

F .Net

F.1

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

1 Überblick



- Anwendungsplattform für alle Microsoft-Betriebssysteme
 - ◆ inspiriert von der Java-Entwicklung
 - ◆ Sprachunabhängigkeit
 - ◆ dynamisches Code-Laden
 - ◆ verteilte Anwendungen
 - ◆ Integration der XML-Technologie
 - Web-Services, XSLT ...
- Wesentliche Teile sind standardisiert: ECMA-Standard
- Implementierungen
 - ◆ Microsoft .Net Framework
 - ◆ Open-Source-Implementierung Mono

F.2

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

2 Common-Language-Runtime



- Unterstützung verschiedener Sprachen
 - ◆ C#, J#, VisualBasic, ...
 - ◆ Compiler übersetzt jeweils in Zwischensprache
 - MSIL (Microsoft Intermediate Language)
 - abstrakte Stackmaschine

- Common-Language-Runtime-Environment
 - ◆ Ausführung von MSIL-Programmen
 - Loader
 - (JIT-)Compiler und Verifier
 - MSIL-Programme werden nie interpretiert (bei Microsoft)
 - Übersetzung vor oder während der Ausführung
 - Sicherheitsüberprüfungen
 - Multithreading

F.3

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

2 Common-Language-Runtime (2)



- Managed-Code
 - ◆ MSIL-Programm

- Unmanaged-Code
 - ◆ Brücke zu anderen Code-Teilen
 - Standard-DLLs
 - COM+-Aufrufe

- Managed-Data
 - ◆ Garbage-Collection der Datenbereiche im Managed-Code

- Unmanaged-Data
 - ◆ manuell verwaltete Datenbereiche im Managed-Code
 - mit malloc/free verwaltet

F.4

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

2 Common-Language-Runtime (3)



■ Sprachunabhängige Bibliotheken

- ◆ in allen Sprachen stehen die gleichen Bibliotheken zur Verfügung
- ◆ keine sprachspezifischen Bibliotheken vorgesehen (z.B. Java, C++)

■ Ausschnitt

System	System.Drawing
System.IO	System.Drawing.Text
System.Net	System.Drawing.Drawing2D
System.Collections	System.Web.Services
System.Reflection	System.Web.Services.Discovery
System.Threading	System.Web.Security
System.Xml	System.Web.UI
System.Xml.Xslt	System.Web.UI.WebControls.CheckBoxList
System.Xml.XPath	System.Windows.Forms
System.Data	System.Windows.Forms.RadioButton
System.Data.ADO	System.Runtime.Remoting
System.Data.SQL	System.Runtime.Serialization

F.5

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

2.1 Assembly



■ Container für Module

- ◆ äquivalent zu einer Menge von Java JAR-Files
- ◆ ersetzt DLL- und EXE-Files
- ◆ enthält:
 - Manifest (Import-, Export- und Sicherheitsanweisungen, Verweise auf andere Module)
 - Program-Executables (MSIL Code-Bereiche)
 - Informationen für Reflection
- ◆ mit Versionskennung versehen
 - im gleichen System mehrere Versionen der gleichen Assembly möglich

■ Modulweise dynamisch ladbar

- ◆ von Platte
- ◆ über das Internet

F.6

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

2.1 Assembly (2)



- Laden einer Assembly
 - ◆ dynamisch bei Bedarf
 - ◆ Application-Domain (Anwendungsdomäne)
 - abgegrenzter Bereich innerhalb der CLR für eine Anwendung
 - Laden in eine Application-Domain (je Domain verschiedene Version einer Assembly möglich)
 - Prozess kann neue Application-Domains erzeugen
 - mehrere Application-Domains pro Prozess möglich
 - Sicherheit durch CLR Verifier gewährleistet

F.7

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

2.2 Kontext und Application-Domain



- Application-Domain unterteilt in Kontexte
 - ◆ Kontext bündelt gleich zu behandelnde Objekte
 - z.B. Thread-Behandlung, JIT-Aktivierung, Synchronisation
 - ◆ Kontext besitzt abfragbare Properties
 - ◆ Eingriffsmöglichkeit auf kontextlokale Methodenaufrufe von außen
 - Prä- und Postoperationen
- Interaktionen zwischen Kontexten und Application-Domains
 - ◆ erfordern spezielle Aufrufbehandlung
 - Marshalling der Parameter und Ergebnisse

F.8

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

3 .Net-Remoting



- Framework für Kommunikation zwischen (entfernten) Kontexten und Application-Domains
- Unterscheidung von drei Objektsorten
 - ◆ verteilungsignorante Objekte
 - weder entfernt aufrufbar noch als Parameter für entfernte Aufrufe nutzbar
 - ◆ serialisierbare Objekte
 - als Parameter entfernter Aufrufe nutzbar (*Call-by-Value-Semantik*)
 - Klasse wird als serialisierbar markiert (Attribut: `[Serializable]`)
 - Marshalling serialisiert zugehöriges Objekt
 - ◆ entfernt referenzierbare Objekte
 - als Parameter nutzbar (*Call-by-Reference-Semantik*)
 - entfernt aufrufbar
 - Klasse muss von `MarshalByRefObject` erben
 - Marshalling serialisiert Referenz auf zugehöriges Objekt

F.9

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

3 .Net-Remoting (2)



- Serverobjekte
 - ◆ Client-activated object
 - vom Client erzeugte Objekte
 - ◆ Well-known object
 - bereits instanziiert und aufrufbereit
 - ◆ Server-activated object
 - Objekt entsteht erst beim Aufruf
 - Singleton-Modus: ein Objekt behandelt alle Aufrufe
 - SingleCall-Modus: ein Objekt entsteht pro Aufruf
- Zugang zu entfernten Objekten
 - ◆ Aufruf eines *Activators* mit Objektbezeichnung (z.B. URL)
 - ◆ es entsteht ein Proxy

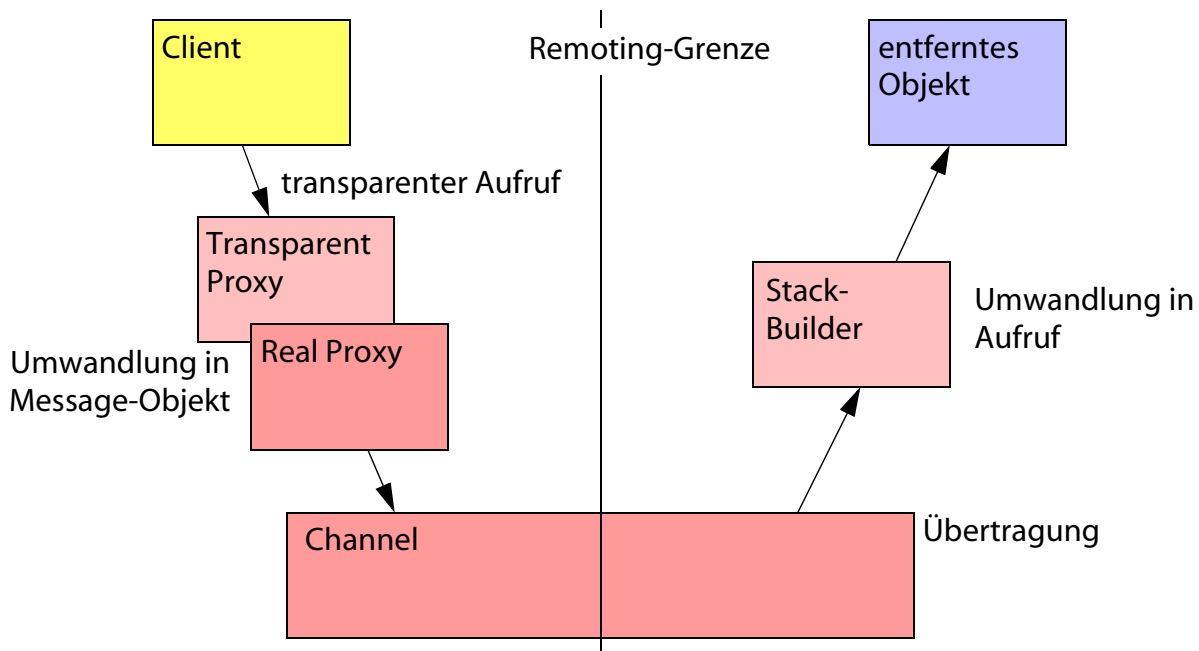
F.10

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

3.1 Architektur von .Net-Remoting



Aufrufpfad



F.11

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

3.1 Architektur von .Net-Remoting (2)



Transparent Proxy

- ◆ automatisch generiert aus Schnittstelle
- ◆ wandelt Aufruf in ein Message-Objekt um (vgl. DII von CORBA)

Real Proxy

- ◆ Ansatzpunkt für kontextabhängige Erweiterungen
- ◆ Default: Weitergabe des Message-Objekts an Channel

Channel

- ◆ besteht auf jeder Seite aus Sink-Objekten
- ◆ Sink-Kette pro Kontext beeinflussbar
- ◆ Sinks bestimmen Protokoll, Übertragungsformat und sonstige Seiteneffekte

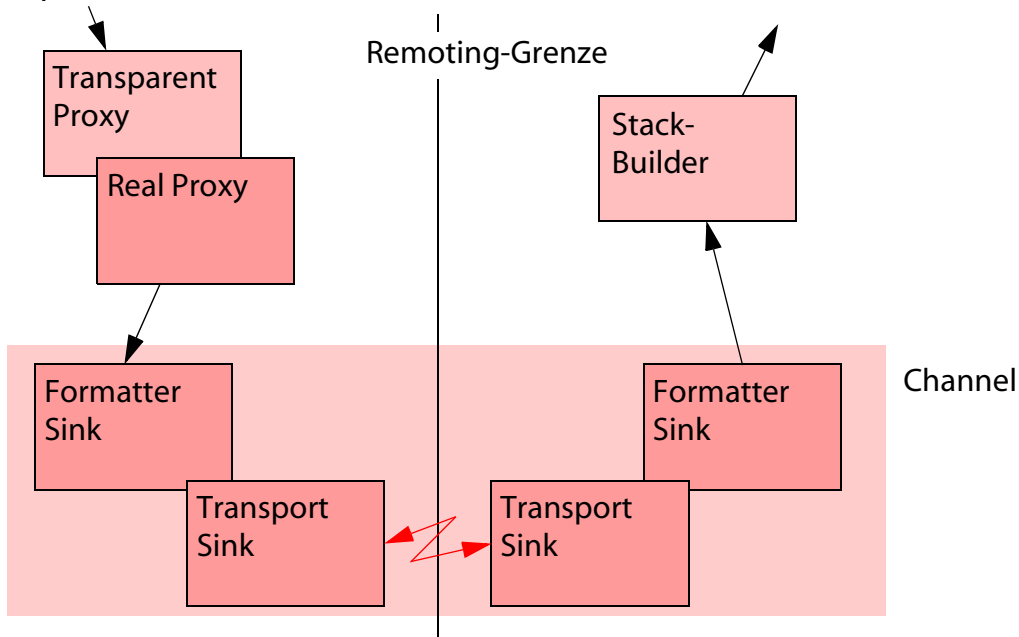
F.12

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

3.1 Architektur von .Net-Remoting (3)



■ Beispiel für Channel



F.13

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

3.1 Architektur von .Net-Remoting (4)



■ Einsatz verschiedener RPC-Protokolle

- ◆ verschiedene Formatter-Sinks
 - SOAP für Web-Services
 - Formatter für .Net-Remoting-Protocol
 - (Formatter für GIOP)

■ Einsatz verschiedener Transportprotokolle

- ◆ verschiedene Transport-Sinks
 - TCP/IP für .Net-Remoting-Protocol (oder für IIOP)
 - HTTP für Web-Services

■ Weitere Sinks in der Kette

- ◆ z.B. für Transaktionen, Gruppenkommunikation etc.

F.14

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

3.2 Objektreferenz



- Darstellung von entfernten Objektreferenzen
 - ◆ „ObjRef“-Objekt
 - ◆ enthält
 - Typbezeichner für entferntes Objekt
 - Assembly-Informationen
 - Basisklassen (und Schnittstellen)
 - Objekt-URL zur Aktivierung
 - Channel-Informationen (welche Sinks müssen enthalten sein)
 - ◆ ObjRef-Objekt wird By-value übergeben
 - daraus kann erzeugt werden:
 - Proxies (per Reflection aus Typbezeichner)
 - Sink-Objekte aus Channel-Informationen und Kontext-Properties

F.15

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

4 Erzeugung entfernter Objekte



- Klasse für verteilte Objekte muss von `MarshalByRefObject` erben
- ◆ Beispiel (in C#): Hello-Objekt

```
class Hello : MarshalByRefObject
{
    public string sayHello()
    {
        return "Hallo";
    }
}
```

- Zwei unterscheidbare Vorgehensweisen
 - ◆ Objekterzeugung durch Server: Server-activated objects
 - ◆ Objekterzeugung durch Client: Client-activated objects

F.16

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

4 Erzeugung entfernter Objekte (2)



- Erzeugung von Instanzen durch `new()`-Aufruf
 - ◆ lokale Objekterzeugung
 - Instanz wird automatisch exportiert
 - vorherige Konfiguration/Erzeugung von Channels notwendig
 - ◆ entfernte Objekterzeugung (*Client-Activated Object, CAO*)
 - durch entsprechende Konfiguration führt `new()`-Aufruf zur Erzeugung einer Instanz auf dem Server
 - typ- bzw. klassenweise Konfiguration des Erzeugungsortes
 - ◆ Erzeugung über Entwurfsmuster der Fabrik
- Erzeugung durch Referenzermittlung an einem lokalen *Activator*
 - ◆ entfernte Objekterzeugung (*Server-Activated Object, SAO*)
 - nach entsprechender Konfiguration instanziiert Server Objekt bei Bedarf (*Single-Call Object, Singleton Object, Published Object*)

F.17

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

4.1 Single-Call Object



- Eigene Objektinstanz für jeden Aufruf
 - ◆ Objektinstanz ohne speicherbaren Zustand
- Server-Seite
 - ◆ Erzeugung des Channels (hier: HTTP/SOAP-Kanal)

```
ChannelServices.RegisterChannel( new HttpChannel(4711) );
```

 - alternativ: TCP-Channel mit binärem RPC-Protokoll
 - ◆ Registrierung des Objekttyps

```
RemotingConfiguration.RegisterWellKnownServiceType(
    typeof>Hello), "SayHello.soap",
    WellKnownObjectMode.SingleCall );
```

F.18

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

4.1 Single-Call Object (2)

■ Client-Zugriff

◆ Ermitteln der Referenz durch Aktivierung am lokalen Activator

- Beispiel

```
sayer= (Hello) Activator.GetObject( typeof(Hello),  
    "http://localhost:4711/SayHello.soap" );
```

- Referenzerzeugung abhängig vom registrierten Objekttyp

◆ Lokales Activator-Objekt

- bekommt Objekt-URI
- erzeugt daraus client-seitige .Net-Architektur: Proxy und Sinks

■ Lifecycle-Management bei Single-Call Objects im Server

◆ automatische Erzeugung einer Instanz für jeden Aufrufe

◆ nebenläufige Aufrufe möglich

- Threading-Einstellung im jeweiligen Kontext

F.19

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

4.2 Singleton Object

■ Genau eine Objektinstanz für alle Aufrufe

- ◆ reiner Dienst
- ◆ Zustand möglich

■ Server-Seite

- ◆ Erzeugung des Channels
- ◆ Registrierung des Objekttyps

```
RemotingConfiguration.RegisterWellKnownServiceType(  
    typeof(Hello), "SayHello.soap",  
    WellKnownObjectMode.Singleton );
```

■ Client-Zugriff

- ◆ identisch zu Single-call object

F.20

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

4.3 Published Object



- Genaue eine Objektinstanz
 - ◆ reiner Dienst (wie Singleton Object)
 - ◆ aber: Instanz wird vorab erzeugt und registriert

- Server-Seite
 - ◆ Erzeugung des Channels
 - ◆ Erzeugung und Registrierung des Singleton

```
Hello sayer= new Hello();  
RemotingServices.Marshal( sayer, "SayHello.soap" );
```

- Client-Zugriff
 - ◆ identisch zu Singleton object

F.21

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

4.4 Direkte Objekterzeugung



- Client kann beliebige Objektinstanzen eines Typs erzeugen und nutzen

- Server-Seite
 - ◆ Erzeugung eines Channels
 - ◆ Registrierung eines Typs

```
RemotingConfiguration.ApplicationName= "MySayer";  
RemotingConfiguration.RegisterActivatedServiceType(  
    typeof(Hello) );
```

- Client-Zugriff
 - ◆ Erzeugung eines Channels
 - ◆ Registrierung eines Client-Typs

```
RemotingConfiguration.RegisterActivatedClientType(  
    typeof(Hello), "http://localhost:4711/MySayer" );  
...  
M= new Hello(); // erzeugt entferntes Objekt
```

F.22

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

4.5 Erzeugung über Fabrik



- Factory-Pattern (Entwurfsmuster der Fabrik)
 - ◆ Registrierung eines Singleton (die Fabrik)
 - ◆ Aufruf einer `create()`-Methode an der Fabrik
 - Erzeugung eines neuen Objekts im Server
 - Rückgabe der Referenz an Aufrufer
 - automatische Zuweisung einer Kommunikationsadresse durch .Net-Remoting

F.23

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

4.6 Konfiguration



- Konfigurationsdatei statt explizitem Erzeugen und Registrieren
 - ◆ XML-Datei (vgl. Deployment-descriptor)
- Beispiel: Singleton object im Server

```
<configuration>
  <system.runtime.remoting>
    <application>
      <channels>
        <channel ref="http" port="4711" />
      </channels>
      <service>
        <wellknown mode="Singleton"
          type="Hello, Server"
          objectURI="SayHello.soap" />
      </service>
    </application>
  </system.runtime.remoting>
</configuration>
```

Assembly

F.24

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

4.6 Konfiguration (2)



■ Laden der Konfiguration im Server

- ◆ Angabe des Dateinamens

```
Remoting.Configuration.Configure( "HelloServer.exe.config" );
```

■ Ähnliche Konfiguration im Client

■ Weitere Konfigurationsmöglichkeiten

- ◆ Wechsel des Channels (z.B. von HTTP auf TCP)
- ◆ eigene Channel-Implementierungen
 - Angabe der Implementierungsklassen für Transport- und Formatter-Sinks
- ◆ Angaben zum Lifecycle-Management

F.25

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

5 Assembly-Struktur



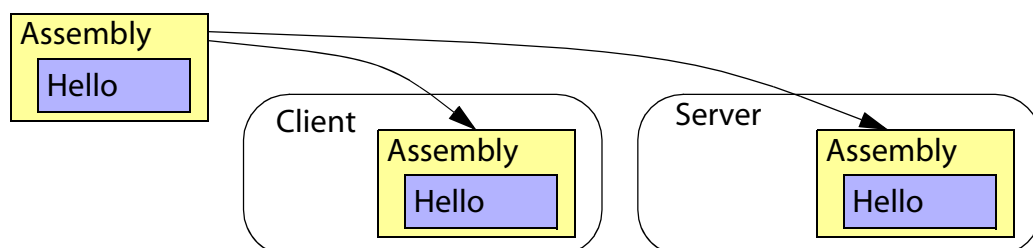
■ Verteilter Objektzugriff erfordert:

- ◆ gemeinsam genutzte Typinformationen in allen beteiligten Knoten

5.1 Gemeinsam genutzte Klasse

■ Shared assembly für Objektklasse

- ◆ über das Netz zugreifbar (Webserver, URL)
- ◆ Objektklasse ist sowohl Client- als auch Server-Knoten bekannt



F.26

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

5.1 Gemeinsam genutzte Klasse



- * Vorteil
 - ◆ einfach
- ▲ Nachteile
 - ◆ Client kann Objekt auch direkt instanziiieren
 - ◆ Benutzer auf Client-Computer können Code disassemblieren

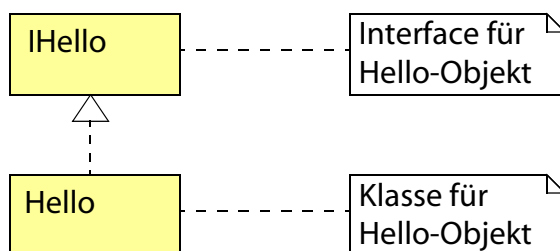
F.27

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

5.2 Gemeinsam genutztes Interface



- Objekt bekommt Interface für entfernten Zugriff (vgl. RMI)



- ◆ Shared assembly für Interface, private Assembly für Klasse (nur Server)

- * Vorteil
 - ◆ Trennung möglich
- ▲ Nachteil
 - ◆ Referenz nicht weitergebbar an Dritte
 - Cast auf Interface lässt sich dort nicht durchführen

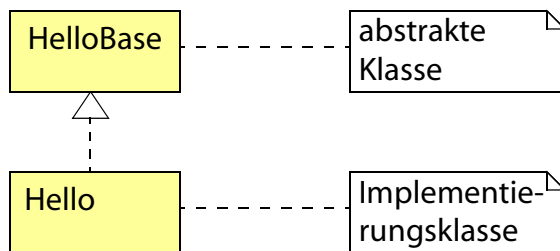
F.28

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

5.3 Gemeinsam genutzte Basisklasse



■ Abstrakte Basisklasse für Implementierungsklasse



- ◆ Shared assembly für Basisklasse, private Assembly für Implementierungsklasse (nur Server)

* Vorteil

- ◆ Trennung möglich
- ◆ Objektreferenzen weitergebbar

F.29

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

5.3 Gemeinsam genutzte Basisklasse (2)



▲ Nachteil

- ◆ lediglich mit Activator nutzbar
- ◆ keine direkte Objekterzeugung möglich
 - Compiler unterbindet Erzeugung von Objekten einer abstrakten Klasse

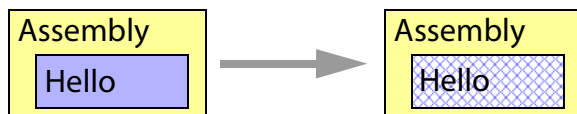
F.30

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

5.4 Generierte Metadaten



- Werkzeugunterstützung
 - ◆ Programm SoapSuds erzeugt zweite Assembly für reine Metadaten



- Wrapped-proxy-Ansatz
 - ◆ SoapSuds integriert Kommunikationsadresse in Metadaten
 - ◆ Client-Zugriff
 - ```
m= new Hello();
```
    - erzeugt Objekt im Server
- Nonwrapped-proxy-Ansatz
  - ◆ Client-Zugriff über `Activator` oder über Typ-Registrierung

F.31

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

## 6 Lebenszeitverwaltung



- Verwaltung über Leases
  - ◆ Abräumen des Objekts sobald Lease abgelaufen (Garbage Collection)
  - ◆ Default-Einstellungen
    - Lease gültig bis 5min nach Objekterzeugung
    - Erneuerung der Lease bis zu 2min bei jedem Aufruf
- Sponsoren
  - ◆ registrierte Ansprechpartner bei Ablauf einer Lease
    - .Net-Remoting kontaktiert Sponsor vor dem Löschen
  - ◆ innerhalb von 2min (Default) muss Sponsor Verlängerung der Lease bestätigen

F.32

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

## 6.1 Konfiguration der Einstellungen

---



- Anwendungsabhängige Änderung der Einstellungen
  - ◆ programmatisch: Zuweisung von Zeitspannen an gewisse Systemvariablen
    - z.B. `LifetimeServices.LeaseTime = ...`
  - ◆ deklarativ: XML-Elemente in Konfigurationsdatei
    - z.B. `<lifetime leaseTimeout="..." />`

F.33

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

## 6.1 Konfiguration der Einstellungen (2)

---



- Klassenabhängige Änderung der Einstellungen
  - ◆ programmatisch: Methode `InitializeLifetimeService()` in Basisklasse `MarshalByRefObject`
    - erzeugt Objekt mit Interface `ILease`
      - enthält Einstellungsparameter
    - Rückgabe eines `null`-Zeigers möglich
      - keine Garbage-Collection
  - ◆ deklarativ: XML-Elemente in Konfigurationsdatei
    - Setzen der Parameter muss in `InitializeLifetimeService()` erfolgen
    - Parameter können aus Konfigurationsdatei stammen
      - anwendungsabhängige Konfigurationsparameter (vgl. Java Properties)
      - z.B. `<appSettings>`
        - `<add key="Hello_LeaseTime" value="..." /> ...`

F.34

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

## 6.2 Sponsoren

---



- Automatische Befragung des Sponsors über Lease-Verlängerung
  - ◆ gibt neue Lease-Lebenszeit zurück
    - Lebenszeit gleich Null: Objekt wird aufgeräumt
  
- Client-seitige Sponsoren
  - ◆ entspricht periodischem Prüfen, ob Client noch vorhanden
    - entspricht früherem DCOM-Ansatz
  - ◆ Achtung: Client wird kurzzeitig zum Server, Server zum Client
    - Client muss Channel für Anfragen besitzen
      - z.B. HTTP-Channel: Portnummer angeben oder `port="0"`
  - ◆ Achtung: Sponsoren sind Remote-Objects
    - benötigen eigenes Lease-Management (evtl. durch sich selbst)

F.35

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

## 6.2 Sponsoren (2)

---



- Server-seitige Sponsoren
  - ◆ auf gleichem oder anderem Server-Knoten
  - ◆ Bindung der Lebenszeit an anwendungsabhängige Bedingungen
    - z.B. Verweigerung der Lease-Erneuerung, falls essenzielle Anwendungskomponenten abgestürzt oder unerreichbar

F.36

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

## 7 Vergleich mit theoretischen Anforderungen

---



- \* .Net basiert auf einem monolithischen Objektmodell
- Weltweit eindeutige **Objektbezeichner** zum Parametertransport
  - ◆ ObjRef-Objekt
- **Erzeugung** neuer verteilter Objekte
  - ◆ `new()`-Operator oder über Activator
  - ◆ transparente entfernte Objekterzeugung möglich
- Schnittstellenspezifische **Stellvertreterobjekte**
  - ◆ Proxy-Objekte im Defaultfall generisch (vgl. Java RMI 1.5)
  - ◆ Sink-Objekte im Defaultfall generisch

F.37

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

## 7 Vergleich mit theoretischen Anforderungen (2)

---



- **RPC**-basiertes Kommunikationsprotokoll
  - ◆ SOAP und TCP-basiertes RPC-Protokoll
- **Parameterübergabe** von Objektreferenzen
  - ◆ Übermittlung eines ObjRef-Objekts
- **Automatische Generierung** der Stellvertreter
  - ◆ keine Generierung erforderlich
- **Namensdienst**
  - ◆ normalerweise kein Namensdienst
  - ◆ stattdessen initiale Konfiguration

F.38

© 2002-2006, Franz J. Hauck, Verteilte Systeme, Univ. Ulm, [2007w-AvO-F-Net.fm, 2008-01-13 19.15] <http://www-vs.informatik.uni-ulm.de/teach/ws06/avo/>

## 8 Literatur

---



### ■ .Net

◆ I. Rammer: Advanced .Net Remoting. Apress, 2002.

◆ Articles on Microsoft Developer Network:

.NET Framework "Deploying & Distribution".

<<http://msdn.microsoft.com/netframework/using/deploying/default.aspx>>

**F.39**