
DD D4 0F C1 7A 0D 5F 3A 7E 5A 08 B3 B5 7F BD A6
BA AD BC 80 8F 5F 73 E7 B0 4F 12 84 D5 CD 72 3B
E2 93 6F 02 FE 23 C1 31 2E FE 40 DA 77 95 23 A9
3B 37 1E F8 2B 0F 45 A8 1C 87 CF B5 84 24 12 5E
60 6B 97 9D C1 73 FC 8F 3A 41 12 C0 89 87 CA E3
EF F6 94 D6 E0 37 64 21 BA 06 E5 3B 05 6D 82 C2)
    .hash algorithm 0x00008004
    .ver 1:0:0:1
}
```

5 Signieren

- Schlüsselpaar erzeugen mittels *Strong Name Utility*

```
Win> sn -k MyKey.snk
```

- in einem beliebigen Source-File (z. B. in `AssemblyInfo.cs`) die Schlüssel-Datei mittels `AssemblyKeyFile`-Attribut angeben:

```
using System.Reflection;
[assembly: AssemblyKeyFile(@"MyKey.snk")]
```

```
Win> csc /target:library Bank.cs
```

- oder mittels Assemblylinker hinzufügen:

```
Win> csc /target:module Bank.cs
...
Win> al /out:Bank.dll /keyfile:MyKey.snk Bank.netmodule
```

5 Signieren

- Manifest einer Anwendung...

- ◆ ...mit einer Referenz auf eine private Bibliothek:

```
.assembly extern Bank
{
    .ver 1:0:0:1
}
```

- ◆ ... mit einer Referenz auf eine signierte Bibliothek:

```
.assembly extern Bank
{
    .publickeytoken = (B4 26 2B 47 22 84 93 46 )
    .ver 1:0:0:1
}
```

5 Verzögertes Signieren

D.1 .NET-Anwendungen bauen und binden

- Verzögertes Signieren, falls kein privater Schlüssel vorliegt

- öffentlichen Schlüssel aus Schlüsselpaar extrahieren:

```
Win> sn -p MyKey.snk MyPublicKey.snk
```

- in Quellcode:

```
using System.Reflection;
[assembly: AssemblyDelaySigning(true)]
[assembly: AssemblyKeyFile(@"MyPublicKey.snk")]
```

- Schlüsselverifikation (beim Installieren im GAC & Binden) abschalten:

```
Win> sn -Vr Bank.dll
```

- ... und wieder aktivieren:

```
Win> sn -Vu Bank.dll
```

- Bibliothek nachträglich mit signieren (privaten Schlüssel ergänzen):

```
Win> sn -R Bank.dll MyKey.snk
```

Übungen zu Middleware

© Meik Felser, Andreas Weißel, Universität Erlangen-Nürnberg • Informatik 4, 2004

Binding.fm 2004-02-12 13.51

D.13

Reproduktion jeder Art oder Vervielfältigung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

6 Versionierung

D.1 .NET-Anwendungen bauen und binden

- Versionsnummer besteht aus 4 Teilnummern durch Punkt (.) oder Doppelpunkt (:) getrennt:
<major version>.<minor version>.<buildnumber>.<revision>

- wird durch ein Attribut festgelegt

```
[assembly: AssemblyVersion("1.0.0.1")]
```

- kann auch nur teilweise festgelegt werden (automatische Erzeugung):

```
[assembly: AssemblyVersion("1.0.*")]
```

- Keine Versionskontrolle bei privaten Assemblies

- Versionskontrolle beim Binden von existierenden Anwendungen beeinflussen durch:

- ◆ die Anwendungsconfigurationsdatei
- ◆ die systemweite Konfiguration
- ◆ eine Richtliniendatei des Herstellers (*publisher policy*)

Übungen zu Middleware

© Meik Felser, Andreas Weißel, Universität Erlangen-Nürnberg • Informatik 4, 2004

Binding.fm 2004-02-12 13.51

D.14

Reproduktion jeder Art oder Vervielfältigung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

6 Umleiten von Assemblyversionen durch die Anwendungsconfigurationsdatei

D.1 .NET-Anwendungen bauen und binden

- Customer.exe.config

```
<configuration>
  <runtime>
    <assemblyBinding
      xmlns="urn:schemas-microsoft-com:asm.v1">
      <probing privatePath="libs" />
      <dependentAssembly>
        <assemblyIdentity name="Bank"
          publicKeyToken="b4262b4722849346" />
        <bindingRedirect oldVersion="1.0.0.1-1.0.0.2"
          newVersion="1.0.0.3" />
      </dependentAssembly>
    </assemblyBinding>
  </runtime>
</configuration>
```

Übungen zu Middleware

© Meik Felser, Andreas Weißel, Universität Erlangen-Nürnberg • Informatik 4, 2004

Binding.fm 2004-02-12 13.51

D.15

Reproduktion jeder Art oder Vervielfältigung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

6 Umleiten von Assemblyversionen durch die systemweite Konfiguration

D.1 .NET-Anwendungen bauen und binden

- Systemweite Konfigurationsdatei:

```
<.NET install path>\Config\Machine.config
```

- Beispiel:

```
c:\windows\Microsoft.NET\Framework\v1.1.4322\CONFIG\Machine.config
```

- Unter Windows mittels msconfig.msc editieren

```
Win> mmc
c:\windows\Microsoft.NET\Framework\v1.1.4322\msconfig.msc
```

Übungen zu Middleware

© Meik Felser, Andreas Weißel, Universität Erlangen-Nürnberg • Informatik 4, 2004

Binding.fm 2004-02-12 13.51

D.16

Reproduktion jeder Art oder Vervielfältigung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

6 Umleiten von Assemblyversionen durch eine Richtliniendatei (publisher policy)

- Assembly mit speziellem Namen:
policy.<major>.<minor>.<assemblyname>
- Name entspricht der Version für welche die Konfiguration angewendet werden soll
- wird aus einer XML-Datei durch den Assembly Linker (al.exe) gebaut

```
Win> al /link:PublisherPolicy.xml
       /out:policy.1.0.Bank.dll
       /keyfile:MyKey.snk
       /v:1.0.0.0
```

- Format: XML, wie Anwendungsconfigurationsdatei

6 Anwendung der Richtliniendatei unterdrücken

- Verarbeitung der Richtliniendatei kann durch Anwendungsconfiguration unterdrückt werden.

- ◆ für die gesamte Anwendung: (Eintrag unter <assemblyBinding>)

```
<configuration>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <publisherPolicy apply="no" />
    </assemblyBinding>
  </runtime>
</configuration>
```

- ◆ für einzelne Bibliotheken (Eintrag unter <dependentAssembly>):

```
<configuration>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name="Bank"
                          publicKeyToken="b4262b4722849346" />
        <publisherPolicy apply="no" />
      </dependentAssembly>
    </assemblyBinding>
  </runtime>
</configuration>
```

6 Umleiten von Assemblyversionen

- Versionsumleitung ist nicht schachtelbar:
 - ◆ Beispiel
 - Anwendungsconfiguration
leitet Version **1.0.0.0** auf Version **1.0.0.1** um
 - Systemconfiguration
leitet Version **1.0.0.1** auf Version **1.0.0.2** um
 - ◆ trotzdem wird die Version 1.0.0.1 verwendet

- Konfigurationsvorrang:

- (1) Anwendungsconfiguration
- (2) Richtliniendatei
- (3) Systemconfiguration

7 Position der Assemblies

- GAC
- Anwendungsverzeichnis oder ein Unterverzeichnis

- ◆ <probing privatePath="libs" />

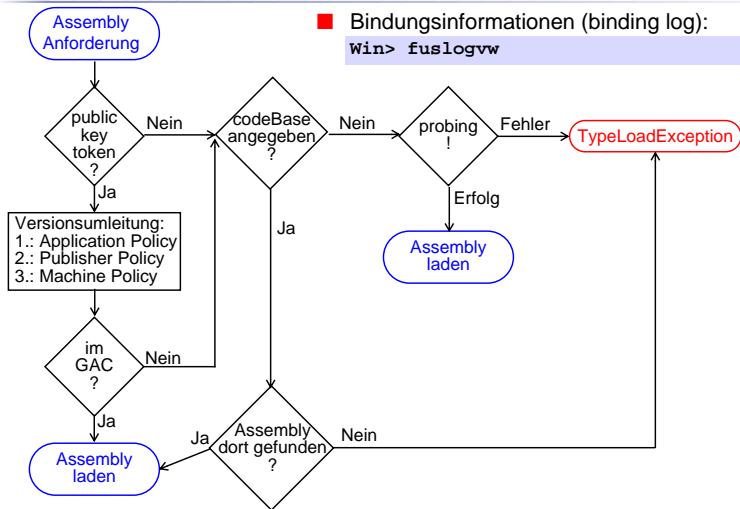
- beliebiges Verzeichnis oder Web-Seite:

```
[...]
<dependentAssembly>
  <assemblyIdentity name="Bank"
                    publicKeyToken="b4262b4722849346" />
  <bindingRedirect oldVersion="1.0.0.1-1.0.0.2"
                  newVersion="1.0.0.3" />
  <codeBase version="1.0.0.3"
            href="file://c:/MyLib.dll"/>
</dependentAssembly>
[...]
```

- ◆ private Assemblies müssen im Anwendungsverzeichnis oder in einem Unterverzeichnis liegen (<codeBase> dann ohne Versionsnummer).

8 Binden - Übersicht

D.1 .NET-Anwendungen bauen und binden



Übungen zu Middleware

© Meik Felser, Andreas Weißel, Universität Erlangen-Nürnberg • Informatik 4, 2004

Binding.fm 2004-02-12 13.51

D.21

Reproduktion jeder Art oder Vervielfältigung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.2 Advanced .NET Remoting

1 Remotekonfigurationsdatei

■ Server:

```
<application>
  <service>
    ...
  </service>

  <channels>
    <channel port=4711
      type="System.Runtime.Remoting.Channels.Http.HttpChannel,
        System.Runtime.Remoting, Version=1.0.3300.0,
        Culture=neutral, PublicKeyToken=b77a5c561934e089" />
    </channel>
  </channels>
</application>
```

■ Client:

```
<application>
  <client>
    ...
  </client>
  <channels>
    <channel type="..." />
  </channels>
</application>
```

Übungen zu Middleware

© Meik Felser, Andreas Weißel, Universität Erlangen-Nürnberg • Informatik 4, 2004

Binding.fm 2004-02-12 13.51

D.23

Reproduktion jeder Art oder Vervielfältigung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.2 Advanced .NET Remoting

D.2 Advanced .NET Remoting

1 Remotekonfigurationsdatei

- zur einfachen Konfiguration der Kanäle und registrierten Objekte
- Format: XML
- Position und Name: beliebig, meist jedoch in Anwendungskonfigurationsdatei

```
<configuration>
  <system.runtime.remoting>
    ...
  </system.runtime.remoting>
</configuration>
```

■ Einlesen:

```
RemotingConfiguration.Configure("Customer.exe.config");
```

- Von mono (noch) nicht unterstützt

Übungen zu Middleware

© Meik Felser, Andreas Weißel, Universität Erlangen-Nürnberg • Informatik 4, 2004

Binding.fm 2004-02-12 13.51

D.22

Reproduktion jeder Art oder Vervielfältigung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.2 Advanced .NET Remoting

1 Remotekonfigurationsdatei

■ Kanal-Vorlagen:

```
<configuration>
  <system.runtime.remoting>
    <application>
      <channels>
        <channel port=4711 ref="MyChannel" />
      </channels>
    </application>
    <channels>
      <channel id="MyChannel" type="..." />
    </channels>
  </system.runtime.remoting>
</configuration>
```

■ vordefinierte Vorlagen z.B.: http, tcp

```
<channels>
  <channel port=4711 ref="http" />
</channels>
```

Übungen zu Middleware

© Meik Felser, Andreas Weißel, Universität Erlangen-Nürnberg • Informatik 4, 2004

Binding.fm 2004-02-12 13.51

D.24

Reproduktion jeder Art oder Vervielfältigung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Remotekonfigurationsdatei

- serveraktiviertes Objekt: Server (keine Referenz auf DLL mehr nötig!):

```
<application>
  <service>
    <wellknown mode="Singleton"
      type="BankLibrary.Bank, Bank"
      objectUri="MyURI.soap" />
  </service>
  <channels>
    <channel port=4711 type="http" />
  </channels>
</application>
```

- serveraktiviertes Objekt: Client

```
<application>
  <client displayName="BankClient">
    <wellknown type="BankLibrary.Bank, Bank"
      url="http://localhost:4711/MyURI.soap" />
  </client>
  <channels>
    <channel type="http" />
  </channels>
</application>
```

1 Remotekonfigurationsdatei

- clientaktiviertes Objekt: Server

```
<application>
  <service>
    <activated type="BankLibrary.Account, Bank" />
  </service>
  <channels>
    <channel port=4711 id="http" />
  </channels>
</application>
```

- clientaktiviertes Objekt: Client

```
<application>
  <client url="http://localhost:4711" >
    <activated type="BankLibrary.Account, Bank" />
  </client>
  <channels>
    <channel type="http" />
  </channels>
</application>
```

2 Metadaten-Assembly

- Problem: Client benötigt Metadaten zur Erstellung des Proxies

- Alternativen:

- ◆ Assembly muss auch auf Clientseite vorhanden sein
- ◆ entferntes Objekt von Interface ableiten:
Interface Assembly ist sowohl beim Client als auch beim Server vorhanden
 - Nachteil: von einem Interface kann man kein Objekt erzeugen
 - → keine clientaktivierte Objekte
 - → serveraktivierte Objekte müssen mittels `Activator.GetObject` angesprochen werden
- ◆ Metadaten Assembly:
Assembly mit gleichem Namen und gleichen Klassen inkl. Methoden wie auf Serverseite, jedoch ohne Implementierung
 - manuelle Erzeugung (s. vorherige Übung):
`throw new NotSupportedException` in jeder Methode)
 - Tool: `soapsuds.exe` erzeugt Metadaten-Assemblies

2 Metadaten-Assembly - soapsuds auf Clientseite

- Auf Clientseite:

```
Win> soapsuds -url:http://localhost:4711/MyURI.soap?wsdl
      -oa BankProxy.dll
```

- erzeugt ein Proxy-Assembly, Proxy kann mittels `new` instantiiert werden, Serveradresse ist fest codiert (Client benötigt keine Remoting-Konfig.)
 - ◆ wrapped-Proxy (um den transparenten Proxy herum)
 - ◆ Metadaten werden von der entfernten Anwendung geladen
 - ◆ nur mit Http-Channel und SOAPFormatter möglich

- Alternative Ausgabeformate:

- ◆ C#:

```
Win> soapsuds -url:../MyURI.soap?wsdl -gc BankProxy.cs
```

- ◆ WSDL (Web Services Description Language, sprich: "wiz-dull"):

```
Win> soapsuds -url:../MyURI.soap?wsdl -os bank.wsdl
```

XML-basierte Sprache zur Beschreibung von Interfaces

2 Metadaten-Assembly - soapsuds auf Clientseite

- bei clientaktivierten Objekten:

```
Win> soapsuds -url:http://localhost:4711/
RemoteApplicationMetadata.rem?wsdl
-oa BankProxy.dll
```

- ◆ keine vordefinierte URI
- ◆ RemoteApplicationMetadata.rem als spezielle Adresse

- liefert Metadaten für *alle* verwendbaren Objekte inkl. "well-known" Objekte
- Nachteil: zu den Objekten werden keine Informationen über mögliche Konstruktoren erzeugt (nur Standardkonstruktor möglich)

3 Contexts

- Ein Application Domain kann mehrere Contexte enthalten, mindestens den "default context"
 - ◆ Kapselung innerhalb einer Application Domain, kontrollierte Ausführung, auch bei Fernaufrufen
- Ableiten von "System.ContextBoundObject"
 - ◆ Die Laufzeitumgebung prüft vor der Aktivierung eines Kontext-gebundenen Objekts, ob der Kontext in Ordnung ist: `IContextAttribute.IsContextOK`
 - ◆ Falls nicht, wird mit `IContextAttribute.GetPropertiesForNewContext` die erforderliche Umgebung für jedes Attribut erzeugt
- Anwendungen
 - ◆ z.B. Logging, Sicherheit, Transaktionen
 - ◆ Synchronisierung, z.B. nur ein Thread darf in diesem Context aufgeführt werden, Objekte selbst müssen sich nicht um Synchronisierung kümmern

2 Metadaten-Assembly - soapsuds auf Serverseite

- auf Serverseite:

```
Win> soapsuds -ia Bank -oa BankProxy.dll
```

- Proxy-Generierung ausschalten (nur Metadaten erzeugen):

```
Win> soapsuds ... -nowp
```

- oder nur einzelne Typen:

```
Win> soapsuds
-types:BankLibrary.Bank,Bank;BankLibrary.Account,Bank
-oa BankProxy.dll
```

- Proxy mit fest codierter URL möglich:

```
Win> soapsuds
-types:BankLibrary.Bank,Bank,
http://localhost:4711/MyURI.soap;
BankLibrary.Account,Bank
-oa BankProxy.dll
```