Konzepte von Betriebssystem-Komponenten Middleware: .NET Remoting

André Frimberger

16.11.2004

1 Was ist .NET Remoting?

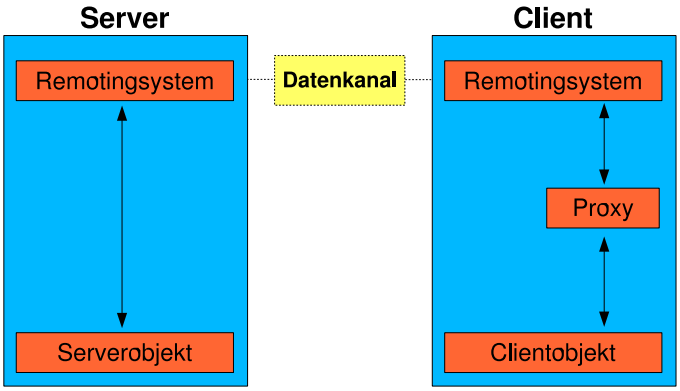
.NET Remoting ist ein Framework aus .NET, welches die Interprozesskommunikation zwischen zwei Prozessen als abstraktes Objekt bereitstellt. Ein Framework stellt eine Sammlung von Klassen dar, die einen Problembereich abdecken und zugleich allgemeingültiger als normale Klassen gehalten sind. Es ist völlig egal, ob zwei Anwendungen über lokale- oder über Remoteverbindungen kommunizieren, da von *.NET Remoting* die komplette Prozesskommunikation übernommen wird. Die Abstraktion geht sogar soweit, dass das verwendete Übertragungsprotokoll der Anwendung geändert werden kann, ohne die Applikation neu kompilieren zu müssen.

2 .NET Remoting

2.1 Prozess vs. Anwendungsdomäne

Jede Anwendung wird in .NET in einem eigenen, abgeschlossenen Adressraum ausgeführt. Diesen Adressbereich bezeichnet man im .NET Jargon als Anwendungsdomäne (Application Domain). Innerhalb so einer Domäne können zwar Prozesse von verschiedenen .NET Programmen ausgeführt werden, der Zugriff auf Objekte fremder Prozesse ist jedoch nur mit Hilfe des *.NET Remoting* Frameworks möglich.

2.2 Das Prinzip der verteilten Anwendung in .NET



Das remoting Framework kapselt den entfernten Objektaufruf so, dass der Client nicht mitbekommt, ob die Anwendung lokal oder remote ausgeführt wird. Um dies zu erreichen wird zwischen Remotingsystem (siehe Abb.) und Clientobjekt ein virtueller Proxy geschaltet, der die Transparenz gegenüber dem Programmierer sichert. Die Remotingsysteme selbst kommunizieren über einen Datenkanal (Data Channel), der weitestgehend protokollunabhängig ist. Es werden hauptsächlich zwei Datenübertragungsmethoden zur Inter-Process-Communication angeboten:

- binäre Datenübertragung per TCP
- SOAP über HTTP

Die binäre Übermittlung ist um Faktor 4 [1] schneller als SOAP, da bei diesem XML basiertem Datenaustauschprotokoll der Datenoverhead sehr viel größer ist, wie man am folgenden Beispiel gut sehen kann:

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        <m:GetLastTradePrice xmlns:m="Some-URI">
            <symbol>MyParameter<symbol>
        </m:GetLastTradePrice>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Für die Übermittlung eines elf Byte Paramters werden hier ca. 300 Byte übermittelt. SOAP bietet allerdings den Vorteil, auch in Netzwerken arbeiten zu können, die durch einen Proxy getrennt sind, der nur HTTP nach Außen durchlässt.

2.3 Parameterübergabe

2.3.1 Marshaling by Reference

Bei Marshaling by Reference werden die Objekte per Referenz angesprochen. Das Objekt wird auf dem Server instanziiert, der Client bekommt lediglich eine Refernz zurückgeliefert. Für die Übertragung ist die spezielle Klasse ObjRef zuständig. Da die Methoden auf dem Server ausgeführt werden, braucht der Client nur gegen eine Beschreibung der verwendeten Methoden (dies geht z.B. mit #soapsuds -ia:<classname>-gc) gelinkt zu werden.

2.3.2 Marshaling by Value

Bei Marshaling by Value werden die zu übertragenden Werte kopiert, sodass die Änderungen die Senderseite nicht berühren. Bei diesem Verfahren muss die Klasse das Attribut "*serializable*" tragen, d.h. die Felder des Objektes können in einen Datenstrom gepackt werden und vom Sender zum Empfänger übertragen werden. Dort werden sie wieder deserialisiert. Kriterien für ein serialisierbares Objekt: Das Objekt muss serialisierbare Felder oder Referenzen auf MarshalByRef Objekte enthalten. Außerdem muss die komplett implementierte Klasse auf dem Client verfügbar sein, da das serialisierte Objekt auf dem Client ausgeführt wird.

2.4 Instanziierung von Server Objekten

2.4.1 Client Aktivierung

Bei dieser Instanzierungsart wird ein "**new**"-Aufruf, bei der remote Objektinstanziierung auf dem Client, an den Server gesendet. Der Server instanziiert das Objekt und sendet eine Referenz auf dieses zurück. Der Client steuert also, welches Objekt (in welcher Version) erzeugt werden soll.

2.4.2 Server Aktivierung

Im Gegensatz zur Client Aktivierung bestimmt hier der Server, welche Version eines Objekts verwendet wird. Desweiteren unterstützt .NET zwei weitere Modi Server aktivierte Instanzen von Remote Objekten abzuarbeiten:

- SingleCall
- Singleton

Der SingleCall Modus fertigt eine (und nur eine) ankommende Anfrage ab. Diese Methode wird oft in Anwendungen eingesetzt, bei denen die Objekte nur wenig Arbeit leisten müssen. Außerdem sind Single Calls Zustandslos ("stateless"), d.h. es können keine Zustandsinformationen zwischen Methodenaufrufen gespeichert werden. Das Gegenstück stellen die Singleton Objects dar, diese können Informationen zwischen verschiedenen Client Anfragen speichern. Darüber hinaus können Singletons mehrere Clients bedienen und sind deswegen in Anwendungsszenarien sinnvoll, bei denen Daten explizit zwischen Clients ausgetauscht werden müssen.

2.5 Assemblies

Assemblies sind selbstbeschreibende Bibliotheken oder Programme, die eine Erweiterung von herkömmlichen .exe und .dll Dateien darstellen. Die genaue Definition ist unter [2] zu finden. Im *.NET Remoting* Kontext reicht es, sich ein Assembly als kompiliertes und gekapseltes Objekt vorzustellen, welches die komplette Implementierung oder nur einen Definitionsrumpf (ähnlich Interfaces) eines Serverobjektes darstellt.
Zur Erzeugung Server-aktivierter Objekte reicht ein einfacher Definitionsrumpf der verwendeten Remoteobjekte aus. Marshal-By-Value Objekte hingegen müssen die komplette Implementierung auf dem Client vorliegen haben, was unter Umständen zu Problemen, wie Reengeneering oder Versionskonflikten führen kann.
Abhilfe schafft ein sogenanntes Fabrik Entwurfsmuster (Factory Pattern), welches -je nach Kontext- spezialisierte Klassen mit gemeinsamen Schnittstellen bereitstellt, ohne dass der Client diese spezielle Klasse kennen muss. Muster selbst stellen eine weitere Abstraktionsebene gegenüber Klassen dar und können so Expertenwissen abstrakt beschreiben. Nähre Informationen unter [5]

2.6 Lebensdauer von Objekten

Die Lebensdauer eines Objektes wird bei der Aktivierung einer Instanz als Parameter mit angegeben. Wird innerhalb dieses Timeouts nicht auf das Objekt zugegriffen, so verfällt das erzeugte Objekt nach abgelaufener Zeit. Server aktivierte SingeCall Objekte stellen eine Ausnahme dar, da sie nur für den Zeitraum eines Methodenaufrufes existieren.

Im Gegensatz zu anderen Mechanismen entsteht bei diesem Verfahren kein weiterer Netzwerkverkehr. Das Laufzeitverhalten kann über die beiden Parameter ILease.InitialLeaseTime (Defaultwert: 5 min) und ILease.RenewOnCallTime (Defaultwert: 2 min) verändert werden. Sollte ein Aufrufer das Objekt nach dem Timeout noch benötigen, so besteht die Möglichkeit einen Sponsor zu erzeugen, der über einen Ping Keep-alive-Mechanismus das Objekt am Leben erhält.

2.7 Versionierung

Die Versionierung ist vom erzeugten Objekttyp abhängig. Die Serveraktivierten Objekte verwenden, falls keine Versionsnummer angegeben wurde, stets die neueste Version. Dabei spielt es keine Rolle, gegen welche Version der Client gelinkt wurde. Unterstützt dieser die Version des Servers nicht, wird am Client eine Ausnahme geworfen.

Objekte, die auf dem Client aktiviert werden, kann die zu verwendende Version mitgegeben werden. Beide Kommunikationspartner verwenden dann die vom Client gewünschte Version. Bei fehlender Versionsangabe wird implizit die neueste ausgewählt.

2.8 Sicherheit

Im Gegensatz zu RMI oder SunRPC fehlen bei *.NET Remoting* Sicherheitsmechanismen. Es bleibt dem Programmierer folglich selbst überlassen, geeignete Zugriffsmechanismen zu implementieren. Auf Windows Systemen können jedoch die vom Betriebssystem bereitgestellt Authentifizierungsmechanismen genutzt werden.

3 Beispiele

Folgendes Beispiel implementiert einen Remotetaschenrechner, der zwei Zahlen miteinander multiplizieren kann. Zuerst wird die Mathe Library erstellt:

Listing 1: **MathLib.cs**

```
1  using System;
2
3  public class MathLib : MarshalByRefObject {
4          public int multiply(int number1, int number2) {
5                  return number1 * number2;
6          }
7  }
```

Die Klasse kompiliert man mit
>csc /noconfig /t:library MathLib.cs
zu einem Assembly, um dieses -je nach Objektaktivierungsart- später an den Server und/oder Client zu binden.

Listing 2: **Server.cs**

```
1  using System;
2  using System.Runtime.Remoting;
3
4  public class Server {
5          public static void Main() {
6                  RemotingConfiguration.Configure("Server.exe.config");
7                  Console.Write("Warte auf Anfragen ... ");
8                  Console.ReadLine();
9          }
10 }
```

Den Server kompiliert man mit: >csc /noconfig /r:MathLib.dll Server.cs

Listing 3: **Server.exe.config**

```
1  <configuration>
2    <system.runtime.remoting>
3      <application>
4        <service>
5          <wellknown mode="Singleton"
6                     type="MathLib,MathLib"
7                     objectUri="MathLib.rem" />
8        </service>
9        <channels>
10         <channel ref="http" port="9999" />
```

```
11        </channels>
12      </application>
13    </system.runtime.remoting>
14  </configuration>
```

Listing 4: **Client.cs**

```
1  using System;
2  using System.Runtime.Remoting;
3
4  public class Client {
5          public static void Main() {
6                  RemotingConfiguration.Configure("Client.exe.config");
7                  MathLib lib = new MathLib();
8
9                  Console.WriteLine("Welcome to MathLib Client: ");
10                 Console.Write("Bitte Zahl 1 eingeben:");
11                 string zahl1 = Console.ReadLine();
12                 Console.Write("Bitte Zahl 2 eingeben:");
13                 string zahl2 = Console.ReadLine();
14                 Console.WriteLine(lib.multiply(Convert.ToInt32(zahl1), Convert.ToInt3
15         }
16 }
```

Den Client kompiliert man mit: >csc /noconfig /r:MathLib.dll Client.cs

Listing 5: **Client.exe.config**

```
1  <configuration>
2    <system.runtime.remoting>
3      <application>
4        <client>
5          <wellknown type="MathLib, MathLib"
6                     url="http://localhost:9999/MathLib.rem" />
7        </client>
8      </application>
9    </system.runtime.remoting>
10 </configuration>
```

Wie man hier sehen kann, beinhaltet *.NET Remoting* schon eine Funktion, Konfigurationsdateien zu verwenden, so dass der Programmierer keine weiteren Aufwand dafür betreiben muss.

Literatur

[1] http://www.sdm.de/download/publikationen/OOP2003_Net_Haller.pdf

[2] http://wwwcs.upb.de/cs/ag-schaefer/Lehre/PG/SHUTTLE/seminar/ausarbeitungen/Entwurfsmuster_Ausarbeitung.pdf

[3] http://de.wikipedia.org/wiki/Entwurfsmuster

[4] http://msdn.microsoft.com/library/DEU/cpguide/html/cpconnetremotingoverview.asp

[5] http://www.devgroup-stuttgart.de