

Web Services mit .NET

Autor: Frank Listing, MicroConsult GmbH

Vielfach assoziieren sogar IT-Unternehmen Web Services mit dem Bereitstellen von Informationen im Internet. Weit gefehlt, denn der Begriff benennt einen ganz konkreten Dienst: Funktionsaufrufe über das Internet. Das .NET-Framework hält dem Entwickler mittlerweile zahlreiche Optionen für das einfache Erstellen und Nutzen der neuen Dienste offen.

Ein Web Service offeriert ein ganz bestimmtes Stück Funktionalität im Internet, das von einem Client-Programm wie bei einem RPC (Remote Procedure Call) aufgerufen werden kann. Prinzipiell ist das nichts Neues: Lange bekannte und etablierte Technologien wie CORBA (Common Object Request Broker Architecture), DCOM (Distributed COM) und Java-RMI (Remote Method Invocation) bieten solche Dienste in umfangreichem Maße.

Warum Web Services?

Böse Stimmen behaupten, dass mit etwas Neuem wieder mehr Geld zu verdienen wäre. Das ist zum Glück nicht die einzige Existenzberechtigung der Web Services. Die oben erwähnten Technologien sind immer an bestimmte Plattformen, Programmiersprachen oder Infrastrukturen gebunden, Client und Server stehen in einer engen Beziehung zueinander. Mit Web Services lassen sich dagegen derzeit bestehende Grenzen überwinden. Dem Programmierer wird keine Technologie mehr vorgeschrieben, sondern lediglich das Transportprotokoll SOAP (Simple Object Access Protocol) über HTTP.

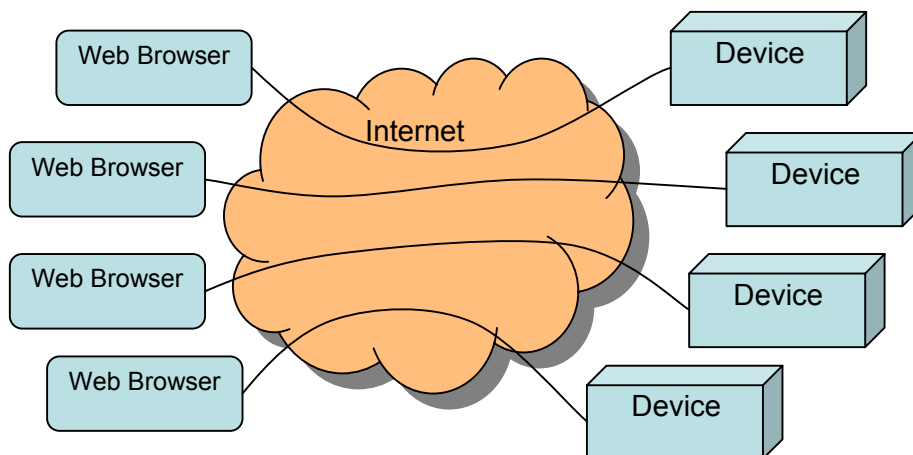
Daraus ergeben sich einige interessante Vorteile:

- Unabhängigkeit von Betriebssystem und Hardware,
- freie Wahl der Programmiersprache,
- problemlose Nutzung über das Intranet, da der HTTP-Port auf den Firewalls freigegeben ist, sowie
- Nutzung von HTTPS für sichere Kommunikation.

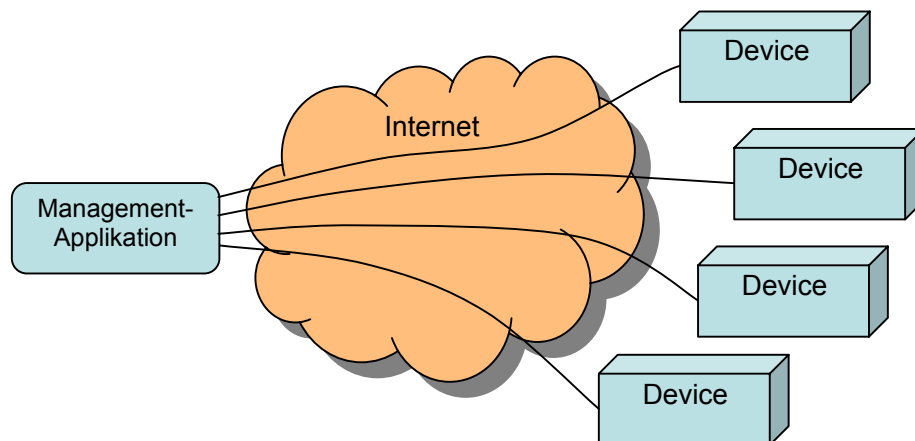
Ein Web Service macht also Funktionalität für andere über das Internet verfügbar. Zum Web Service gehört aber nicht nur das Bereitstellen der Funktion, sondern auch ihre Beschreibung. So kann ein Interessent die Schnittstellenbeschreibung bei einem Web Service erfragen und auf dieser Basis sein Client-Programm entwickeln.

Vorteile – ein typisches Szenario

In der Zeit des Internets bekommen immer mehr Embedded Devices einen Netzwerkanschluss. Waren vor einigen Jahren nur Netzwerkkomponenten wie Router oder Switches mit Möglichkeiten zur Fernwartung versehen, hält diese Technologie auch in anderen Geräten Einzug. Meist läuft auf diesen ein kleiner Web Server, über den die Konfiguration für ein einzelnes Gerät komfortabel angezeigt oder geändert werden kann.



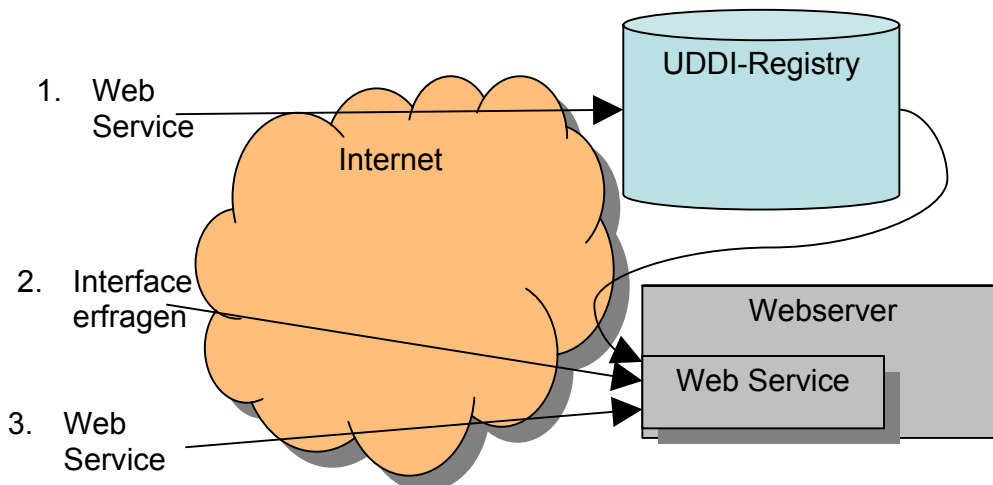
Mit den Web Services werden die Möglichkeiten der Fernwartung, -diagnose und Ferninstallation wesentlich erweitert, es erschließen sich sogar komplett neue Anwendungsfälle.



Fernwartung und -diagnose können nun von einer komplexen Applikation für mehrere Geräte gleichzeitig erfolgen. Besonders interessant ist dies für Geräte, die untereinander in Beziehung stehen. Im Gegensatz zur Variante mit einem simplen Web Server ist hier eine einfache Automatisierung möglich. Auch die Ferninstallation ist mit Web Services direkt und ohne Zwischenschritte durchführbar. So lassen sich einfach neue Firmware-Versionen und Steuerdaten einbringen.

Web Service-Architektur

Bis zur Benutzung eines Web Services sind es im wesentlichen drei Schritte:



- ## 1. Web Service suchen

Für die Suche nach einem passenden Web Service lässt sich eine sogenannte UDDI-Datenbank (Universal Description, Discovery and Integration) zu Rate ziehen. Über Schlüsselworte findet man – falls vorhanden – einen Verweis auf den Server mit dem passenden Web Service. Derzeit gibt es allerdings keine produktiven UDDI-Server im Einsatz, da sich die neuen Dienste erst langsam durchsetzen. Für den Test von UDDI kann man beispielsweise bei uddi.microsoft.com kostenlos einen Web Service anmelden.

- ## 2. Interface erfragen

Wenn der passende Service gefunden ist, muss die Beschreibung des Interfaces erfragt werden. Das Ergebnis ist eine Beschreibung im WSDL-Format (Web Service Description Language).

Auf Basis dieser Vorgabe kann der Entwickler einen Client erstellen, der den Web Service nutzt.

Die Interface-Beschreibung eines einfachen Addierers würde beispielsweise so aussehen:

```
<?xml version="1.0" encoding="utf-8" ?>
<definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"  xmlns:s="http://www.w3.org/2001/XMLSchema"
  xmlns:s0="http://microconsult.de/webservices/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
```

```

xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="http://microconsult.de/webservices/" xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <s:schema elementFormDefault="qualified"
targetNamespace="http://microconsult.de/webservices/">
      <s:element name="Add">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="Op1" type="s:int" />
            <s:element minOccurs="1" maxOccurs="1" name="Op2" type="s:int" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="AddResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="AddResult" type="s:int" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="int" type="s:int" />
    </s:schema>
  </types>
  <message name="AddSoapIn">
    <part name="parameters" element="s0:Add" />
  </message>
  <message name="AddSoapOut">
    <part name="parameters" element="s0:AddResponse" />
  </message>
  <message name="AddHttpGetIn">
    <part name="Op1" type="s:string" />
    <part name="Op2" type="s:string" />
  </message>
  <message name="AddHttpGetOut">
    <part name="Body" element="s0:int" />
  </message>
  <message name="AddHttpPostIn">
    <part name="Op1" type="s:string" />
    <part name="Op2" type="s:string" />
  </message>
  <message name="AddHttpPostOut">
    <part name="Body" element="s0:int" />
  </message>
  <portType name="AddServiceSoap">
    <operation name="Add">
      <input message="s0:AddSoapIn" />
      <output message="s0:AddSoapOut" />
    </operation>
  </portType>
  <portType name="AddServiceHttpGet">
    <operation name="Add">
      <input message="s0:AddHttpGetIn" />
      <output message="s0:AddHttpGetOut" />
    </operation>
  </portType>
  <portType name="AddServiceHttpPost">
    <operation name="Add">
      <input message="s0:AddHttpPostIn" />
      <output message="s0:AddHttpPostOut" />
    </operation>
  </portType>
  <binding name="AddServiceSoap" type="s0:AddServiceSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
    <operation name="Add">
      <soap:operation soapAction="http://microconsult.de/webservices/Add" style="document"
/>
      <input>
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal" />
      </output>
    </operation>
  </binding>
  <binding name="AddServiceHttpGet" type="s0:AddServiceHttpGet">
    <http:binding verb="GET" />
    <operation name="Add">

```

```

        <http:operation location="/Add" />
        <input>
            <http:urlEncoded />
        </input>
        <output>
            <mime:mimeXml part="Body" />
        </output>
    </operation>
</binding>
<binding name="AddServiceHttpPost" type="s0:AddServiceHttpPost">
    <http:binding verb="POST" />
    <operation name="Add">
        <http:operation location="/Add" />
        <input>
            <mime:content type="application/x-www-form-urlencoded" />
        </input>
        <output>
            <mime:mimeXml part="Body" />
        </output>
    </operation>
</binding>
<service name="AddService">
    <port name="AddServiceSoap" binding="s0:AddServiceSoap">
        <soap:address location="http://fl/adder/adder.asmx" />
    </port>
    <port name="AddServiceHttpGet" binding="s0:AddServiceHttpGet">
        <http:address location="http://fl/adder/adder.asmx" />
    </port>
    <port name="AddServiceHttpPost" binding="s0:AddServiceHttpPost">
        <http:address location="http://fl/adder/adder.asmx" />
    </port>
</service>
</definitions>

```

3. Web Service nutzen

Der Client greift nun über das SOAP-Protokoll auf den Web Service zu.

Das Simple Object Access Protocol (SOAP) wurde entwickelt, um Nachrichten in standardisierter Form zu übertragen. Es ist ein XML-Standard und definiert sowohl das Format der Methodenaufrufe als auch das Format der Daten, welche in der Nachricht übertragen werden. Da XML textbasiert ist, kann SOAP auf beliebigen Transportprotokollen aufsetzen. Derzeit ist wegen der Internet-Tauglichkeit hauptsächlich HTTP oder HTTPS im Einsatz. Der SOAP-Request des Addierers setzt sich aus einem HTML-Körper und der in XML codierten Nachricht zusammen. Er sieht folgendermaßen aus:

```

POST /adder/adder.asmx HTTP/1.1
Host: fl
Content-Type: text/xml; charset=utf-8
SOAPAction: "http://microconsult.de/webservices/Add"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <Add xmlns="http://microconsult.de/webservices/">
      <Op1>3</Op1>
      <Op2>6</Op2>
    </Add>
  </soap:Body>
</soap:Envelope>

```

Und die Antwort des Web Services:

```

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <AddResponse xmlns="http://microconsult.de/webservices/">
      <AddResult>9</AddResult>
    </AddResponse>
  </soap:Body>
</soap:Envelope>

```

Die Übertragung der Daten im Textformat bietet klare Vorteile bei der Fehlersuche in der Entwicklungsphase. Allerdings ist die Sichtbarkeit der Daten im Produktiveinsatz selten erwünscht. In diesem Fall ist dann HTTPS als Übertragungsprotokoll zu nutzen, um das unerwünschte Mitlesen von Dritten zu vermeiden. Zur Zeit sind auch Bestrebungen im Gange, eine Verschlüsselung der Daten auf Applikationsebene zu definieren. Das Erstellen dieser Standards ist aber noch nicht abgeschlossen und das Ergebnis noch abzuwarten. Microsoft bietet momentan das WSDK (Web Service Development Kit) an, mit dem sich sichere Web Services generieren lassen. Dieses nutzt den WS-Security-Standard, der sich gerade im Betastadium der Standardisierung befindet.

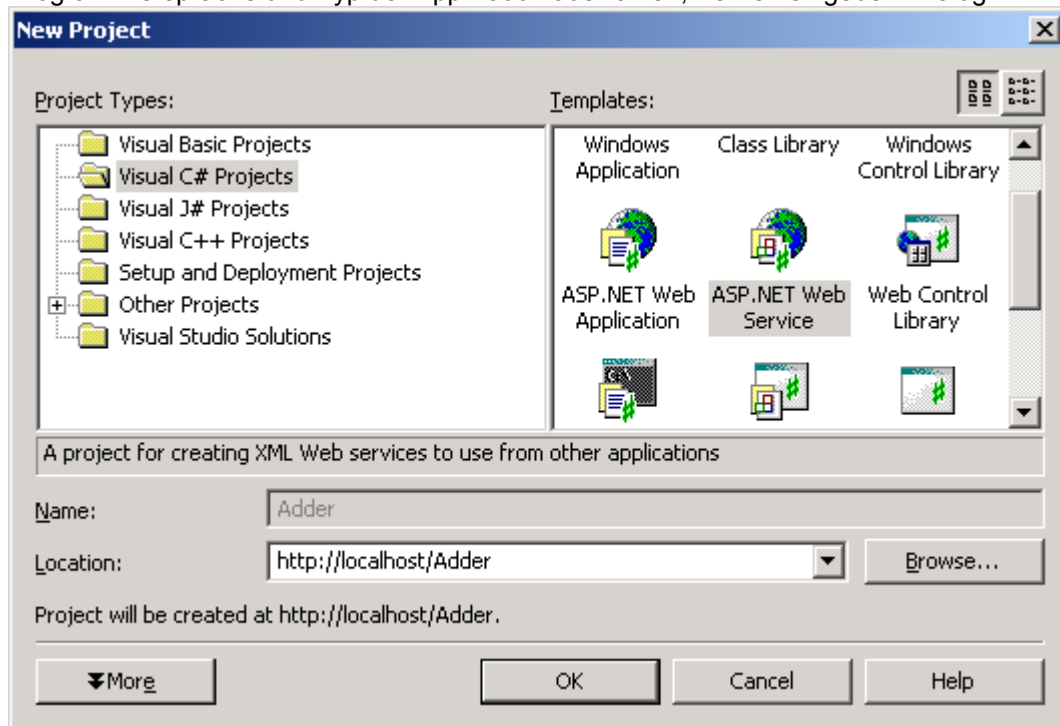
Web Services und .NET

Nach anfänglichen Schwierigkeiten hat Microsoft das große Potential des Internets erkannt und demzufolge in ihre neue .NET-Strategie vieles eingebaut, was die Entwicklung von Internet-Anwendungen erleichtert. So sind auch die Web Services mittlerweile ein integraler Bestandteil des .NET-Frameworks, den es gewinnbringend zu nutzen gilt.

Erstellen eines Web Services

Als Beispiel soll der bereits oben erwähnte, einfache Addierer-Web-Service mit dem Microsoft Visual Studio .NET 7.0 erstellt werden.

Zuerst wird ein neues Web Service-Projekt angelegt. Das geschieht weitgehend automatisch: Programmiersprache und Typ der Applikation auswählen, Namen eingeben – fertig.



Ich habe mich für die Programmiersprache C# entschieden. Aber in .NET sind alle Programmiersprachen gleichberechtigt. Es gibt nun keine Unterschiede im Funktionsumfang mehr. So kann man jetzt z.B. mit Visual Basic .NET die gleichen Programme entwickeln wie mit C# oder J#. Es geht sogar noch weiter: Eine in C# entwickelte Klasse kann ohne Probleme auch in Visual Basic .NET verwendet werden.

Das Visual Studio erzeugt nun eine Reihe von Dateien:

- Den Rumpf für den Quellcode.
- Den Einstieg für den Web Service.
- Einige Steuerdateien.

Weiterhin wird automatisch eine Verbindung zum Web Server aufgebaut und der neue Web Service dort bekannt gemacht. D.h. die Eigenschaften werden im Internet Information Server so eingestellt, dass der Web Service ablauffähig ist. Es ist auch möglich mit dem Debugger des Visual Studio nach Fehlern im Programm zu suchen.

Ein Zauberer (Wizard) hilft nun beim Erstellen einer neuen Methode für den Web Service...

... und diese wird dann mit Leben erfüllt.

Vorher:

```
public int Add(int Op1, int Op2)
{
    return 0;
}
```

Nachher:

```
[WebMethod]
public int Add(int Op1, int Op2)
{
    return Op1+ Op2;
}
```

Kompilieren – fertig.

Mehr ist nicht nötig, um den Web Service zu erstellen. Der Rest wurde automatisch vom Visual Studio erstellt oder ist Bestandteil des .NET-Framework.

Sehr gut gelöst ist auch die saubere Trennung von Informationen für den Internet Information Server und dem Quellcode. Microsoft verwendet dort die sogenannte „Codebehind“-Technik.

Die Datei, die aufgerufen wird, wenn man zu dem Web Service verbindet, besteht aus einer einzigen Zeile:

```
<%@ WebService Language="c#" Codebehind="Adder.asmx.cs" Class="Adder.AddService" %>
```

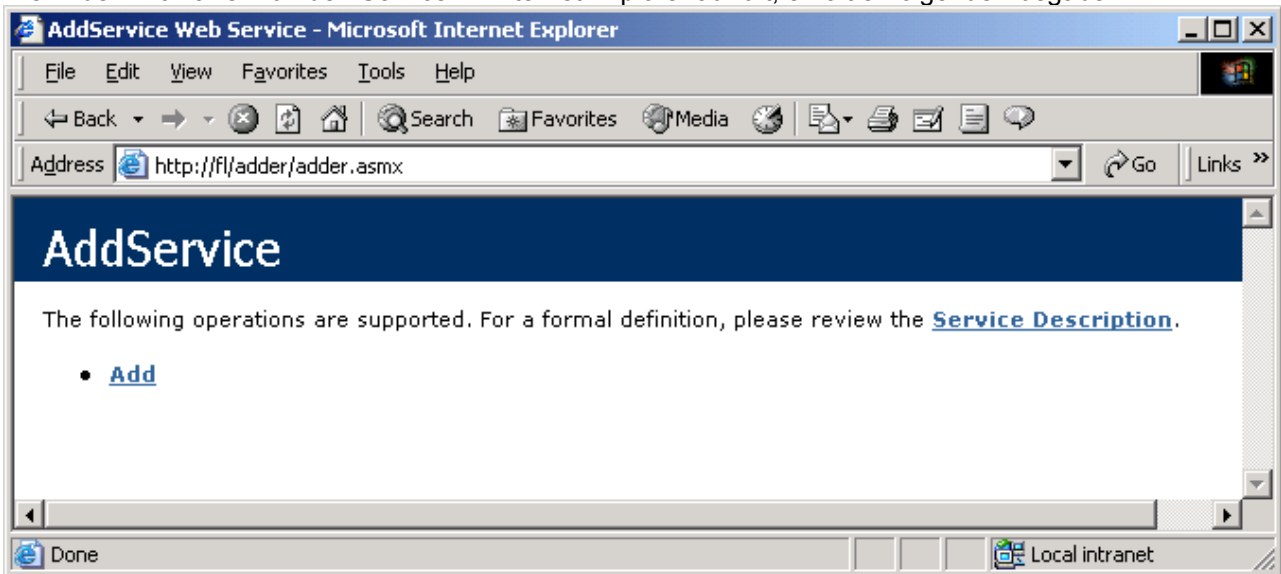
Sie verweist lediglich auf die Klasse AddService im Namensraum Adder, welche in der Datei Adder.asmx.cs definiert ist. Hier ist der gesamte Quellcode enthalten (und auch nur dieser) und sieht so aus:

```
using System;
using System.Web.Services;

namespace Adder
{
    [WebService(Namespace="http://microconsult.de/webservices/")]
    public class AddService : System.Web.Services.WebService
    {
        [WebMethod]
        public int Add(int Op1, int Op2)
        {
            return Op1+ Op2;
        }
    }
}
```

Der Web Service enthält auch ein paar Dienste, die helfen Informationen über ihn zu erhalten und sogar einfache Funktionstests durchzuführen ohne einen speziellen Client zur Verfügung zu haben. Es reicht ein simpler Web Browser.

Wenn der Entwickler nun den Service im Internet Explorer aufruft, erhält er folgende Ausgabe:



Über den Hyperlink ‚Service Description‘ gelangt er an die WSDL-Beschreibung (siehe oben) unseres Services.

Mit dem Link ‚Add‘ lässt sich die Funktion mit dem Browser testen, wobei zusätzlich Beispiele für den Aufruf über SOAP (siehe oben) oder HTML angezeigt werden. Das sind aber nur Hilfen, die der Web Service anbietet. Die Hauptanwendung ist natürlich der Aufruf aus einem Programm heraus.

Alle diese Informationen hat Visual Studio .NET automatisch erstellt. Ein Entwickler kann sich somit auf seine Hauptarbeit, das Entwickeln von Funktionen, beschränken und muss sich nicht mit dem Erstellen von WSDL-Dateien herumschlagen. Die WSDL-Beschreibung wird nun genutzt, um einen Client zu generieren, der diesen Service aufrufen kann.

Erstellen eines Web Service Clients

Um den Web Service von einem Client-Programm aus zu verwenden, muss der Entwickler eine sogenannte „Proxy-Klasse“ erzeugen. Dabei unterstützt ihn das Programm `wsdl.exe`. Mit seiner Hilfe wird direkt aus den Informationen auf dem Web Server der Programmcode für eine Klasse erzeugt, die es dem Entwickler des Clients ermöglicht, die Funktionen des Web Services so zu nutzen, als sei es lokaler Programm-Code. Der Verbindungsaufbau und auch die Umwandlung der Daten in das Netzwerkformat sind völlig transparent und werden vom .NET-Framework übernommen.

Wir starten das Programm in der DOS-Box:

```
wsdl -out:AdderProxy.cs http://fl/Adder/Adder.asmx
```

Wurde diese Zeile auf Kommandoebene ausgeführt, steht der Code der „Proxy-Klasse“ in der Datei `AdderProxy.cs`.

```
//-----  
// <autogenerated>  
//   This code was generated by a tool.  
//   Runtime Version: 1.0.3705.209  
//  
//   Changes to this file may cause incorrect behavior and will be lost if  
//   the code is regenerated.  
// </autogenerated>  
//-----  
  
//  
// This source code was auto-generated by wsdl, Version=1.0.3705.209.  
//  
using System.Diagnostics;  
using System.Xml.Serialization;  
using System;  
using System.Web.Services.Protocols;  
using System.ComponentModel;  
using System.Web.Services;
```

```

/// <remarks/>
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.ComponentModel.DesignerCategoryAttribute("code")]
[System.Web.Services.WebServiceBindingAttribute(Name="AddServiceSoap",
Namespace="http://microconsult.de/webservices/")]
public class AddService : System.Web.Services.Protocols.SoapHttpClientProtocol {

    /// <remarks/>
    public AddService() {
        this.Url = "http://fl/Adder/Adder.asmx";
    }

    /// <remarks/>

[System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://microconsult.de/webservices/Add", RequestNamespace="http://microconsult.de/webservices/",
ResponseNamespace="http://microconsult.de/webservices/",
Use=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)]
    public int Add(int Op1, int Op2) {
        object[] results = this.Invoke("Add", new object[] {
            Op1,
            Op2});
        return ((int)(results[0]));
    }

    /// <remarks/>
    public System.IAsyncResult BeginAdd(int Op1, int Op2, System.AsyncCallback callback, object
asyncState) {
        return this.BeginInvoke("Add", new object[] {
            Op1,
            Op2}, callback, asyncState);
    }

    /// <remarks/>
    public int EndAdd(System.IAsyncResult asyncResult) {
        object[] results = this.EndInvoke(asyncResult);
        return ((int)(results[0]));
    }
}

```

Diese Datei ist in das Client-Projekt einzufügen. Jetzt reichen zwei Zeilen in unserem Programm, um die Add-Methode zu testen:

```

AddService ads= new AddService();

Console.WriteLine("5+ 6= {0}", ads.Add(5, 6));

```

Beim Erstellen des Clients ist es unwichtig, wer diesen Web Service zur Verfügung stellt und mit welcher Programmiersprache er erstellt wurde. Es muss nur eine WSDL-Beschreibung verfügbar sein.

Wir haben gesehen, dass auch die Erstellung eines Clients ganz einfach ist. Ein kleiner Wehmutstropfen bleibt dennoch: Ohne .NET geht es nicht. Wer kein .NET zur Verfügung hat, muss weiterhin viel Handarbeit erledigen oder Komponenten von Drittanbietern dazukaufen.

Fazit

Der große Nutzen der Web Services liegt einerseits in der Möglichkeit, die unterschiedlichsten Plattformen zu verbinden und andererseits in der Option, Funktionen über das Internet bereit zu stellen oder zu verwenden, ohne dass die Verbindung an einer Firewall scheitert. Damit lassen sich heute Aufgaben einfach lösen, die vorher viel Aufwand bedeutet haben. Das Microsoft .NET-Framework unterstützt Entwickler umfassend beim Erstellen und Einsatz von Web Services.