

Remoting mit dem .NET Framework

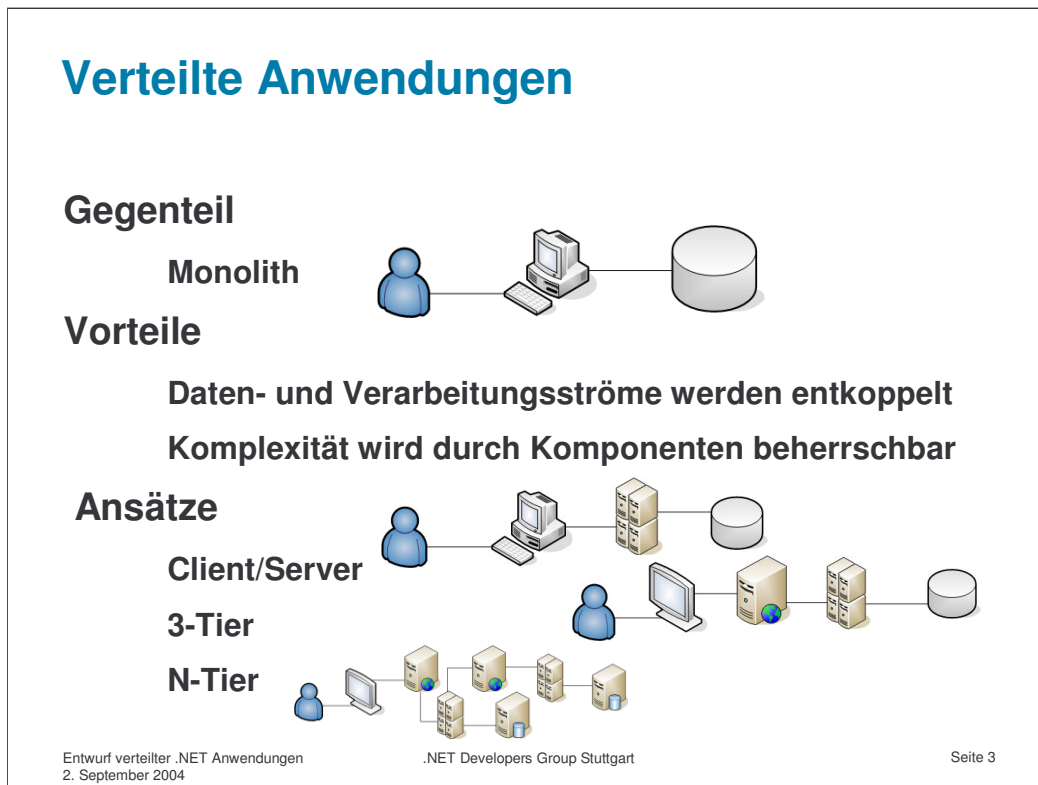
Entwurf verteilter .NET Anwendungen

.NET Developers Group Stuttgart, <http://www.devgroup-stuttgart.de/>
Henry Boehlert, 25.08.2004

Remoting umfasst eine Reihe von Technologien im .NET Framework für die Interprozesskommunikation und kann daher als Ersatz von DCOM in der Managed-Code -Welt angesehen werden. Die dabei zu beachtenden Aspekte sollen in diesem Vortrag beleuchtet werden.

Agenda

- **Verteilte Anwendungen**
- **Interprozesskommunikation und Marshaling**
- **Aktivierung und Lebensdauer von Objekten**
- **Konfiguration und Versionierung**
- **Erweiterbarkeit**



Verteilte Anwendungen sind dann sinnvoll, wenn die Komplexität der Daten- und Verarbeitungsströme ansteigt, bspw. durch die Anzahl der Benutzer oder Menge der zu verarbeitenden Daten.

Neben der Trennung der Programmkomponenten wird die Ausführung der einzelnen Aufgaben auf mehrere Prozesse verteilt, ggf. auch auf mehrere Rechner.

Die Aufgaben können von spezialisierten Anbietern erledigt werden, z.B. Web-Server- oder Datenbank-Hosting.

Das Grundprinzip ist die Trennung in Client- und Server-Prozesse. In mehrschichtigen Architekturen treten Server u.U. wieder als Clients weiterer Server auf.

Die Kommunikation wird allerdings komplexer, insbesondere, wenn Client und Server nicht füreinander geschaffen wurden.

Architektur I

Windows Distributed interNet Applications Architecture

Entwurf verteilter .NET Anwendungen
2. September 2004

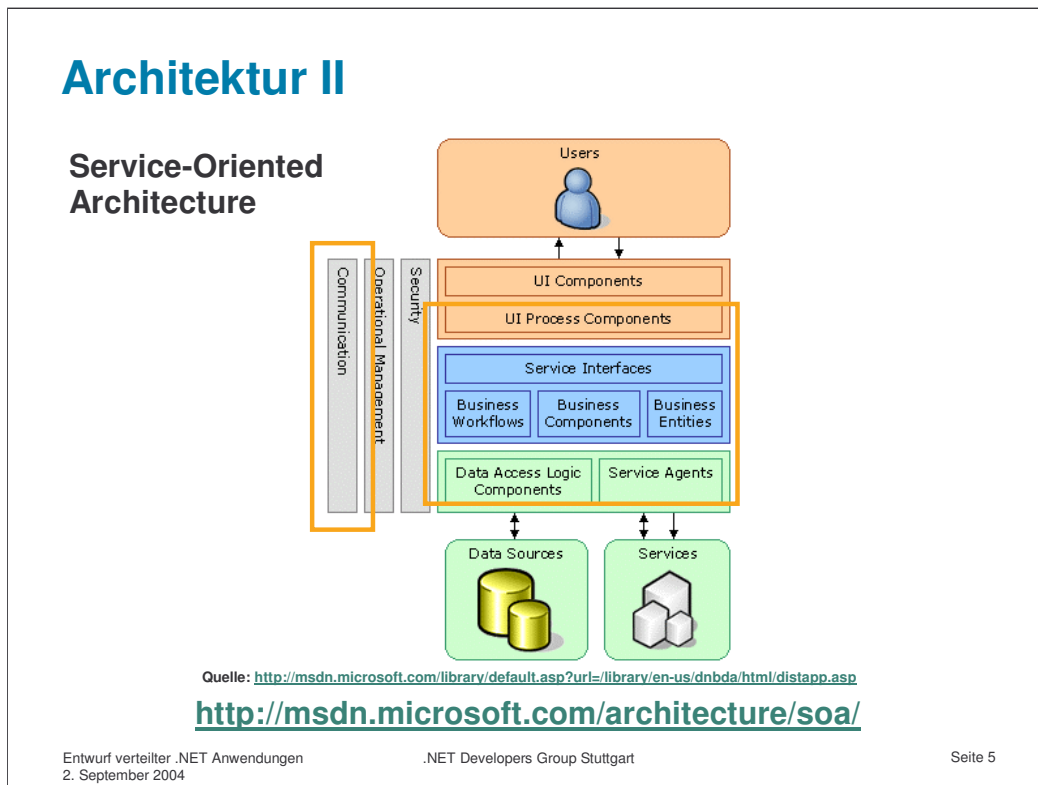
.NET Developers Group Stuttgart

Seite 4

Für Windows galt lange Zeit Windows DNA als Standardarchitektur. Dies ist im Wesentlichen eine Architektur mit einem Anwendungsserver, der auf einen Datenbankserver zurück greift.

Für Windows besteht diese Architektur aus IIS mit ASP-Webseiten für die Präsentation, COM-Objekten für die Geschäftslogik sowie ADO und Stored Procedures für den Zugriff auf die Daten in einem SQL Server. Auf der Server-Seite wird hauptsächlich über DCOM kommuniziert.

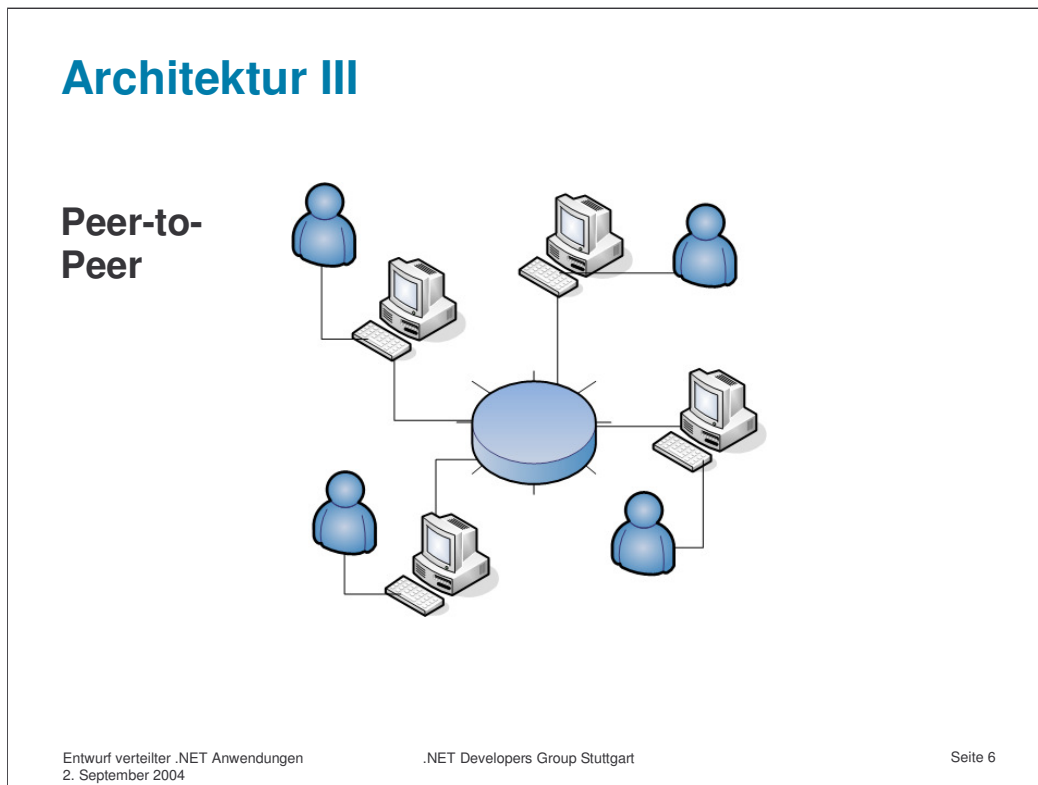
Auf MSDN ist die Dokumentation dieser Architektur nicht mehr verfügbar.



Zur Zeit stehen Dienst-orientierte Architekturen im Vordergrund, vor allem die Möglichkeit, Workflows und Aufgabenbereiche als einzelne Dienste abgrenzen und auslagern zu können.

Für die Kommunikation zwischen Benutzer und System, sowie für den Datenaustausch mit den untergeordneten Diensten werden Web-Protokolle eingesetzt, d.h. HTTP und Web Services. Damit sind die Systeme nicht auf eine bestimmte Plattform beschränkt.

Dieser Vortrag bezieht sich auf den Einsatz von .NET Remoting in solchen Architekturen. Da die Plattformbindung recht stark ist, dient es der Kommunikation der Komponenten innerhalb eines bestimmten Systems. Insofern kann es DCOM in einigen Bereichen ablösen.



Eine weitere Architekturalternative ist Peer-to-Peer, bei der jeder Beteiligte alle oder zumindest einen Teil der insgesamt angebotenen Dienste erbringen kann.

Die Kommunikation erfolgt hier meist über HTTP oder binäre Protokolle, der Einsatz von Remoting ist aber vorstellbar.

Interprozesskommunikation

Übertragung von Daten und Ereignissen zwischen Prozessen

http://msdn.microsoft.com/library/en-us/ipc/base/interprocess_communications.asp

Entwurf verteilter .NET Anwendungen .NET Developers Group Stuttgart Seite 7
2. September 2004

Die Speicherbereiche von Prozessen sind voneinander getrennt. Um trotzdem auf gemeinsamen Objekten arbeiten zu können, müssen die Prozesse Daten und Ereignisse austauschen.

File Mapping

Key Point: File mapping is an efficient way for two or more processes on the same computer to share data, but you must provide synchronization between the processes.

Data Copy

Key Point: Data copy can be used to quickly send information to another application using Windows messaging.

Clipboard

Key Point: All applications should support the clipboard for those data formats that they understand. For example, a text editor or word processor should at least be able to produce and accept clipboard data in pure text format.

Windows Sockets

Key Point: Windows Sockets is a protocol-independent interface capable of supporting current and emerging networking capabilities.

Dynamic Data Exchange

Key Point: DDE is not as efficient as newer technologies. However, you can still use DDE if other IPC mechanisms are not suitable or if you must interface with an existing application that only supports DDE.

Mailslots

Key Point: Mailslots offer an easy way for applications to send and receive short messages (~400 bytes). They also provide the ability to broadcast short messages across all computers in a network domain.

Pipes

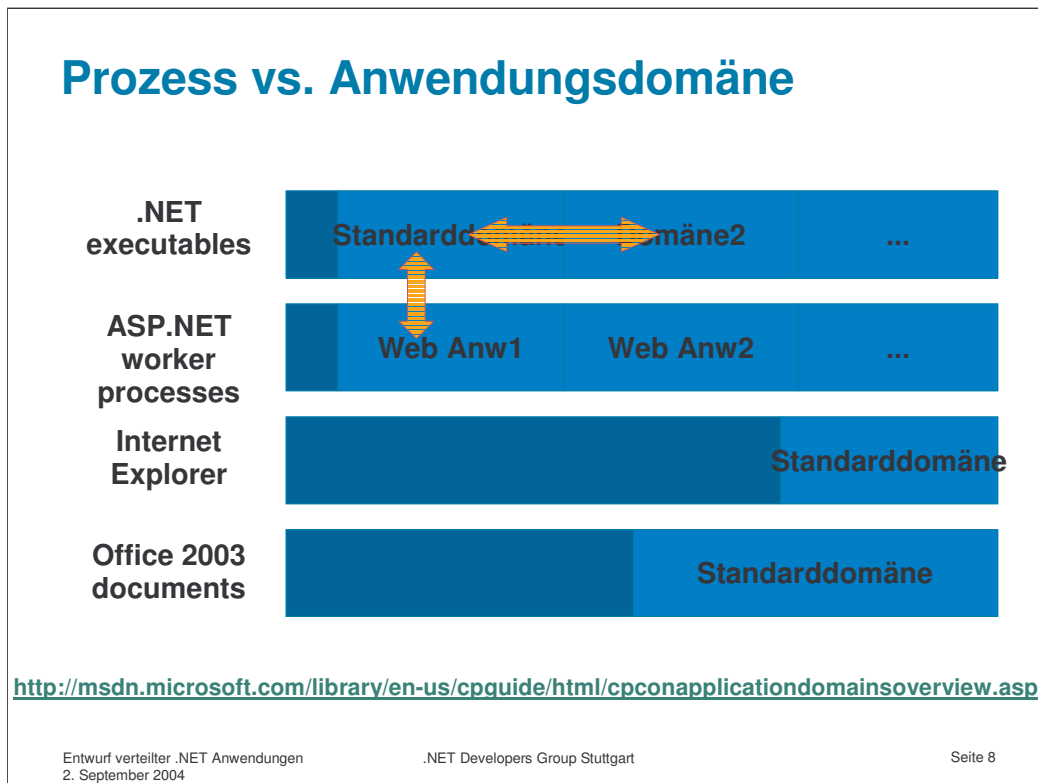
Key Point: Anonymous pipes provide an efficient way to redirect standard input or output to child processes on the same computer. Named pipes provide a simple programming interface for transferring data between two processes, whether they reside on the same computer or over a network.

RPC

Key Point: RPC is a function-level interface, with support for automatic data conversion and for communications with other operating systems. Using RPC, you can create high-performance, tightly coupled distributed applications.

COM and ActiveX Object Services, DCOM

Key Point: OLE supports compound documents and enables an application to include embedded or linked data that, when chosen, automatically starts another application for data editing. This enables the application to be extended by any other application that uses OLE. COM objects provide access to an object's data through one or more sets of related functions, known as interfaces.



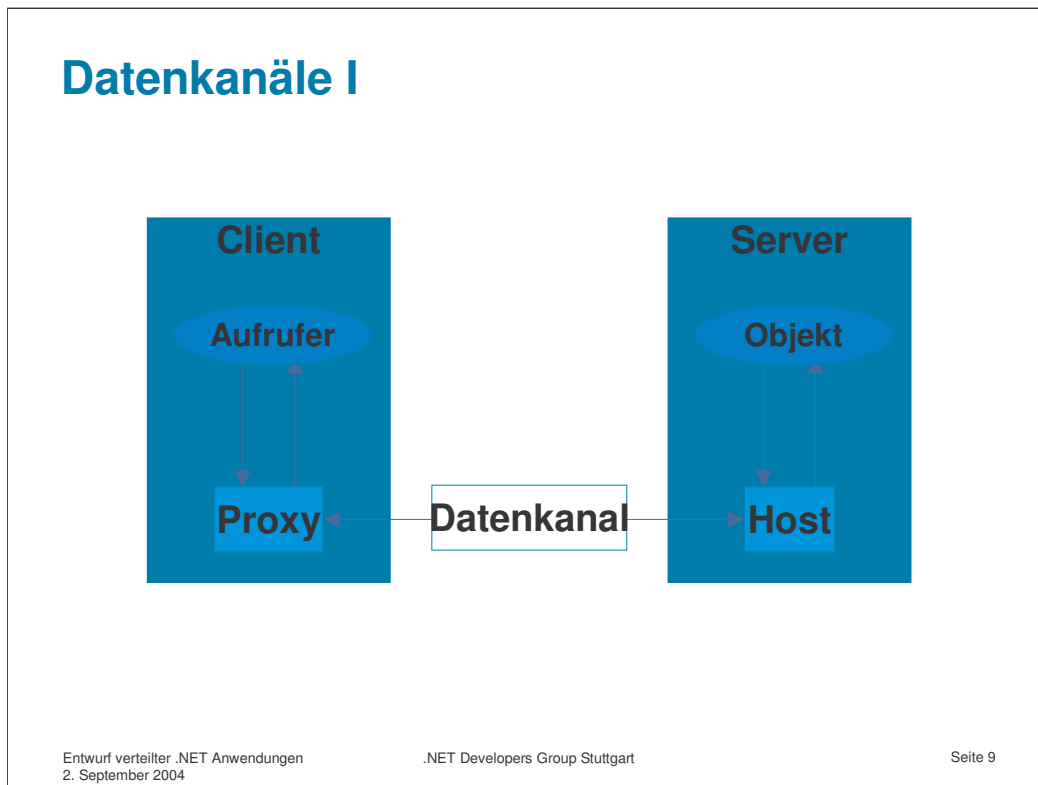
Jede Win32-Anwendung kann die .NET Common Language Runtime laden (CorBindToRuntimeEx):

- <http://msdn.microsoft.com/library/en-us/cpgenrefer/html/grfunCorBindToRuntimeEx.asp>

Die bekanntesten Hosts sind ausführbare .NET-Programme und ASP.NET-Worker-Prozesse. Aber auch im Internet Explorer können auch .NET-Programme ausgeführt werden. Mit Office 2003 wurde die Möglichkeit geschaffen, die Funktionalität von Word- und Excel-Dokumente mit .NET-Code zu erweitern.

Die .NET Laufzeitumgebung erzeugt stets eine Anwendungsdomäne (application domain), die Standarddomäne (default domain). Anwendungsdomänen sind abgeschlossene Ausführungsumgebungen für .NET Programme. Mit Anwendungsdomänen können mehrere .NET Programme in einem Prozess ausgeführt werden. Das ist möglich, weil die Common Language Runtime den Speicher verwaltet und die Programme daher effektiv trennen kann.

Um die Speichertrennung zu überwinden und auf gemeinsamen Objekten arbeiten zu können, müssen zwischen Anwendungsdomänen Daten und Ereignisse ausgetauscht werden. Dafür wird .NET Remoting verwendet.



Für die Kommunikation zwischen der Anwendungsdomäne des Clients und des Servers wird auf dem Client i.d.R. ein sogenannter Proxy, ein Stellvertreter, erzeugt, der die Aufrufe entgegen nimmt und an das eigentliche Objekt weiterleitet.

Der Proxy verwendet einen Datenkanal (Channel), um mit seiner Gegenstelle, dem Host (z.T. auch als Stub bezeichnet), zu kommunizieren. Der Host leitet die Aufrufe an das Server-Objekt weiter, nimmt die Ergebnisse entgegen und überträgt sie zurück zum Proxy, der sie an den Aufrufer weiter gibt, so als ob der Aufruf lokal erfolgt wäre.

Standardmäßig besteht der Kanal aus einer TCP- oder einer HTTP-Verbindung. Mit .NET 2.0 kommt auch ein IPC-Kanal dazu, der Pipes verwendet und nur lokal auf einer Maschine verwendet werden kann. Die Vorteile gegenüber dem TCP-Kanal liegen auch im höheren Durchsatz, aber vor allem in der höheren Sicherheit, da kein TCP-Port geöffnet werden muss.

Marshaling

Main Entry: ²marshal

Variant(s): *also* marshall

Function: *verb*

Etymology: Middle English, from Old French *mareschal*, of Germanic origin; akin to Old High German *marahscalc* marshal, from *marah* horse + *scalc* servant

Inflected Form(s): -shaled *or* -shalled; -shal-ing *or* mar-shal-ling

/'märsh-(ə-)li[ng]/

transitive senses

1 : to place in proper rank or position <*marshaling* the troops>

2 : to bring together and order in an appropriate or effective way <marshal arguments>

3 : to lead ceremoniously or solicitously : USHER <*marshaling* her little group of children down the street>

intransitive senses : to take form or order <ideas *marshaling* neatly>

<http://webster.com/cgi-bin/dictionary?book=Dictionary&va=marshal>

Für Übertragung von Objekte über eine Datenkanal sind verschiedene Operationen notwendig, die die Daten dieser Objekte sammeln und in eine für den Datenkanal geeignete Datenstruktur überführen. Diese Operationen werden als Marshaling bezeichnet.

Häufig werden Objekte durch Serialisierung auf der Senderseite, Übertragung und Deserialisierung auf der Empfängerseite ge-marshaled. Die jeweilige Technologie hängt vom Übertragungskanal und der Objektimplementierung ab.

Ähnlich wie bei der Parameterübergabe an eine Funktion gibt es zwei Möglichkeiten, Daten zu übergeben: als Wert oder als Referenz.

Marshaling by reference

```
public abstract class Brush : MarshalByRefObject,
ICloneable, IDisposable
{
    internal Brush();
    public abstract object Clone();
    public void Dispose();
    protected virtual void Dispose(bool disposing);
    ~Brush();
    internal void SetNativeBrush(IntPtr nativeBrush);
    internal IntPtr nativeBrush;
}
```

Entwurf verteilter .NET Anwendungen
2. September 2004

.NET Developers Group Stuttgart

Seite 11

MarshalByRef Objekte werden über Referenzen angesprochen. Sie entsprechen Zeigern, Referenz- oder Rückgabeparametern, z.B.

```
public static bool TryParse(string s, NumberStyles style,
IFormatProvider provider, out double result)
```

Für die Übertragung der Referenzen wird die spezielle Klasse ObjRef verwendet.

Da die Methoden nicht auf der Client-Seite ausgeführt werden, braucht der Client keinen Zugriff auf das Server-Assembly. Eine Beschreibung der Methoden durch ein Interface, durch eine abstrakte Basisklasse oder ein Skeleton-Assembly ist ausreichend.

Skeleton-Assemblies können mit dem Tool „soapsuds“ aus dem .NET Framework SDK erzeugt werden:

```
soapsuds -ia:Brush -gc
```

Ähnlich wie die Proxy/Stub-DLL von COM, enthalten Skeleton-Assemblies lediglich Meta-Daten.

Marshaling by value

```
[Serializable]
public struct Point {
    static Point();
    public Point(int x, int y);
    public void Offset(int dx, int dy);

    public bool IsEmpty { get; }
    public int X { get; set; }
    public int Y { get; set; }

    public static readonly Point Empty;
    private int x;
    private int y;
}
```

Entwurf verteilter .NET Anwendungen
2. September 2004

.NET Developers Group Stuttgart

Seite 12

Marshaling by value entspricht der Parameterübergabe „by value“, z.B. wie „y“ in $x = \text{Math.Sin}(y)$

„By value“ bedeutet, dass der Wert zur Übergabe kopiert wird. Daher wirken sich Änderungen, die auf der Empfängerseite vorgenommen werden, auch nicht auf die Senderseite aus.

.NET Remoting verwendet für Marshaling by value Serialisierung. Ein Formatter kopiert dazu die Felder des Objektes in einen Datenstrom, der vom Senderprozess zum Empfängerprozess übertragen wird.

Durch das Attribut „Serializable“ wird deklariert, dass Instanzen einer Klasse auf diese Weise serialisiert werden können.

Ein serialisierbares Objekt darf nur serialisierbare Felder oder Referenzen auf MarshalByRef-Objekte enthalten.

Methoden auf serialisierten Objekten werden auf der aufrufenden Seite ausgeführt. Daher muss das Assembly, das die Klasse enthält auf der Client-Seite verfügbar sein. Wenn die dadurch bewirkten Änderungen an Objekten auf der anderen Seite sichtbar sein sollen, ist Serialisierung nicht geeignet.

Es gibt viele Klassen, die z.B. Betriebssystem-Ressourcen oder Datenbankverbindungen beinhalten, die nicht ohne weiteres serialisiert werden können. Die jeweiligen Handles haben nur innerhalb eines Prozesses Gültigkeit. Andere Klassen könnten z.B. eine so große Menge Daten enthalten, dass die Serialisierung nicht in jedem Fall sinnvoll ist.

In diesen Fällen ist die Übergabe einer Referenz auf das betreffende Objekt erforderlich.

Serialisierung SOAP Formatierer

- Standard-Serialisierung für HTTP-Datenkanal
- SOAP 1.1 Encoding
- In NET 2.0 möglicherweise nicht mehr unterstützt zugunsten
 - Binären Formatierer oder Web Service verwenden

```
<SOAP-ENV:Envelope xmlns:si="http://www.w3.org/2001/XMLSchema-instance" xmlns:sd="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:cl="http://schemas.microsoft.com/soap/encoding/ clr/1.0" SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" >
  <SOAP-ENV:Body>
    <I2set_TopLeft id="ref-1" xmlns:i2="http://schemas.microsoft.com/clr/nsassem/Plane.Rectangle/Plane">
      <value xsi:type="a1:Point" xmlns:a1="http://schemas.microsoft.com/clr/nsassem/Plane/Plane%2C%20
        Version%3D1.0.1705.32698%2C%20Culture%3Dneutral%2C%20PublicKeyToken%3Dnull">
          <x>1</x>
          <y>2</y>
        </value>
      </I2set_TopLeft>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

Der HTTP-Datenkanal verwendet standardmäßig den SOAP-Formatierer. Im Konstruktor kann jedoch auch ein BinaryServerFormatterSinkProvider, resp. BinaryClientFormatterSinkProvider angegeben werden.

Serialisierung Binärer Formatierer

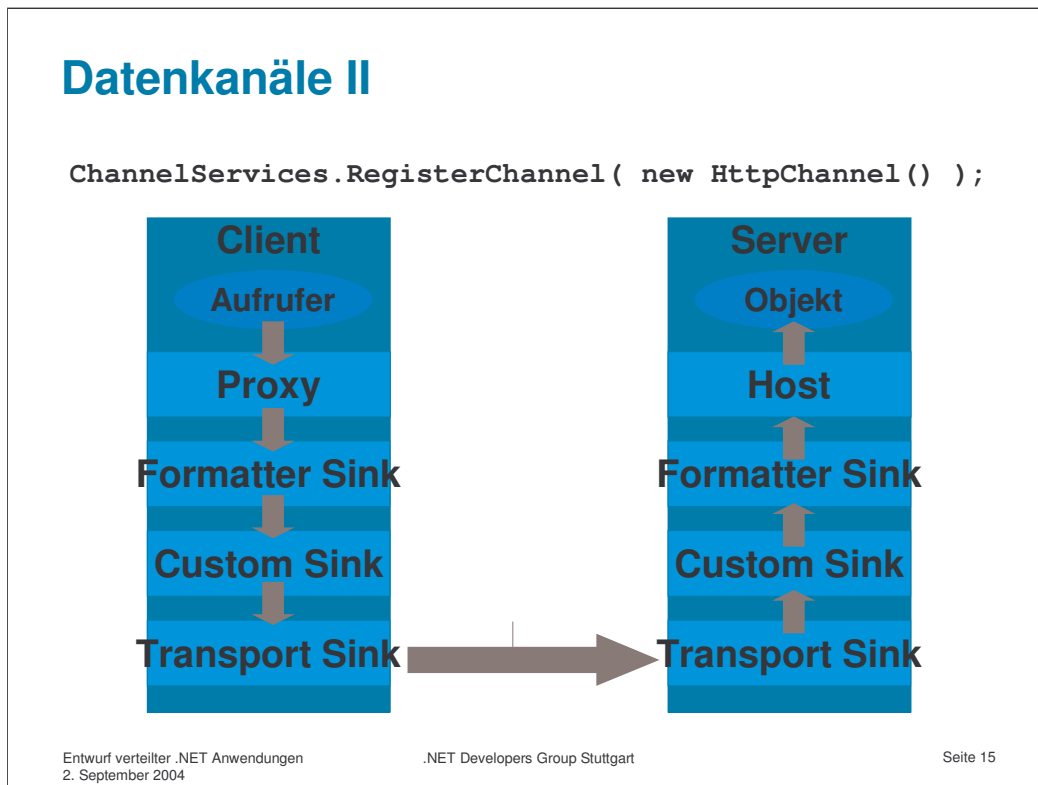
- Standard-Serialisierung für TCP-Datenkanal
- Hohe Übertragungsrate
- Proprietäres Format
 - Für Interoperabilität Web Service verwenden

```

00000000 2e 4e 45 54 01 00 00 00 00 00 77 00 00 00 04 00 .NET.... ..W....
00000010 01 01 1e 00 00 00 74 63 70 3a 2f 2f 78 70 73 65 .....:c p://xpse
00000020 72 76 65 72 31 3a 32 30 30 31 2f 52 65 63 74 61 rver:20 01/Recta
00000030 6e 67 6c 65 06 00 01 01 18 00 00 00 61 70 79 6c ngle... ..app
00000040 69 63 61 74 69 6f 6e 2f 6f 63 74 65 74 2d 73 74 ication/ octet-st
00000050 72 65 61 6d 00 00 00 00 00 01 00 00 00 00 00 00 ream..
00000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000070 00 15 11 00 00 00 12 08 54 6f 53 74 72 69 6e 67 ..... ToString
00000080 12 54 50 6c 61 6e 65 2e 52 65 63 74 61 6e 67 6c .TPlane. Rectangl
00000090 65 2c 20 50 6c 61 6e 65 2c 20 56 65 72 73 69 6f 6e, Plane Versio
000000a0 6e 3d 31 2e 30 2e 31 37 30 35 2e 33 32 36 39 38 n=1.0.17 05:32698
000000b0 2c 20 43 75 6c 74 75 72 65 3d 6e 65 75 74 72 61 , Cultur e=neutra
000000c0 6c 2c 20 50 75 62 6c 69 63 4b 65 79 54 6f 6b 65 1, Publi cKeyToke
000000d0 6e 3d 6e 75 6c 6c 0b n=null.

```

Der TCP-Datenkanal verwendet standardmäßig den binären Formatierer. Im Konstruktor kann jedoch auch ein SoapServerFormatterSinkProvider, resp. SoapClientFormatterSinkProvider angegeben werden.



Die Datenübertragung im Kanal erfolgt über mehrere Schritte, sogenannte Sinks, über die jeder Aufruf und jeder Rückgabewert als Nachricht (Message) verschickt werden.

Kanäle können durch eigene Sinks erweitert werden.

Aktivierung I

- **Instanziierung von Server-Objekten**
 - **Client-aktiviert**
 - **Server-aktiviert**
 - **Singleton**
 - **SingleCall**
 - **Dynamische Veröffentlichung**

MarshalByRef-Objekte laufen auf dem Server. Daher ist es wichtig zu klären, wann und wie sie instanziiert werden.

Client-Aktivierung bedeutet, dass jede Instanziierung eines Remote-Objektes auf dem Client durch die Instanziierung eines Objektes auf dem Server abgebildet wird und der Client die Referenz auf das Server-Objekt erhält.

Server-Aktivierung bedeutet, dass der Server entscheidet, wann er eine Instanz erzeugt, die die Anfrage der Clients beantwortet. .NET Remoting kennt hierfür zwei Modi, Singleton, in dem nur eine einzige Instanz des Remote-Objektes erzeugt wird und SingleCall, in dem jeder Aufruf von einer neuen Instanz beantwortet wird.

Darüber hinaus gibt es die Möglichkeit, ein Objekt direkt zu veröffentlichen.

Client-aktivierte Objekte

Server:

```
ChannelServices.RegisterChannel( new HttpChannel( 2001 ) );  
RemotingConfiguration.ApplicationName = "Rectangle";  
RemotingConfiguration.RegisterActivatedServiceType(  
    typeof( Rectangle ) );
```

Client:

```
ChannelServices.RegisterChannel( new HttpChannel() );  
RemotingConfiguration.RegisterActivatedClientType(  
    typeof( Rectangle ),  
    "http://xpserver1:2001/Rectangle" );  
Rectangle r = new Rectangle();
```

Client-aktivierte Objekte werden bei jeder Instanziierung durch eine Client angelegt. Das Remote-Objekt kann sich daher spezifische Daten merken, ohne dass Zugriffe anderer Clients synchronisiert werden müssen.

Voraussetzung für die Verwendung client-aktiverter Objekte ist die Registrierung des Typs als „Activated Service“ auf beiden Seiten.

Der Client muss die Portnummer und den Application Name des Remote-Objektes kennen.

Auf beiden Seiten muss ein Assembly vorhanden sein, das eine Implementierung des entsprechenden Typs beinhaltet. Im Falle von MarshalByRef-Objekten reicht ein Assembly, dass mit SoapSuds erstellt wurde.

Server-aktivierte Objekte

Client:

```
RemotingConfiguration.RegisterWellKnownClientType(  
    typeof( Rectangle ),  
    "http://xpserver1:2001/Rectangle" );  
  
Rectangle r = new Rectangle();  
  
// oder  
  
Rectangle r = (Rectangle)Activator.GetObject(  
    typeof( Rectangle ),  
    "http://xpserver1:2001/Rectangle" );
```

Server-aktivierte Objekte werden als „Well Known Service“ bezeichnet, da der Client die komplette URL des Remote-Objektes kennen muss.

Da das Remote-Objekt auf dem Server instanziiert wird, kann es nur ein Default-Konstruktor verwendet werden.

Server-aktivierte Objekte Singleton

Server:

```
ChannelServices.RegisterChannel( new HttpChannel( 2001 ) );  
RemotingConfiguration.RegisterWellKnownServiceType(  
    typeof( Rectangle ), "Rectangle",  
    WellKnownObjectMode.Singleton );
```

Die .NET Remoting Laufzeitumgebung erzeugt von Singleton-Remote-Objekten nur eine einzige Instanz auf dem Server, die an alle Clients ausgegeben wird.

Das Remote-Objekt muss sich selbst um die Synchronisation der Zugriffe kümmern.

Diese Vorgehensweise entspricht in etwa dem Multi-Threaded-Apartment bei COM.

Server-aktivierte Objekte Single Call

Server:

```
ChannelServices.RegisterChannel( new HttpChannel( 2001 ) );  
RemotingConfiguration.RegisterWellKnownServiceType(  
    typeof( Rectangle ), "Rectangle",  
    WellKnownObjectMode.SingleCall );
```

SingleCall-Objekte werden von der .NET Laufzeitumgebung für jede Client-Anforderung neu instanziiert.

In diesem Modus kann das Remote-Objekt nicht ohne zusätzliche Maßnahmen spezifische Daten speichern.

SingleCall kann aber sehr gut bei Lastverteilung auf mehrere Server eingesetzt werden.

Server-aktivierte Objekte Dynamische Veröffentlichung

Server:

```
ChannelServices.RegisterChannel( new HttpChannel( 2001 ) );  
RemotingServices.Marshal( new Rectangle( new Point( 10, 11  
) , new Point( 12, 13 ) ), "Rectangle" );
```

Server-aktivierte Objekte müssen zwangsläufig über den Standard-Konstruktor instanziiert werden. Muss ein anderer Konstruktor eingesetzt werden, weil z.B. dynamische Konfigurationsdaten an das Objekt übergeben werden müssen, kann das Objekt direkt veröffentlicht werden.

Die URL des Objektes kann mit `RemotingServices.GetObjectUri` ermittelt werden.

Aktivierung II

- **Shared Assemblies**
 - **Server-Assembly**
 - **einzige Möglichkeit für Marshaling by value**
 - **Abstrakte Basisklasse**
 - **nur SOA**
 - **Interface**
 - **nur SOA**
 - **Skeleton (Meta-Daten-Assembly)**
 - **nur Default-Konstruktor möglich**
- **Factory**
 - **zusammen Interfaces oder abstrakten Basisklassen**
 - **nur MarshalByRef**

Entwurf verteilter .NET Anwendungen
2. September 2004

.NET Developers Group Stuttgart

Seite 22

Um Server-Objekte vom Client aus aktivieren und benutzen zu können, muss der Client über Assemblies verfügen, die Meta-Daten wie Typ-Namen und Methodensignaturen liefern.

Marshal-By-Value-Objekte können nur verwendet werden, wenn das entsprechende Assembly auf dem Client vorhanden ist.

Für andere Objekte wird man diese Variante eher vermeiden, denn sonst muss der Server-Code an den Client ausgeliefert werden, was zu rechtlichen und technischen Schwierigkeiten führen kann (z.B. Re-Engineering oder Versionskonflikte).

Im Prinzip sind abstrakte Basisklassen oder Interfaces ausreichend, allerdings nur für Server-aktivierte Objekte, da abstrakte Klassen und Interfaces nicht direkt instanziiert werden können.

Die Lösung kann das Fabrik-Entwurfsmuster (Factory) liefern durch ein Server-aktiviertes Fabrik-Objekt, das wiederum Objekte vom gewünschten Typ auf dem Server erzeugen und an den Client ausliefern kann.

Lebensdauer

- **.NET Remoting verwendet Laufzeit-Lease**
 - **Eigenschaften**
 - Remote-Objekte werden für eine bestimmte Laufzeit erzeugt
 - Wird innerhalb der Restlaufzeit nicht zugegriffen, verfällt das Objekt
 - Konfigurierbar über MarshalByRefObject und Sponsoren
 - **Vorteile gegenüber Ping (DCOM)**
 - Kein zusätzlicher Netzwerkverkehr
 - Unabhängig von Firewall etc.

Neben der Art und Weise der Aktivierung eines Objektes muss auch festgelegt werden, wie lange die Objekt-Instanz auf dem Server aktiviert bleibt.

Ausnahme sind Server-aktivierte SingleCall-Objekte, die nur während für die Ausführung eines Methodenaufrufs aktiviert bleiben.

Bei DCOM und anderen Technologien wird hierfür ein Ping verwendet, der regelmäßig zwischen Client und Server verschickt wird. Bleibt der Client-Ping aus, wird das Server-Objekt abgebaut. Das verbraucht Netzwerkressourcen, wie auch das Offenhalten einer TCP-Verbindung.

Lease

- **Konfiguration der Laufzeit**
 - `ILease.InitialLeaseTime`
 - **Standard: 5min**
 - `ILease.RenewOnCallTime`
 - **Standard: 2min**
- **Konfiguration des Lease Managers für die Anwendungsdomäne**
 - **Klasse `LifetimeServices`**
 - `LifetimeServices.LeaseManagerPollTime`
 - **Standard: 10sec**

Eine Lease ist Laufzeit, die dem Objekt verliehen wird. Initial sind das standardmäßig 5 Minuten. Jeder Methodenaufruf setzt die Laufzeit standardmäßig auf mindestens 2 Minuten.

Der LeaseManager des Servers überprüft in einem Intervall die verbleibende Laufzeit aller Remote-Objekte und gibt abgelaufene Objekte zur Garbage Collection frei. Clients, die nach Ablauf der Laufzeit auf ein Server-Objekt zugreifen, erhalten eine Fehlermeldung.

Sponsor

- **Sponsor kann Laufzeit von Objekten verlängern**
 - Aufruf von `Renew(ILease lease)` durch den Lease Manager
 - Client-side Sponsor entspricht DCOM-Ping
 - Client muss Portnummer für Callback konfigurieren
 - Server-side Sponsor
 - Keep-Alive-Mechanismus
- **Konfiguration**
 - `SponsorshipTimeout` gibt Zeit vor, in der der Sponsor antworten muss

Das Lease-Konzept sieht vor, dass sogenannte Sponsor-Objekte die Laufzeit eines Servers kontrollieren können. Diese Objekte werden benachrichtigt, wenn der Ablauf der Laufzeit des Server-Objektes kurz bevorsteht und erhalten die Möglichkeit, eine neue Restlaufzeit vorzugeben.

Sponsoren können von der Client- und von der Server-Seite aus registriert werden.

Konfiguration Server

- `RemotingConfiguration.Configure(string filename)`

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.runtime.remoting>
    <application name="Rectangle">
      <lifetime leaseTime="10000MS"
        renewOnCallTime="2S"
        leaseManagerPollTime="10" />
      <channels>
        <channel ref="http" port="2001" />
      </channels>
      <service>
        <wellknown mode="Singleton"
          type="LeasedObject.LeaseObject, LeasedObject"
          objectUri="LeasedObject" />
        <activated
          type="Plane.Rectangle, Plane" />
      </service>
    </application>
  </system.runtime.remoting>
</configuration>
```

Die Konfiguration von Remoting kann durch Konfigurationsdateien erfolgen. Anders als die Anwendungskonfigurationsdatei wird jedoch die Remoting-Konfiguration nicht automatisch geladen, sondern muss explizit aufgerufen werden.

Konfiguration Client

- `RemotingConfiguration.Configure(string filename)`

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.runtime.remoting>
    <application name="Rectangle">
      <channels>
        <channel ref="http" port="0" />
      </channels>
      <client url="http://xpserver1:2001/Rectangle">
        <wellknown mode="Singleton"
          type="LeasedObject.LleasedObject, LeasedObject"
          url="http://xpserver1:2001/LeasedObject" />
        <activated type="Plane.Rectangle, Plane" />
      </client>
    </application>
  </system.runtime.remoting>
</configuration>
```

Versionierung

- **Strong-Naming**

- `sn -k Demo.snk`
- `[assembly: AssemblyKeyFile("../\\..\\..\\Demo.snk")]`

- **Voll qualifizierter Assembly-Name**

- `Version=1.0.1698.18644, Culture=neutral, PublicKeyToken=7de91de30d1c8ac6`

- **Versionen im Global Assembly Cache ablegen**

- **“gacutil /i VersionedObject.dll” als PostBuild-Event**
- **“gacutil /I VersionedObject” alle VersionedObject anzeigen**

Versionierung von Assemblies erfolgt durch das `AssemblyVersion`-Attribut. Damit Name und Version eines Assemblys nicht nachträglich unbemerkt von Dritten geändert werden können, kann man Assemblies mit einem Strong Name Key signieren. Mit Hilfe des Public Key Token prüft die .NET-Laufzeitumgebung, ob das Assembly unverändert ist.

Im Global Assembly Cache (`C:\WINDOWS\assembly`) können mehrere Versionen eines Assemblys nebeneinander (side-by-side) abgelegt werden. Zur Verwaltung gibt es im .NET SDK das Programm `gacutil`.

Versionierung Server-aktivierte Objekte

- Konfiguration

- Verschiedene Versionen – Verschiedene URL

```
<service>
  <wellknown mode="Singleton"
  type="VersionedObject.VersionedObject, VersionedObject,
  Version=1.0.1698.18644, Culture=neutral,
  PublicKeyToken=7de91de30d1c8ac6"
  objectUri="VersionedObject" />
  <wellknown mode="Singleton"
  type="VersionedObject.VersionedObject, VersionedObject,
  Version=2.0.1698.18857, Culture=neutral,
  PublicKeyToken=7de91de30d1c8ac6"
  objectUri="VersionedObject2" />
</service>
```

Server-aktivierte Objekte können unter einer URL nur eine Version veröffentlichen.

Nur die aktuelle Version kann verwendet werden, wenn der Server das Assembly direkt referenziert (im Visual-Studio-Projekt)!

Versionierung Client-aktivierte Objekte

- Konfiguration
 - Client bestimmt Version
 - Server kann Binding Redirect verwenden

```
<configuration>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name="VersionedObject"
          publicKeyToken="7de91de30d1c8ac6"
          culture="neutral" />
        <bindingRedirect oldVersion="1.0.1698.19096"
          newVersion="2.0.1698.19731" />
      </dependentAssembly>
    </assemblyBinding>
  </runtime>
</configuration>
```

Entwurf verteilter .NET Anwendungen
2. September 2004

.NET Developers Group Stuttgart

Seite 30

Client-aktivierte Objekte werden in der dem Client bekannten Version angefordert.

Der Server muss das entsprechende Assembly zur Verfügung haben, am besten mit Hilfe des Global Assembly Cache.

Mit dem Konfigurationselement `<assemblyBinding>` können auf dem Server aber Versionen auch umgeleitet werden.

Versionierung Marshal By Value/Serializable

- **Implementierung von ISerializable**
 - **Constructor (SerializationInfo info, StreamingContext context)**
 - **void GetData(SerializationInfo info, StreamingContext context)**

Serialisierbare Klassen lassen sich nur schwer mit unterschiedlichen Versionen einsetzen.

Soll dennoch ein alter Client auf einen neuen Server zugreifen, muss mit ISerializable eine spezielle Serialisierung implementiert werden, die entsprechend tolerant ist.

Sicherheit

• Fehlende Sicherheitsmechanismen in .NET Remoting

- Authentifizierung
- Authorisierung
- Verschlüsselung
- Hosting in IIS
 - SSL
- Implementierung eines eigenen Channels

In den .NET-Versionen 1.0 und 1.1 bieten HTTP-Datenkanal und TCP-Datenkanal keine Authentifizierungsmechanismen.

Das kann über spezielle Channel-Sinks nachgerüstet werden (google: IdentitySink und „SSPI Channel Sink“)

Eine der möglichen Hosts für .NET Remoting ist IIS. Dazu wird ein virtuelles Verzeichnis angelegt und darin eine Konfigurationsdatei web.config und ein Unterverzeichnis „bin“, in das alle die Assemblies kopiert werden, die nicht in den Global Assembly Cache installiert wurden.

Die web.config enthält einen Teil für Remoting wie bereits gezeigt. Wichtig ist, dass die URIs der veröffentlichten Objekte auf „rem“ oder „soap“ enden. Dann macht die ASP.NET-IIS-Extension den Rest von selbst.

Die web.config-Datei und die Assemblies können stets geändert werden, ohne den IIS neu zu starten. Die Änderungen werden sofort nach dem Speichern wirksam.

Für das Remoting gelten dann die Sicherheitseinstellungen des virtuellen Verzeichnisses. Auch SSL kann verwendet werden. Allerdings kann nur der HTTP-Datenkanal verwendet werden. Um den Netzwerkdurchsatz zu verbessern kann allerdings binäre Formatierung werden.

Links

- <http://www.microsoft.com/germany/msdn/>
- Ingo Rammer: „Advanced .NET Remoting“, ISBN 1-59059-025-2
http://www.thinktecture.com/Resources/books/default.html#ADR_CSHARP
- Ingo Rammer: „Advanced .NET Remoting in VB.NET“, ISBN 1-59059-062-7
http://www.thinktecture.com/Resources/books/default.html#ADR_VBNET
- <http://msdn.microsoft.com/msdnmag/issues/02/10/NETRemoting/>
- <http://msdn.microsoft.com/library/en-us/dndotnet/html/hawkremoting.asp>

- <http://msdn.microsoft.com/msdntv/archive.aspx>

Indigo

- <http://msdn.microsoft.com/Longhorn/understanding/pillars/Indigo/default.aspx>

- <http://channel9.msdn.com/>

Entwurf verteilter .NET Anwendungen .NET Developers Group Stuttgart Seite 34
2. September 2004

Indigo ist die Weiterentwicklung von Web Services und Remoting für das .NET Framework von Windows „Longhorn“, das 2006 erscheinen soll.

Indigo ist insbesondere auf Service-orientierte Architekturen ausgerichtet und wird alle neuen Web-Service-Standards (WS-*) unterstützen.