

Kapitel 29

Webservices mit der Windows Communication Foundation (WCF)

In diesem Kapitel:

Basisfunktionsumfang der WCF	606
Neuerungen in der WCF 3.5	607
Neuerungen in WCF 3.5 Service Pack 1	608
Architektur	608
Kompatibilität	613
Werkzeuge	614
Erstellung von WCF-Diensten und WCF-Servern	621
Erstellung eines WCF-Clients	641
REST-basierte WCF-Dienste	647
Persistente WCF-Dienste (Durable Services)	649
WCF-Sicherheit	651
Protokollierung	655
Weitere Funktionen	659
Beispiele in World Wide Wings	664
Übliche Stolpersteine	667
Weitere Möglichkeiten von WCF	672
Fazit zu WCF	673

Das .NET Framework bot bereits seit seiner ersten Version mit .NET Remoting und den auf ASP.NET basierenden XML-Webservices (kurz: ASMX) zwei verschiedene Ansätze für Fernaufrufe und Anwendungskopplung. Mit der Windows Communication Foundation (WCF) wollte Microsoft diese beiden Ansätze vereinheitlichen und nebenbei auch Funktionen aus anderen Technologien für verteilte Anwendungen wie Microsoft Message Queuing (MSMQ) und die .NET Enterprise Services (alias COM+-Anwendungsdienste) integrieren. Die WCF ist seit .NET 3.0 verfügbar und wurde sowohl in .NET 3.5 als auch in .NET 3.5 Service Pack 1 nicht unerheblich erweitert. Die WCF ist auch für Silverlight verfügbar, das heißt Silverlight kann WCF-Dienste aufrufen.

Basisfunktionsumfang der WCF

Dieser Abschnitt beschreibt die Grundeigenschaften der WCF, die mit der ersten Version der WCF im .NET Framework 3.0 eingeführt wurden.

Grundeigenschaften

Es gibt drei herausragende Eigenschaften der WCF:

- Die verschiedenen Aspekte verteilter Kommunikation (Schnittstelle, Implementierung, Kommunikationsprotokoll, Infrastrukturdienste, Hosting) lassen sich gut logisch trennen.
- Es gibt sowohl Unterstützung für plattformübergreifende Kommunikationsstandards (W3C/WS-*) als auch proprietäre Protokolle der .NET-Welt.
- Die Kommunikationsform kann ohne Neukompilierung eines Dienstes geändert werden.

Die Qual der Wahl

Die WCF stellt für die Kommunikation zwischen Prozessen oder Systemen zahlreiche frei kombinierbare Funktionen wie Authentifizierung, Verschlüsselung, Autorisierung, Integrität, Zuverlässigkeit, Transaktionen und Nachrichtenwarteschlangen bereit. Wie bei .NET Remoting sind das Übertragungsprotokoll und das Serialisierungsformat wählbar. Microsoft wird zunächst nur die Übertragung per HTTP, HTTPS, TCP, MSMQ sowie Named Pipes unterstützen. Durch die Erweiterbarkeit der WCF sind aber andere Protokolle (z.B. SMTP, UDP) von Drittanbietern ergänzbar.

Bei der Serialisierung (vgl. auch Kapitel »NET-Klassenbibliothek 3.5«, Abschnitt »Serialisierung« im Buch »NET 3.5 Crashkurs« [HS02]) besteht die Wahl zwischen dem textbasierten SOAP, dem Message Transmission Optimization Mechanism (MTOM) und einem binärkodierte SOAP, das Microsoft proprietär entwickelt hat. Die WCF unterstützt viele bestehende Webservicestandards wie WS-Security, WS-Trust, WS-AtomicTransaction, WS-Coordination, WS-SecureConversation und WS-ReliableMessaging.

Zum Hosting von WCF-Diensten stehen dem Entwickler ebenfalls mehrere Optionen zur Verfügung, die von einer einfachen Konsolenanwendung über einen Windows-Dienst und den Internet Information Services (IIS) bis zu dem in Windows integrierten Anwendungsserver COM+ (alias .NET Enterprise Services) reichen. Für die neue Betriebssystemgeneration entwickelt Microsoft mit Windows (Process) Activation Service (WAS) (früherer Name *Webhost*) einen leichtgewichtigen Host für WCF-Dienste.

Migration

Auf den ersten Blick beeindruckt die Vielfalt der in der WCF verfügbaren Optionen. Ein Entwickler verteilter Anwendungen muss sich jedoch bewusst sein, dass die WCF dem Paradigma *Serviceorientierung* folgt. Die in .NET Remoting verfügbare objektorientierte Anwendungsverteilung, z.B. das Übergeben einer Objektreferenz, entfällt. Außerdem wird das in .NET Remoting verwendete Binärformat nicht mehr unterstützt, sodass ein .NET Remoting-Client gar nicht mit einem WCF-Dienst »reden« kann. Da das Binärformat nicht dokumentiert ist, hilft es wenig, dass man die WCF erweitern kann. Entwickler, die auf .NET Remoting gesetzt haben, müssen migrieren oder mit einer alten, zukünftig von Microsoft nicht mehr weiterentwickelten Technologie leben. COM+-Anwendungen hingegen kann man als WCF-Dienst veröffentlichen, und mit COM-Clients lassen sich WCF-Dienste aufrufen.

Neuerungen in der WCF 3.5

Das .NET Framework 3.5 enthält die zweite Version der WCF (alias WCF 3.5) mit folgenden Neuerungen:

- Persistente Dienste für lang andauernde Konversationen (Kommunikationsbeziehungen). Diese Funktion heißt offiziell *Durable Services*,
- Unterstützung der WCF in der Windows Workflow Foundation (siehe Kapitel zu »Windows Workflow Foundation (WF) im Buch [HS02]«),
- RSS 2.0- und ATOM 1.0-Unterstützung (Namensraum `System.ServiceModel.Syndication` mit `Atom10Serializer` und `Rss20Serializer`),
- Webservices ohne SOAP (POX-Webservices) durch ein neues `WebHttpBinding`, ein `HttpTransferEndpointBehavior` und die neue Annotation `HttpTransferContract` (z.B. zum Erzeugen eines RSS-Feeds mit der WCF),
- `WS2007HttpBinding`: Unterstützung der Standards WS-AT und WS-Coordination jeweils in Version 1.1,
- JSON-Serialisierung (`DataContractJsonSerializer`) für AJAX-Webservices,
- Unterstützung der WCF in ASP.NET AJAX (`System.ServiceModel.Web`, `WebHttpBinding`),
- E-Mail-basierter Transport auf Basis des Microsoft Exchange Servers (`Microsoft.ServiceModel.Channels.Mail.*`),
- Mehr Unterstützung für Betrieb von WCF-Diensten unter eingeschränkten Rechten (*Allow Partially Trusted Callers*),
- Unterstützung für neuere WS-*-Standards:
 - WS-RM: Web Services Reliable Messaging v1.1
 - WS-SecureConversation: WS-SecureConversation v1.3
 - WS-Trust: WS-Trust v1.3
 - WS-SecurityPolicy: WS-SecurityPolicy v1.2
 - WS-AT: Web Services Atomic Transaction (WS-AtomicTransaction) Version 1.1
 - WS-Coordination: Web Services Coordination (WS-Coordination) Version 1.1

Neuerungen in WCF 3.5 Service Pack 1

Das Service Pack 1 von .NET 3.5 enthält weitere Neuerungen für die WCF, sodass man hier von der dritten Version der WCF sprechen kann.

Neu sind insbesondere:

- POJO-Serialisierung (Objekte ohne Serialisierungsannotation) in den Klassen `DataContractSerializer` und `NetDataContractSerializer`,
- Verbesserungen des WCF-Testclients,
- Generierung eines IIS-basierten WCF-Hosts in Visual Studio 2008,
- Verbesserungen für die Serialisierung von Referenzen (`DataContractSerializer`),
- Bessere Übermittlung von Fehlern beim Einsatz von `XmlSerializer`.

HINWEIS

Bitte beachten Sie, dass insbesondere zu den letzten beiden Punkten bis zum Redaktionsschluss keine Dokumentation und druckfähige Stellungnahme von Microsoft vorlag. Dies bedeutet, dass diese Punkte hier leider nicht näher erläutert werden können.

Architektur

Abbildung 29.1 zeigt die Grundbestandteile des WCF-Konzeptes und den Aufbau von WCF-Client bzw. Server:

- Ein WCF-Dienst ist aus der Sicht des Servers eine .NET-Klasse, die mit `[ServiceContract]` annotiert ist. In der Regel ist diese Klasse nur eine Fassade für die eigentliche Geschäftslogik, die in einer anderen Klasse implementiert ist. Das Fassadenkonzept dient der Trennung zwischen Logik und Infrastruktur.
- Die WCF-Klasse besteht aus Methoden, die mit `[OperationContract]` annotiert sind.
- Der WCF-Dienst wird in einem WCF-Host bereitgestellt. Die Art der Bereitstellung wird durch eine Konfigurationsdatei in der Sektion `<system.serviceModel>` festgelegt. Alternativ ist auch eine Festlegung im Programmcode möglich.
- Der Client bezieht von dem WCF-Host online oder offline Metadaten. Aus diesen Metadaten wird eine Proxyklasse generiert, die der Client für die Zugriffe nutzt. Der Client speichert den Standort des Dienstes und die Kommunikationsform seinerseits in der Regel in einer Konfigurationsdatei in der Sektion `<system.serviceModel>`.
- Der Client kann danach die Methoden der generierten Proxyklasse aufrufen; die eigentliche Kommunikation ist transparent, das heißt, die WCF-Infrastruktur kümmert sich um Serialisierung und Deserialisierung, Versand und gegebenenfalls aktivierte Sicherheitsfunktionen.

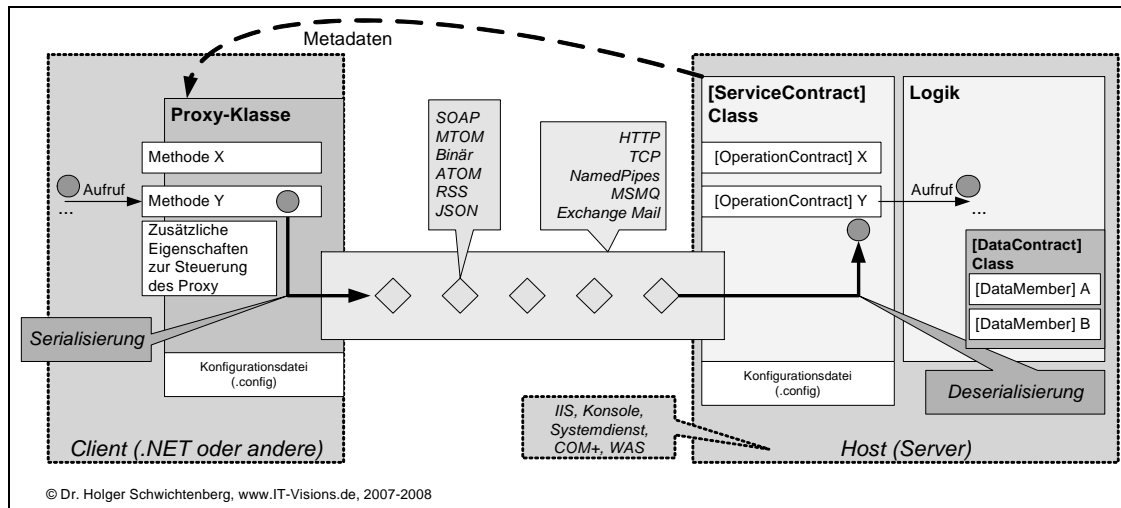


Abbildung 29.1 Bestandteile des WCF-Konzeptes

ABC-Eigenschaften

Ein WCF-Dienst besitzt einen oder mehrere so genannte *Endpunkte* mit verschiedenen Eigenschaften (Abbildung 29.2):

- Die Adresse (Address) besteht aus einem URL mit Protokollangabe, Pfad und Port.
- In der Bindung (Binding) sind die bei der Kommunikation zu verwendenden Protokolle und Formate festgelegt.
- Der Vertrag (Contract) definiert die eigentlichen Operationen, die der Dienst anbietet. Dies geschieht in der Regel durch eine Schnittstellendefinition im Programmcode.

Microsoft spricht daher im Zusammenhang mit WCF-Diensten von den *ABC-Eigenschaften*.

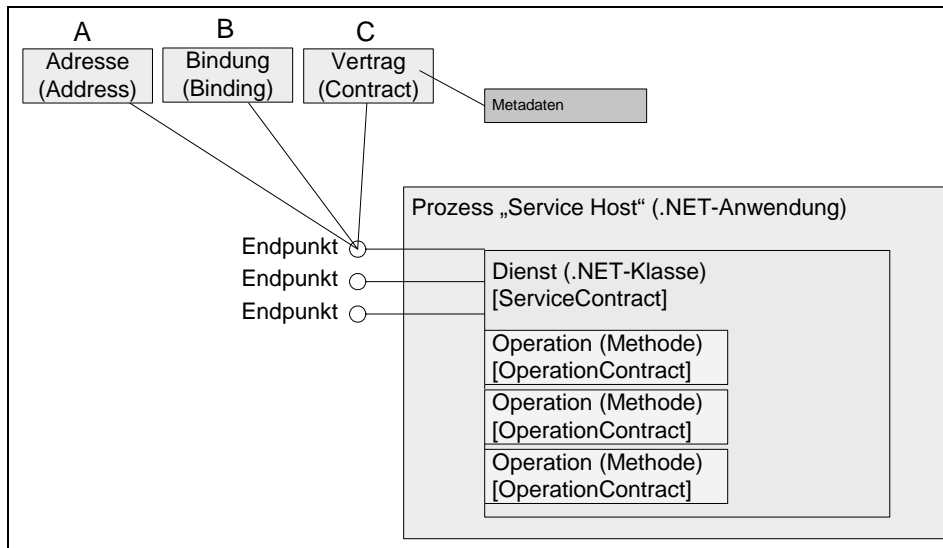


Abbildung 29.2 Aufbau eines WCF-Dienstes

HINWEIS

In Zusammenhang mit Webservices hört man oft auch den Begriff »Contract First«. Damit ist gemeint, dass man erst die Datenaustauschformate festlegt und anschließend eine dazu passende Implementierung (automatisch) erstellt. »Contract Last« ist das Gegenteil, bei dem man den Programmcode zuerst schreibt.

Wie sooft gibt es auch hier ganz verschiedene Einstellungen bei den Entwicklern. Einige Entwickler verstehen unter Contract First, dass sie eine Beschreibung des Dienstes in WSDL/XSD erstellen, und dann daraus Programmcode generiert werden soll. Andere Entwickler beginnen mit einer Schnittstellendefinition und erstellen daraus die Implementierung und nennen dies bereits Contract First. Da es keine allgemeine Definition von Contract First gibt, kann man nicht sagen, dass eine der beiden Seiten falsch liegen würde.

Bindungen (Binding)

Grundsätzlich gibt es in der WCF viele mögliche Konfigurationsmöglichkeiten, das heißt, man kann Kommunikationsprotokolle, Serialisierungsformate sowie die Einstellungen für Sicherheit und Zuverlässigkeit auf unterschiedliche Weise miteinander kombinieren. Eine konkrete Zusammenstellung von Kommunikationseinstellungen bezeichnet man als Bindung (engl. Binding).

Vordefinierte Bindungen (WCF-Systembindungen)

Aus der Vielzahl der sich daraus ergebenden Möglichkeiten hat Microsoft übliche und sinnvolle Szenarien zu vordefinierten Systembindungen zusammengefasst. Die folgende Tabelle zeigt neun in .NET 3.0 vordefinierte Bindungen. Die Tabelle ist aus der Originaldokumentation entnommen und verändert worden. Die Originaltabelle [MSDN18] enthält leider einige unsinnige Angaben. Soweit offensichtlich, wurden diese modifiziert. Außerdem wurden einige fehlende Bindungsarten ergänzt. Bitte beachten Sie, dass der Autor nicht jede einzelne Angabe aus dieser Tabelle selbst geprüft hat. Angaben in Klammern sind Optionen.

In der Tabelle fehlen die Bindungen für Metadaten, die in [MSDN18] ebenso wenig genannt werden. Die einzelnen Einstellungen werden in den nächsten Kapiteln noch erläutert.

Bindung	Protokoll	Serialisierung	Interoperabilität	Sicherheit	Zustand	Zuverlässigkeit	Trans- aktion	Duplex
BasicHttpBinding	HTTP	SOAP, (MTOM)	Webservices mit Basic Profile 1.1 (das heißt ASMX)	Keine (Transport, Nachricht)	Nein	Nein	Nein	Nein
WebHttpBinding	HTTP	Plain Old XML ("POX")	Alle Systeme, die HTTP und XML beherrschen	Keine	Nein	Nein	Nein	Nein
WSHttpBinding	HTTP	SOAP, (MTOM)	Webservices	Nachricht (keine, Transport, gemischt)	Optional	Optional	Nein (ja)	Nein
WSHttpContextBinding	HTTP	SOAP, (MTOM)	Webservices	Nachricht (keine, Transport, gemischt)	Optional	Optional	Nein (ja)	Nein
WS2007HttpBinding	HTTP	SOAP, (MTOM)	WS-Security, WS-Trust, WS-SecureConversation, WS-SecurityPolicy	Nachricht (keine, Transport, gemischt)	Optional	Optional	Nein (ja)	
WSDualHttpBinding	HTTP	SOAP, (MTOM)	Webservices	Nachricht (keine)	Optional	Ja	Nein (ja)	Ja
WSFederationHttpBinding	HTTP	SOAP, (MTOM)	Webservices mit WS-Federation	Nachricht (gemischt, keine)	Nein	Optional	Nein (ja)	Nein
WS2007FederationHttpBinding	HTTP	SOAP, (MTOM)	Webservices mit WS-Federation	Nachricht (gemischt, Keine)	Nein	Optional	Nein (ja)	Nein
NetTcpBinding	TCP	Binär	.NET	Transport (Nachricht, keine, gemischt)	Ja	Optional	Nein (ja)	Ja
NetNamedPipeBinding	Named Pipes	Binär	.NET	Transport (keine)	Ja	Nein	Nein (ja)	Ja
NetMsmqBinding	MSMQ	Binär	.NET	Transport (Nachricht, Transport+Nachricht gleichzeitig)	Nein	Nein	Nein (ja)	Nein
NetPeerTcpBinding	TCP	Binär	.NET	Transport	Nein	Nein	Nein	Ja
MsmqIntegrationBinding	MSMQ	Binär	MSMQ	Transport	Nein	Nein	Nein (ja)	Nein

Tabelle 29.1 Vordefinierte Bindungen in WCF (Quelle: [MSDN18])

HINWEIS

Zur Schreibweise der Bindungsnamen: Die Klassennamen beginnen immer mit einem Großbuchstaben. In den Konfigurationsdateien ist der Name jedoch mit einem kleinen Anfangsbuchstaben zu verwenden.

Verwendung von WCF-Bindungen

Ein Entwickler hat folgende Möglichkeiten, um die Kommunikationseigenschaften festzulegen:

- Verwendung einer der vordefinierten Systembindungen,
- Modifikation einer der vordefinierten Systembindungen,
- Erstellung einer eigenen Bindung (*Custom Binding*).

Zur Anwendung einer Bindung weist man in der Anwendungskonfigurationsdatei einem Dienst einen Endpunkt zu und hinterlegt dort im Attribut `binding` den Namen der Bindung:

```
<system.serviceModel>
  <services>
    <service name="de.WWWings.Dienste.FlugplanService" >
      <endpoint address="net.tcp://e01.IT-Visions.local:1234/FlugplanService"
        binding="netTcpBinding" bindingConfiguration="" name="FlugplanService_TCP"
        contract="de.WWWings.Dienste.IFlugplanService" />
    </service>
  </services>
</system.serviceModel>
```

Listing 29.1 Verwendung einer vordefinierten Bindung

Zur Modifikation einer Bindung setzt man zusätzlich das Attribut `bindingConfiguration` auf den Namen einer Konfigurationsänderung, die man im Element `<bindings>` hinterlegt hat. In dem folgenden Beispiel wird die Sicherheit der vordefinierten Bindung `NetTcpBinding` für einen Endpunkt deaktiviert:

```
<system.serviceModel>
  <bindings>
    <netTcpBinding>
      <binding name="tcp_Unsecured">
        <security mode="None" />
      </binding>
    </netTcpBinding>
  </bindings>
  <services>
    <service name="de.WWWings.Dienste.FlugplanService">
      <endpoint address="net.tcp://e01.IT-Visions.local:1234/WWWings/Flugplanservice"
        binding="netTcpBinding" bindingConfiguration="tcp_Unsecured"
        name="Flugplanservice_TCP" contract="de.WWWings.Dienste.IFlugplanService" />
    </service>
  </services>
</system.serviceModel>
```

Listing 29.2 Modifikation einer vordefinierten Bindung

WCF-Bindungen sind Klassen, die von `System.ServiceModel.Channels.Binding` abgeleitet sind. Eigene Bindings definiert man über die Klasse `System.ServiceModel.Channels.CustomBinding` bzw. das XML-Element `<customBinding>`.

Assemblies

Sowohl WCF-Clients als auch WCF-Server brauchen eine Referenz auf *System.ServiceModel.dll*. Wenn der Server die neuen Serialisierer von WCF verwendet, benötigt er auch eine Referenz auf *System.Runtime.Serialization.dll*.

HINWEIS Die WCF ist derzeit noch nicht für das .NET Compact Framework verfügbar. Eine solche Implementierung ist aber laut [MSDNBLOG01] »in Arbeit«.

Kompatibilität

Ein Wermutstropfen ist, dass die WCF nur eingeschränkt kompatibel mit den Vorgängern ASMX und .NET Remoting ist. Bei der Betrachtung der Kompatibilität ist zu differenzieren zwischen:

- Interoperabilität zur Laufzeit: Kann ein bestehender Endpunkt aus der einen Technologie mit einem Endpunkt der anderen Technologie verbunden werden? Hier spricht man auch von der Kompatibilität *auf der Leitung*.
- Migration: Wie groß ist der Aufwand, einen Endpunkt auf die neue Technologie umzustellen?
- Parallelbetrieb: Können in einer einzigen Anwendung sowohl Endpunkte der alten als auch der neuen Technologie angeboten werden?

	Interoperabilität zur Laufzeit	Migrationsaufwand	Parallelbetrieb
.NET Remoting	Nicht kompatibel	Hoch	Möglich
ASP.NET-basierte XML-Webservices (ASMX)	Kompatibel, wenn Basic Profile 1.1 eingesetzt wird	Niedrig	Möglich
Andere XML-Webservices gemäß Basic Profile 1.1	Kompatibel	Nicht zutreffend	Nicht zutreffend
Andere XML-Webservices mit WS-* -Protokollen	Bedingt kompatibel (abhängig von den verwendeten WS-* -Protokollen)	Nicht zutreffend	Nicht zutreffend

Tabelle 29.2 Kompatibilität der WCF zu anderen Techniken

HINWEIS Der Parallelbetrieb von .NET Remoting, der WCF und den ASP.NET Webservices für dieselben Objekte ist möglich, das heißt, ein .NET-Objekt kann gleichzeitig über alle drei Verfahren unterschiedlichen Clients angeboten werden.

Die WCF deckt ASMX vollständig ab. Ein ASMX-Server oder -Client ist kompatibel zur WCF. Manche WCF-Dienste sind auch kompatibel zu ASMX. Die Migration von ASMX zu WCF ist per Suchen und Ersetzen möglich. Auch wenn es zum Teil funktionale Ähnlichkeiten zu .NET Remoting gibt, ist dennoch weder Interoperabilität noch Migration zwischen WCF und .NET Remoting möglich.

Werkzeuge

In Visual Studio Version 2005 gab es nur sehr spärliche WCF-Werkzeuge. Visual Studio 2008 bietet dagegen bessere Werkzeuge. Weitere Werkzeuge findet man im Windows SDK 6.0.

Visual Studio-Projektvorlagen

Visual Studio 2008 bietet folgende Projektvorlagen für die WCF:

- *WCF-Dienstbibliothek (WCF Service Library)*: Klassenbibliothek mit einem einfachen WCF-Dienst und zugehöriger Konfigurationsdatei.
- *Sequenzielle Workflow-Dienstbibliothek (Sequential Workflow Service Library)*: Klassenbibliothek mit einem WCF-Dienst, der einen sequenziellen Workflow startet.
- *Statuscomputerworkflow-Dienstbibliothek (State Maschine Workflow Service Library)*: Klassenbibliothek mit einem WCF-Dienst, der einen zustandsbasierten Workflow startet.
- *Syndication-Dienstbibliothek (Syndication Service Library)*: Klassenbibliothek mit ATOM- oder RSS-Feed.
- *WCF-Dienstanwendung (WCF Service Application)* (in der Sektion *Web*): Ein WCF-Dienst, der in einer Webanwendung nach dem Webanwendungsmodell betrieben wird.
- *WCF-Dienst (WCF Service)* (unter *Neue Website (New Web Site)*): Ein WCF-Dienst, der in einer Webanwendung nach dem Websitemodell betrieben wird.

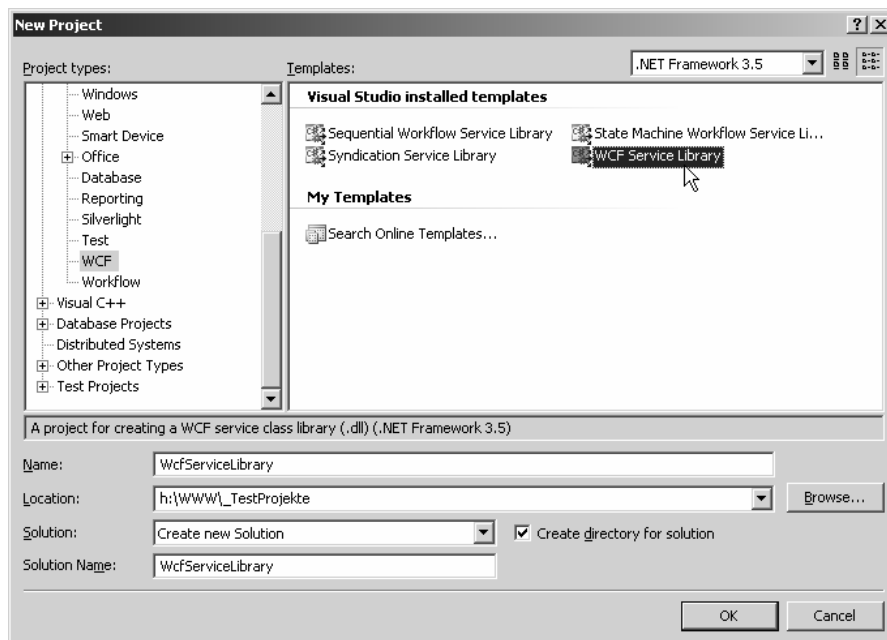


Abbildung 29.3 Projektvorlagen für die WCF in Visual Studio 2008

ACHTUNG Alle Projektvorlagen erzeugen immer HTTP-basierte Dienste. Es gibt leider keine Vorlagen für TCP-basierte Dienste oder einen nicht webbasierten WCF-Host. Diese können Sie sich aber selbst anlegen (vgl. Kapitel zu Visual Studio 2008, die Sie als PDF-Dokument unter dem Namen »Grundlagen.pdf« auf der DVD zu diesem Buch finden).

Visual Studio-Elementvorlagen

Bestehenden Projekten kann man Elemente des Typs *WCF Service* hinzufügen. In Webprojekten steht dahinter sowohl eine *.svc*-Datei als auch eine Klasse sowie eine Schnittstellendefinition und eine Sektion in der Konfigurationsdatei. In anderen Projekten werden nur die Klasse, die Schnittstellendefinition und die Konfigurationssektion erstellt.

Service Configuration Editor

Der *Microsoft Service Configuration Editor* ist eine eigenständige Windows-Anwendung (*SvcConfigEditor.exe*), die mit dem Windows SDK mitgeliefert wird. Der Service Configuration Editor dient dem Erstellen von Konfigurationsabschnitten für die WCF. Die Implementierung des Werkzeugs befindet sich im Namensraum `Microsoft.Tools.ServiceModel.ConfigurationEditor`.

Visual Studio 2008 bietet die Möglichkeit, den Microsoft Service Configuration Editor über den Menüpunkt *Tools/WCF Configuration Editor* oder im Kontextmenü einer Konfigurationsdatei (*Edit WCF Configuration* bei einer *app.config*- oder *web.config*-Datei) aufzurufen.

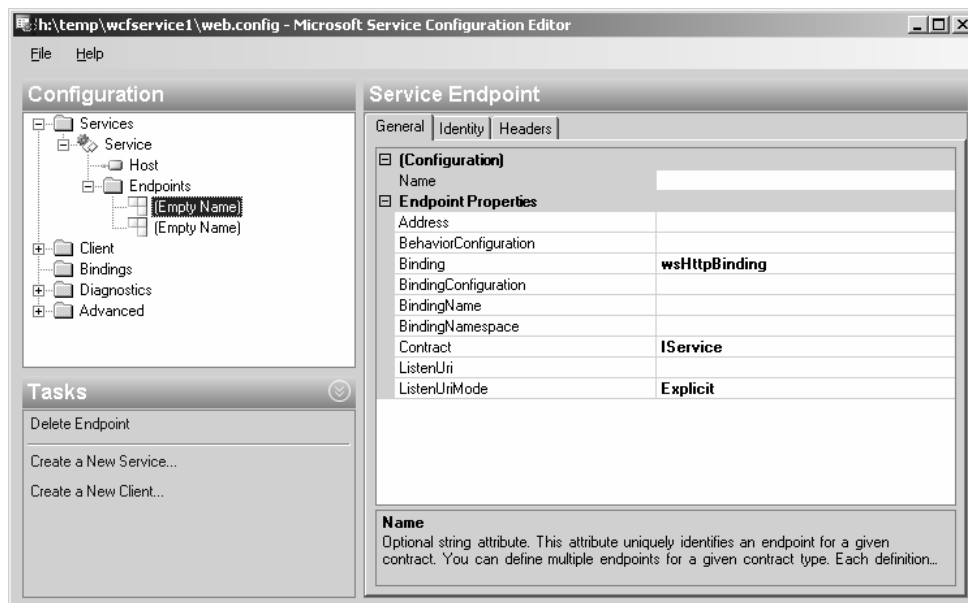


Abbildung 29.4 Ein WCF-Dienst mit zwei Endpunkten im Konfigurationseditor

Visual Studio-Proxygenerator

Für verteilte Anwendungen mit der Windows Communication Foundation (WCF) hat Microsoft den Dialog zur Erstellung eines Proxys für den Client erheblich erweitert (Abbildung 29.5). Im Kontextmenü eines .NET 3.x-Projekts findet man nur noch den Eintrag *Add Service Reference* für Proxys im WCF-Stil. Um einen Proxy im .NET 1.x/2.0-Stil zu erstellen, steht unter *Advanced* die Schaltfläche *Add Web Reference* zur Verfügung. *Add Web Reference* erscheint nur noch im Kontextmenü eines Projekts und im Menü *Project*, wenn als *Target Framework* das *.NET Framework 2.0* gewählt ist.

Alternativ steht das Kommandozeilenwerkzeug *svcutil.exe* zur Verfügung.

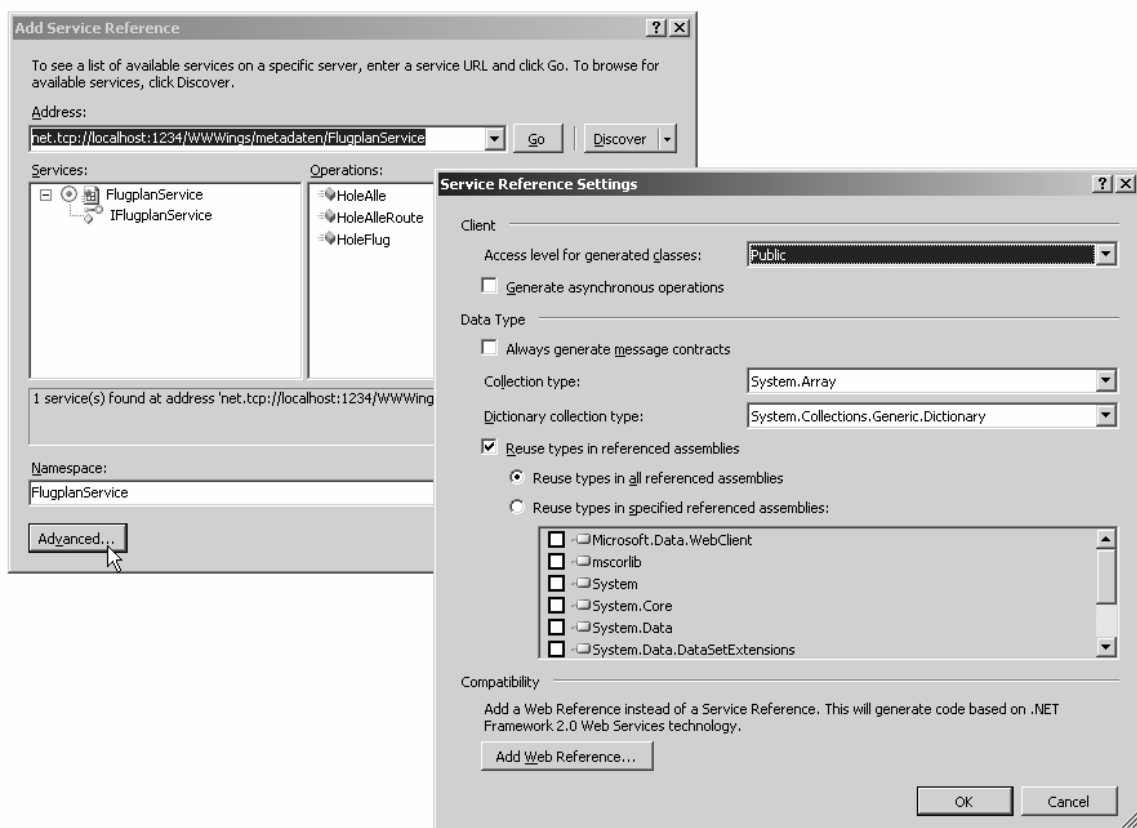


Abbildung 29.5 Erstellen eines WCF-Proxys

WCF-Testhost

Visual Studio 2008 enthält einen WCF-Host zum Testen von WCF-Diensten. Dieser in *WcfSvcHost.exe* implementierte WCF-Server stellt automatisch alle in der Konfigurationsdatei konfigurierten Dienste bereit. Damit kann man eine WCF-Dienstbibliothek auch ohne explizites Bereitstellen eines WCF-Servers testen.

WICHTIG Voraussetzung ist, dass das Klassenbibliotheksprojekt mit der WCF-Dienstbibliothek eine eigene *app.config*-Datei besitzt. Eigentlich weisen Klassenbibliotheksprojekte keine eigenen Konfigurationsdateien auf, weil Konfigurationsdateien nur auf EXE-Dateien wirken.

Der WCF-Testhost und auch der WCF-Testclient (siehe nächster Abschnitt) werden für ein Projekt automatisch aktiviert, wenn man die Projektvorlage *WCF Service Library* verwendet. Das manuelle Einbinden macht viel Arbeit und sollte vermieden werden. Bei der Übernahme von WCF-Dienstbibliotheken aus Visual Studio Version 2005 legt man am besten ein neues WCF-Projekt in Visual Studio 2008 an und kopiert dann Klassen und Konfiguration aus dem Visual Studio Version 2005-Projekt dorthin.

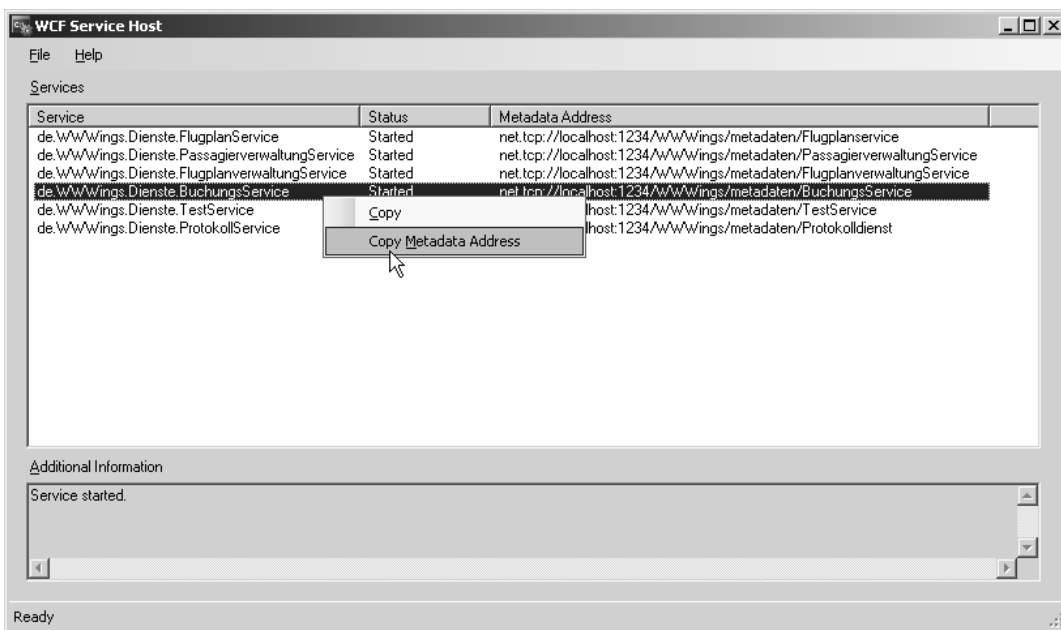


Abbildung 29.6 Der WCF-Testhost zeigt sechs WCF-Dienste

ACHTUNG *WcfSvcHost.exe* ist kein allgemeiner WCF-Server und ist nicht für den Einsatz im Produktivbetrieb geeignet. Er dient nur dem Test, insbesondere mit dem WCF-Testclient.

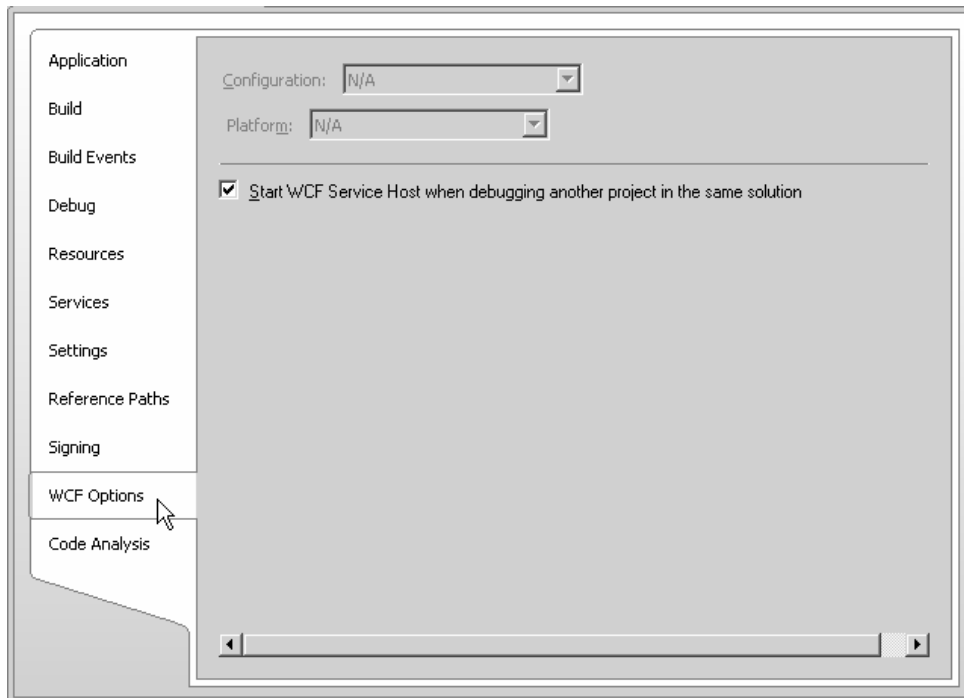


Abbildung 29.7 Aktivieren und Deaktivieren des WCF-Testhosts

ACHTUNG Der Testhost ist sehr hilfreich, aber er wird lästig, wenn man einen selbstgeschriebenen Host testen will. Dann kommen sich die beiden Hosts in die Quere (Typische Meldung »There is already a listener on IP endpoint...«), wenn in dem eigenen Host und dem Testhost die gleichen Adressen konfiguriert sind. Zum Glück kann man den Testhost in den Projekteigenschaften deaktivieren (Abbildung 29.7).

WCF-Testclient

Visual Studio 2008 enthält (im Gegensatz zu Visual Studio Version 2005) einen Testclient für WCF-Anwendungen. Einen Testclient gibt es schon seit .NET 1.0 für ASP.NET-basierte Webservices. Dieser Testclient ist eine Webanwendung und arbeitet nur mit HTTP-/SOAP-basierten Webservices. Der Testclient in Visual Studio 2008 ist eine Desktopanwendung (*WcfTestClient.exe*), die mit allen WCF-Diensten funktioniert. Ein weiterer Vorteil des neuen Testclients ist, dass man auch komplexe Datentypen als Parameter übergeben kann.

HINWEIS Der Testclient wird beim Start des WCF-Testhosts automatisch gestartet, wenn Sie die Projektvorlage *WCF Service Library* verwendet haben. Dies erfolgt über den Eintrag `/client:»WcfTestClient.exe«` unter den Kommandozeilenargumenten für das Debugging in den Projekteigenschaften.

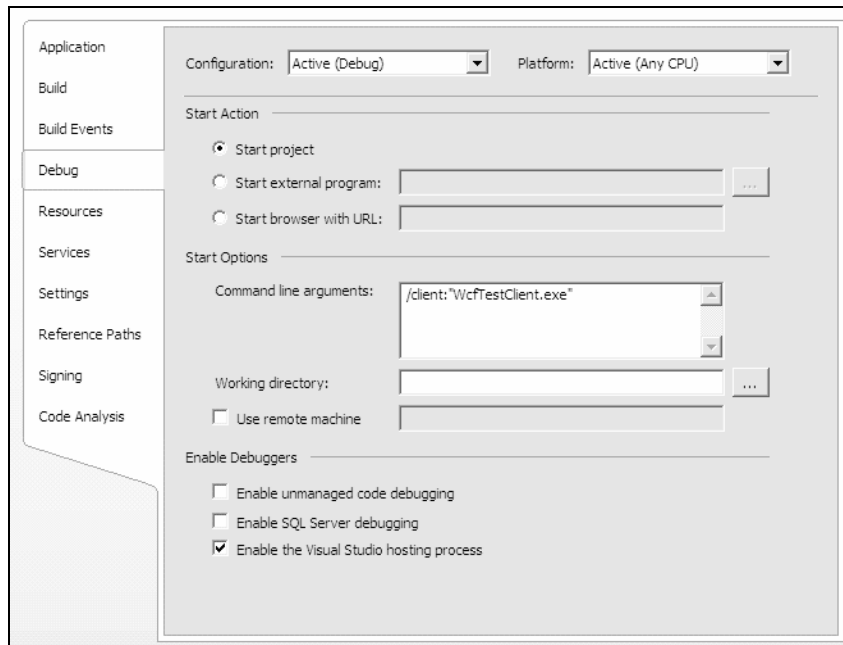


Abbildung 29.8 Konfiguration des WCF-Test-Clients

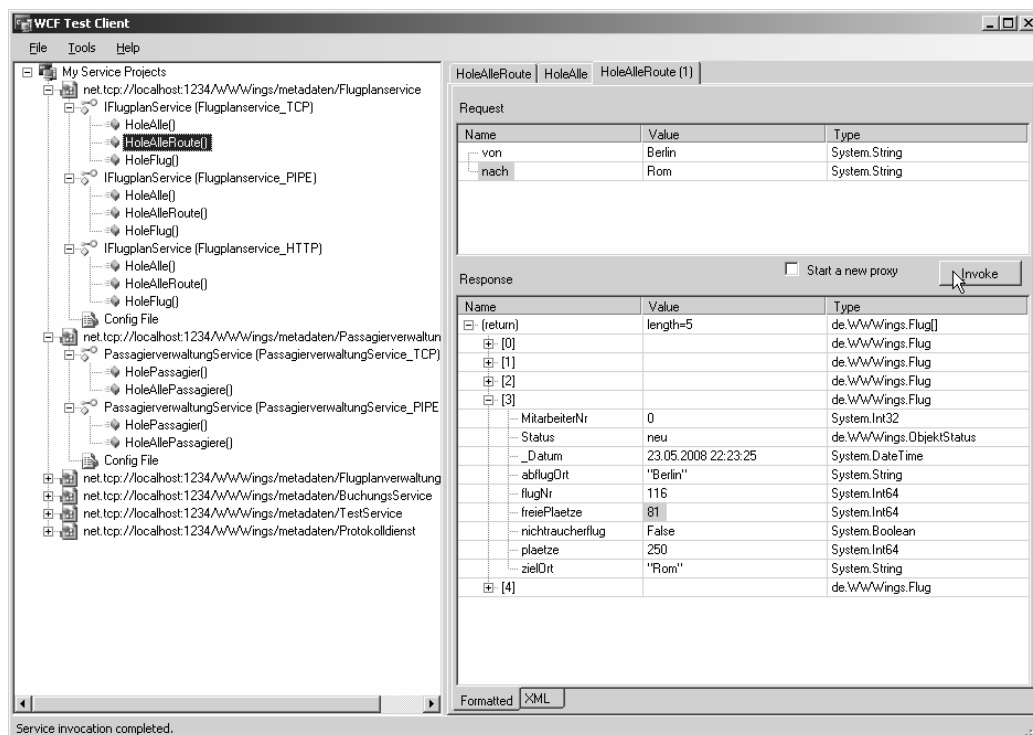


Abbildung 29.9 Testen von WCF-Diensten mit dem WCF-Testclient

TIPP Über die Option *Start a new proxy* kann man den Sitzungszustand kontrollieren. Wenn das Häkchen nicht gesetzt ist, erfolgen alle Anrufe innerhalb eines Fensters in der gleichen Sitzung (sofern der WCF-Dienst Sitzungen unterstützt). Die Konfiguration des Testclients für einen Dienst kann man verändern, indem man in dem Eintrag *Config File* im Baum des Kontextmenüs *Edit with SvcConfigEditor* wählt.

Der Testclient erzeugt Programmcode beim Start, der unter `\Documents\BENUTZER\Local Settings\Application Data\Temp\Test Client Projects\GUID` abgelegt wird.

ACHTUNG Der Testclient unterstützt nicht alle WCF-Funktionen. Nicht unterstützt werden z.B. Dienste mit Duplexkommunikation, Transaktionen, Secure Socket Layer (HTTPS) und WSFederationBinding.

ServiceModel Registration Tool

Das ServiceModel Registration Tool (*ServiceModelReg.exe*) dient der Registrierung der WCF in den IIS. Diese Registrierung wird automatisch vorgenommen, wenn die IIS bei der Installation von .NET 3.0 oder 3.5 bereits installiert sind. Mit dem ServiceModel Registration Tool ist auch eine nachträgliche Registrierung möglich (vgl. *aspnet_regiis.exe* für ASP.NET).

WCF Service Trace Viewer

Der WCF Service Trace Viewer (*SvcTraceViewer.exe*) dient der Analyse von WCF-Protokolldateien und wird später im Zusammenhang mit der Protokollierung noch genauer vorgestellt.

COM+ Service Model Configuration Tool

Das COM+ Service Model Configuration Tool (*ComSvcConfig.exe*) konfiguriert COM+-Schnittstellen, damit diese als WCF-Dienste zur Verfügung gestellt werden können. Dieses Werkzeug wird im vorliegenden Buch nicht besprochen.

WS-AtomicTransaction Configuration Utility

Das WS-AT Configuration Tool (*wsatConfig.exe*) konfiguriert den Microsoft Distributed Transaction Coordinator (MSDTC) für WCF-Transaktionen auf Basis des Standards WS-AtomicTransaction. Dieses Werkzeug wird im vorliegenden Buch nicht besprochen. Ebenso sind die OLE Transactions, die auch mit der WCF möglich sind, kein Thema in diesem Buch.

Erstellung von WCF-Diensten und WCF-Servern

Dieser Abschnitt beschreibt Details zur Implementierung und Bereitstellung von WCF-Diensten und deren Bereitstellung auf WCF-Servern (alias WCF-Hosts). Dazu sind folgende Schritte notwendig:

- Implementierung der WCF-Dienstklasse,
- Erstellung einer Endpunktconfiguration,
- Bereitstellung eines Metadatenendpunktes,
- Bereitstellung oder Implementierung des WCF-Server-Prozesses (WCF-Host).

Dienstklassen

Ein WCF-Dienst wird deklariert, indem eine .NET-Klasse die Annotation `[System.ServiceModel.ServiceContract]` erhält. Die Klasse kann diese Annotation entweder direkt besitzen oder sie durch Implementierung einer .NET-Schnittstelle erlangen, die ihrerseits mit `[ServiceContract]` annotiert ist.

Ein WCF-Dienst erhält Operationen durch die Annotation von Methoden der Klasse mit `[System.ServiceModel.OperationContract]`. Beim Einsatz von Schnittstellen ist die Annotation `[OperationContract]` auf die Methoden der Schnittstelle anzuwenden. `[OperationContract]` darf in einer Klasse oder Schnittstelle nur angewendet werden, wenn dort auch `[ServiceContract]` verwendet wird.

Microsoft hat das Annotationskonzept aus den ASP.NET-Webservices (ASMX) übernommen. Dort wurden anstelle von `[ServiceContract]` die Annotation `[WebService]` und anstelle von `[OperationContract]` die Annotation `[WebMethod]` verwendet. In .NET Remoting gab es solche Annotationen nicht. Aus diesem Grund fällt die Überführung von ASMX zur WCF leichter als von .NET Remoting zur WCF.

Eine mit `[ServiceContract]` annotierte Klasse darf von einer anderen Klasse erben; jedoch darf diese Basis-Klasse nicht selbst mit `[ServiceContract]` annotiert sein. Wenn versucht wird, eine solche Dienstklasse an `ServiceHost` zu übergeben, kommt es zu einer `System.InvalidOperationException`.

Vererbung zwischen WCF-Diensten ist nur möglich auf Basis expliziter Schnittstellen. Dabei darf eine Schnittstelle, die mit `[ServiceContract]` annotiert ist, von einer anderen mit `[ServiceContract]` annotierten Schnittstelle erben und eine Klasse kann die erbende Schnittstelle implementieren.

Das wichtigste Attribut von `[ServiceContract]` ist der Namensraum (Namespace). Wenn dieser nicht gesetzt wird, kommt automatisch der URL `http://tempuri.org` zum Einsatz, wobei *tempuri* für *Temporary Unique Resource Identifier* steht. Ein WCF-Dienst funktioniert zwar mit dieser Adresse; zu Konflikten kann es aber kommen, wenn an einer Anwendung verschiedene WCF-Dienste oder Webservices beteiligt sind, die denselben Namensraum verwenden. Daher sollte man den Namensraum explizit setzen. Grundsätzlich sind für den Namensraum beliebige Bezeichner möglich; die Verwendung von HTTP-Adressen mit DNS-Namen hat sich jedoch wegen der Eindeutigkeit etabliert. Keinesfalls muss der Namensraum der physikalischen Adresse des Dienstes entsprechen.

HINWEIS Folgende Möglichkeiten von .NET sind in Methoden erlaubt, die mit `[OperationContract]` annotiert sind:

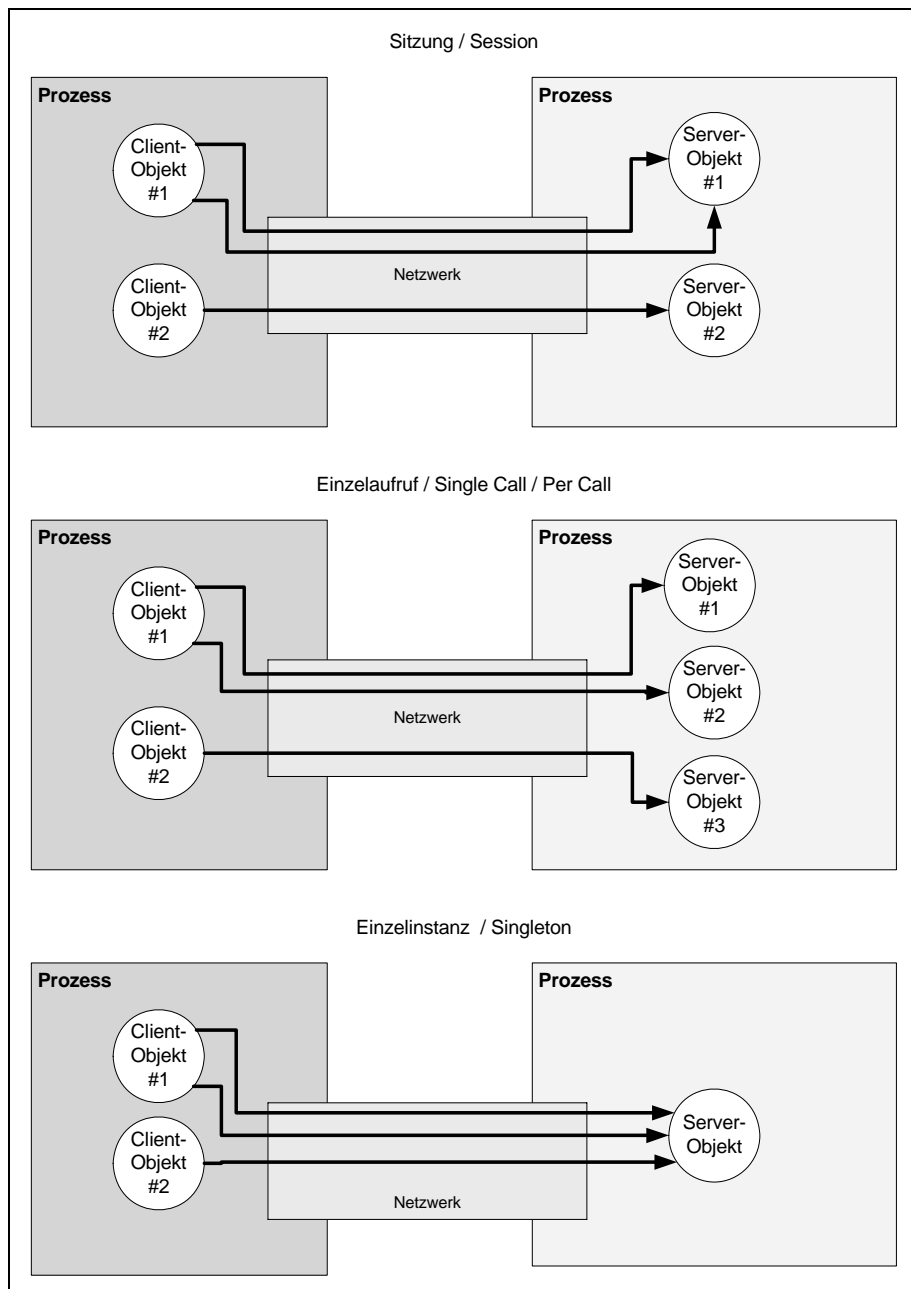
Einfache Datentypen	Ja
Komplexe Datentypen	Ja, wenn serialisierbar
Rückgabewerte	Ja
Ref-Parameter	Ja
Out-Parameter	Ja
Methodenüberladung	Nein!

Instanziierungseigenschaften

Mit `[ServiceBehavior]` legt der Entwickler das Verhalten des Dienstes fest. `[ServiceBehavior]` darf nur auf Klassen, nicht auf Schnittstellen, gesetzt werden.

Die wichtigste Einstellung von `[ServiceBehavior]` ist der `InstanceContextMode`. Dieser bietet drei Möglichkeiten:

- **Single Call** (Wert `PerCall`): Bei jedem Methodenaufruf wird eine neue Objektinstanz erzeugt. Folglich müssen die Objekte zustandslos sein, weil zwei nachfolgende Methodenaufrufe nicht das gleiche Objekt erreichen.
- **Singleton** (Wert `Single`): In diesem Fall existiert nur eine Objektinstanz, bei der alle Clientobjekte ein und dasselbe Serverobjekt verwenden. Ein solches Objekt eignet sich z.B. zum Austausch von Daten zwischen Clients.
- **Sitzungszustand** (Wert `PerSession`): Jeder Client enthält auf Anforderung eine eigene Instanz der Klasse, die lebt, solange der Client mit dem Server in Verbindung steht. Diese Option ist nur für Bindungen verfügbar, die Sitzungszustände unterstützen.

**Abbildung 29.10** Vergleich der Instanziierungsformen

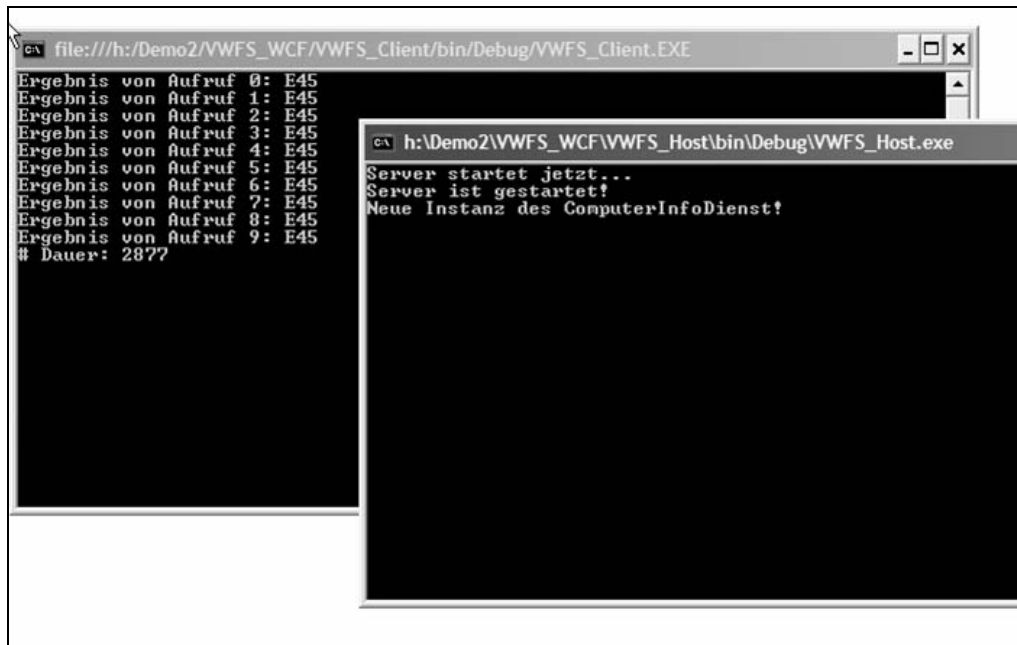


Abbildung 29.11 Beispiel für ein Sitzungsverhalten zwischen Client und Server: eine Serverinstanz pro Client

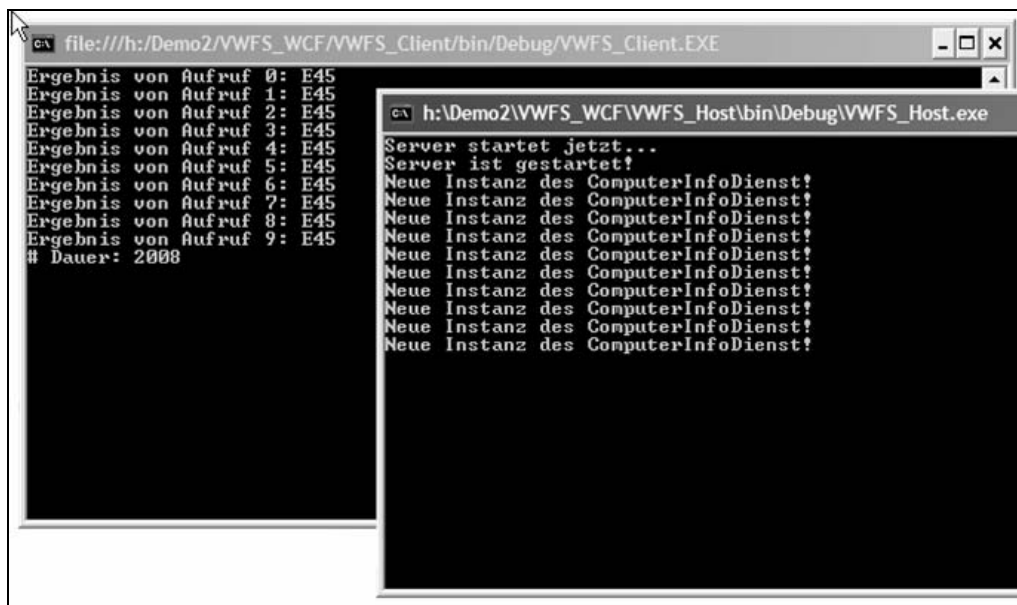


Abbildung 29.12 Beispiel für Single Call (alias PerCall): für jeden Aufruf eine Serverinstanz

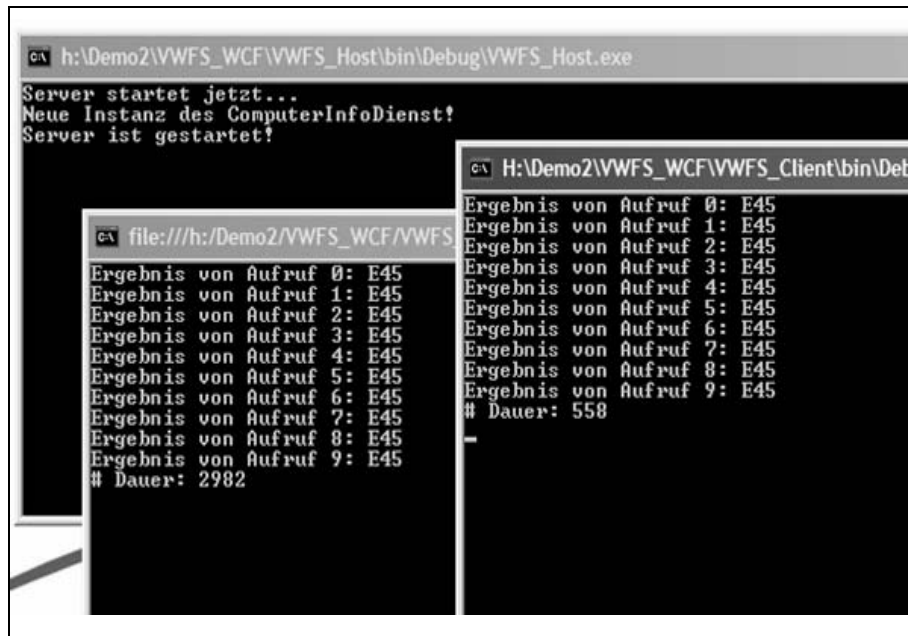


Abbildung 29.13 Beispiel für Singleton: Mehrere Clients nutzen eine einzige Serverinstanz

Beispiel

Die folgenden beiden Listings zeigen

- die Deklaration der Schnittstelle `IFlugplanService`, die als `[ServiceContract]` gekennzeichnet ist, und
- die Deklaration der Klasse `FlugplanService`, welche die Schnittstelle `IFlugplanService` implementiert und durch `[ServiceBehavior]` als Single-Call-Klasse gekennzeichnet ist.

```
[ServiceContract(Namespace = "WWings")]
interface IFlugplanService
{
    [OperationContract]
    de.WWings.FlugMenge HoleAlle();
    [OperationContract]
    de.WWings.FlugMenge HoleAlleRoute(string von, string nach);
    [OperationContract]
    de.WWings.Flug HoleFlug(long FlugNr);
}
```

Listing 29.3 Schnittstelle `IFlugplanService`

```
[ServiceBehavior(InstanceContextMode = InstanceContextMode.PerCall)]
class FlugplanService : IFlugplanService
{
    public FlugplanService()
    {
        Console.WriteLine("Neue Instanz der Fassade FlugplanService!");
    }
    public de.WWWings.Flug HoleFlug(long FlugNr)
    {
        System.Console.WriteLine("HoleFlug wird aufgerufen mit Nr " + FlugNr);
        return de.WWWings.FlugBLManager.HoleFlug(FlugNr);
    }
    public de.WWWings.FlugMenge HoleAlle()
    {
        return de.WWWings.FlugBLManager.HoleAlle();
    }
    public de.WWWings.FlugMenge HoleAlleRoute(string von, string nach)
    {
        return de.WWWings.FlugBLManager.HoleAlle(von, nach);
    }
}
```

Listing 29.4 Klasse FlugplanService

Datenklassen und Serialisierung

Als Datenklassen werden hier Klassen bezeichnet, die von den WCF-Diensten als Parameter oder Rückgabewerte verwendet werden. Alle Datenklassen müssen in der WCF serialisierbar sein, denn eine Übergabe als Objektreferenz (wie in .NET Remoting) ist in der WCF ausgeschlossen.

Die WCF unterstützt drei Serialisierer (vgl. auch Kapitel »NET-Klassenbibliothek 3.5«, Abschnitt »Serialisierung«, im Buch »NET 3.5 Crashkurs« [HS02]):

- XmlSerializer,
- DataContractSerializer,
- NetDataContractSerializer.

Auswahl der Serialisierer

Der DataContractSerializer ist der Standardserialisierer. In WCF-basierten Webservices kommt jedoch der XmlSerializer zum Einsatz. Der Entwickler kann durch Annotationen zwischen den Serialisierern wählen:

Mit dem Attribut XmlSerializerFormat kann man den XML-Serialisierer auswählen und dabei auch zwischen den verschiedenen SOAP-Varianten entscheiden:

```
[ServiceContract, XmlSerializerFormat(
    Style = OperationFormatStyle.Rpc,
    Use = OperationFormatUse.Encoded)]
```

Mit dem Attribut DataContractFormat kann man den DataContractSerializer auswählen. Auch hierbei kann man die SOAP-Art wählen:

```
[ServiceContract, DataContractFormat(Style = OperationFormatStyle.Rpc)]
```

Diese Annotationen sind auf der Dienstklasse oder der Schnittstellendefinition zusammen mit [ServiceContract] anzuwenden.

HINWEIS Es gibt leider keine Annotation zur Auswahl des `NetDataContractSerializer`. Es heißt, Microsoft will die Verwendung dieser Serialisierungsform nicht fördern. Der Beitrag [TOPXML01] zeigt, wie man sich dafür eine eigene Annotation schreiben kann.

Annotationen der Datenklasse

Der `XmlSerializer` nutzt POCO-Serialisierung. Die anderen beiden unterstützen [Serializable] und [DataContract] sowie ab .NET 3.5 Service Pack 1 auch POCO-Serialisierung.

HINWEIS Zum Thema Serialisierung vgl. das Kapitel »NET-Klassenbibliothek 3.5«, Abschnitt »Serialisierung« im Buch »NET 3.5 Crashkurs« [HS02].

Die folgenden Klassen zeigen, dass sich beide Serialisierungsannotationen sowie *Field*-Attribute (Attribute ohne hinterlegten Code) und *Property*-Attribute (Attribute mit hinterlegtem Code) mischen lassen:

```
/// <summary>
/// Daten als Antwort auf einen Ping eines Clients,
/// verwendet [System.Runtime.Serialization.DataContract]
/// </summary>
[System.Runtime.Serialization.DataContract]
public class PingInfo
{
    public PingInfo()
    {
        DBInfo.AnzFluege = de.WWWings.FlugBLManager.AnzFluege();
        DBInfo.AnzBuchungen = de.WWWings.Buchung_BLManager.AnzahlBuchungen();
        DBInfo.AnzPassagiere = de.WWWings.Passagier_BLManager.AnzahlPassagiere();
    }
    [System.Runtime.Serialization.DataMember]
    public DBInfo DBInfo = new DBInfo();
    [System.Runtime.Serialization.DataMember]
    public string ComputerName;
    [System.Runtime.Serialization.DataMember]
    public DateTime ServerTime
    {
        get { return DateTime.Now; }
        set { }
    }
    [System.Runtime.Serialization.DataMember]
    public string ServerIdentity
    {
        get { return System.Security.Principal.WindowsIdentity.GetCurrent().ToString(); }
        set { }
    }
    [System.Runtime.Serialization.DataMember]
    public string Session
```

```
{
    get { return OperationContext.Current.Sitzungs-ID; }
    set { }
}
[System.Runtime.Serialization.DataMember]
public System.Version Version
{
    get { return System.Reflection.Assembly.GetExecutingAssembly().GetName().Version; }
    set { }
}
}
/// <summary>
/// Unterklasse für Statistik aus der Datenbank, verwendet [System.Serializable]
/// </summary>
[System.Serializable]
public class DBInfo
{
    public long AnzFluege;
    public long AnzPassagiere;
    public long AnzBuchungen;
}
```

Listing 29.5 Anwendung der Serialisierungsannotationen

Erstellung einer Endpunktkonfiguration

Bisher wurde für den Dienst nur der Vertrag (Contract) festgelegt. Zum Funktionieren braucht der Dienst aber noch die Adresse und die Bindung. Beides wird üblicherweise in einer Anwendungskonfigurationsdatei gespeichert, damit man diese Einstellungen auch noch zur Betriebszeit der Anwendung festlegen kann. Möglich ist auch eine Festlegung im Programmcode; dies wird in diesem Buch aber nicht gezeigt.

HINWEIS Alle WCF-Einträge in der Anwendungskonfigurationsdatei unterscheiden zwischen Groß- und Kleinschreibung.

TIPP Neu ab Visual Studio 2008 Service Pack 1 ist die Unterstützung für das Refactoring von WCF-Konfigurationselementen: Wenn man den Namen einer WCF-Dienstklasse oder einer WCF-Dienstschnittstelle mit der Funktion *Refactor/Rename* ändert, werden auch die zugehörigen Einträge in den Konfigurationsdateien geändert.

Beispiel

Das folgende Listing zeigt die Deklaration eines WCF-Dienstes mit einem Endpunkt, der die Bindung `NetTcpBinding` verwendet. `NetTcpBinding` greift auf URLs zurück; dafür steht auch die Protokollangabe `net.tcp` in der Adresse:

```
<system.serviceModel>
  <services>
    <service name="de.WWWings.Dienste.FlugplanService">
      <endpoint
        name="FlugplanService"
        address="net.tcp://e01.IT-Visions.local:1234/FlugplanService"
        binding="netTcpBinding">
```



```
contract="de.WWWings.Dienste.IFlugplanService" />  
</service>  
</services>  
</system.serviceModel>
```

Listing 29.6 Konfigurationseinstellungen für einen einfachen WCF-Dienst

WICHTIG Es gibt für die Bindung keine Standardeinstellung. Dieses Attribut muss immer verwendet werden.

Erstellung der Konfiguration mithilfe des Konfigurationsassistenten

Die sechs Abbildungen (Abbildung 29.14 bis Abbildung 29.19) zeigen das Anlegen der obigen Endpunkt-konfiguration für den WCF-Dienst mithilfe des Service Configuration Editors (*SvcConfigEditor.exe*). Zu beachten ist, dass der Editor die Informationen über die verfügbaren Dienstklassen (Abbildung 29.15) immer aus einer kompilierten Assembly bezieht. Eine Integration mit Visual Studio, die im Quellcode nach Dienstklassen sucht, existiert noch nicht.

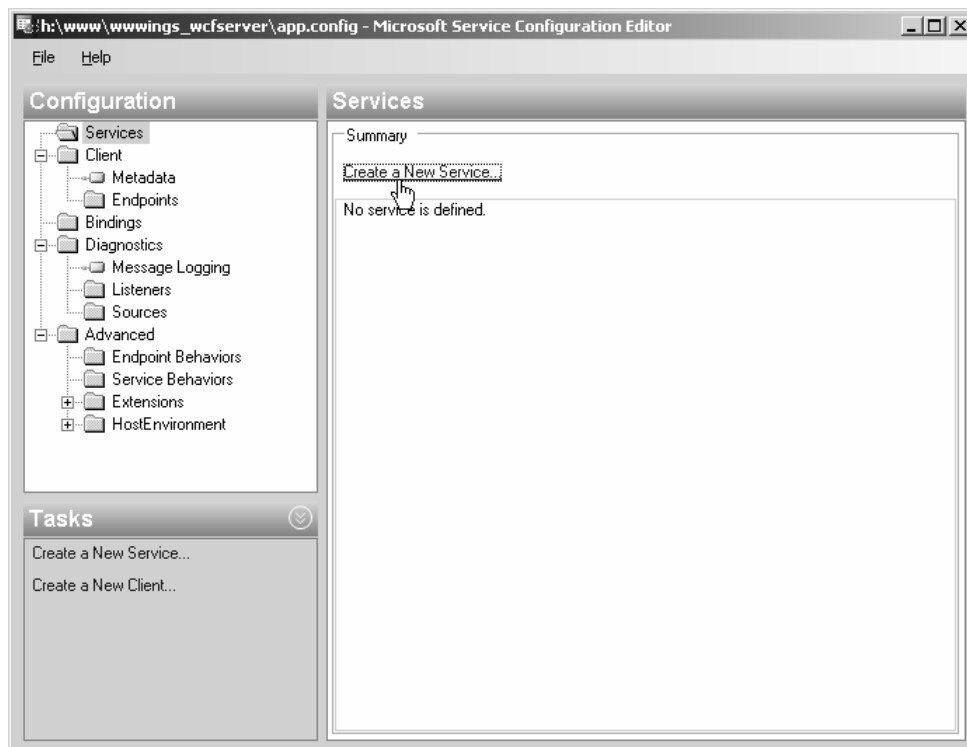


Abbildung 29.14 Anlegen einer neuen WCF-Dienst-Konfiguration

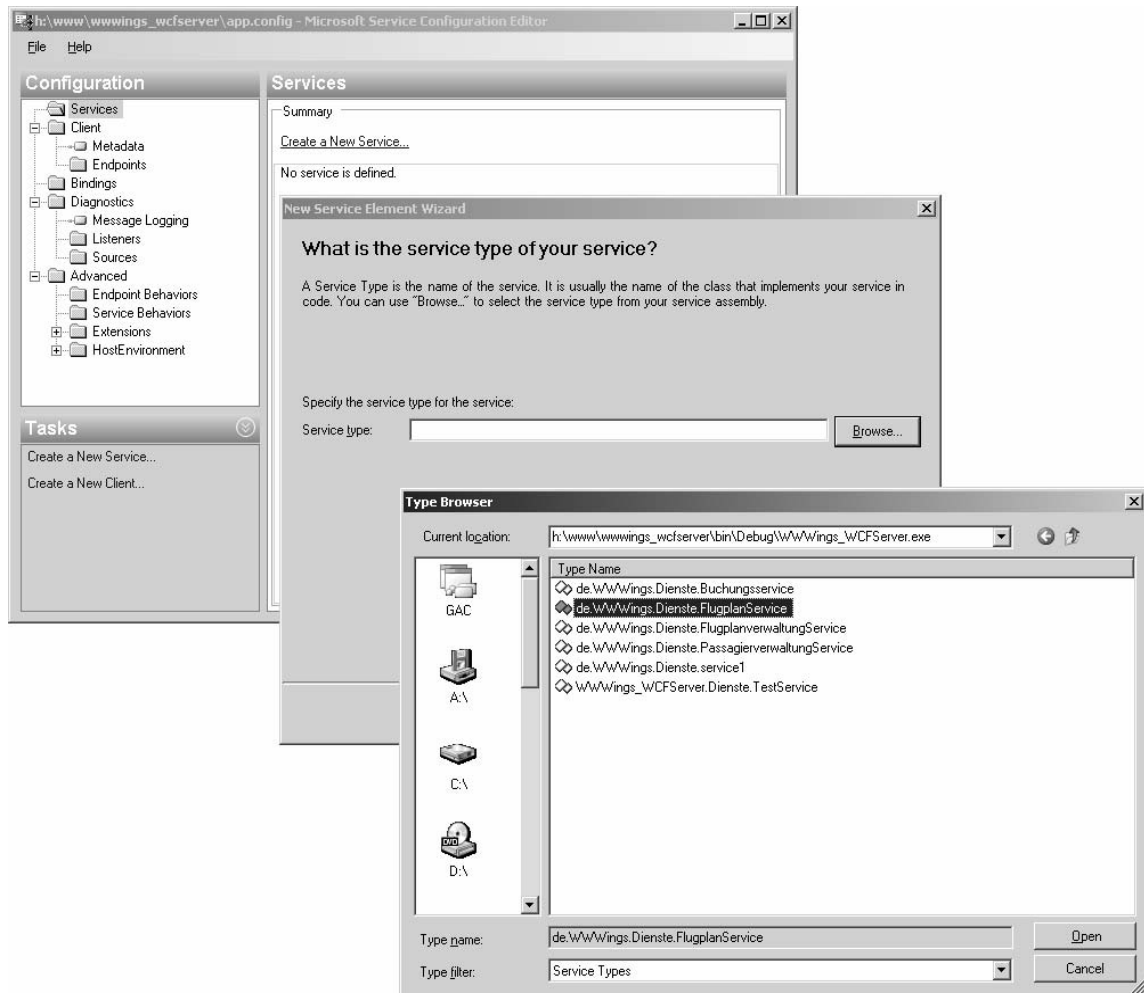


Abbildung 29.15 Auswahl einer mit [ServiceContract] annotierten Klasse



Abbildung 29.16 Auswahl des Vertrags in dem Fall, dass *[ServiceContract]* nicht bei der Klasse, sondern bei einer Schnittstelle verwendet wurde

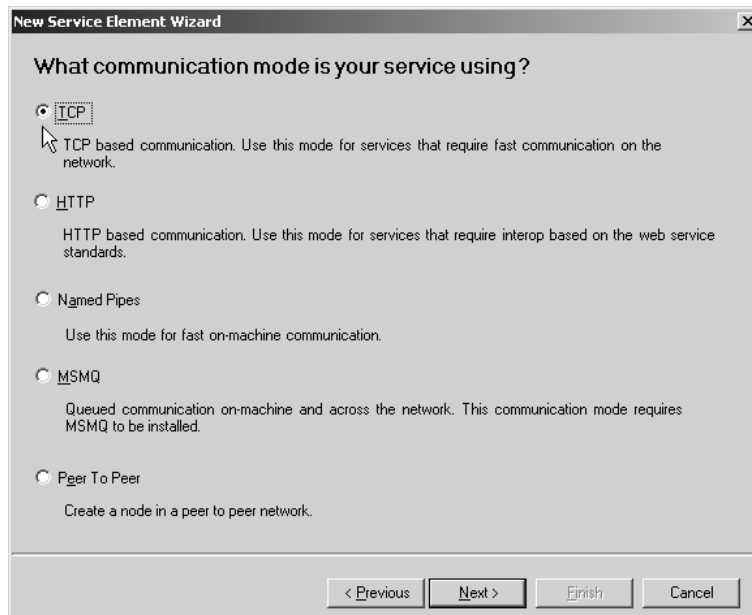


Abbildung 29.17
Auswahl der WCF-Bindung

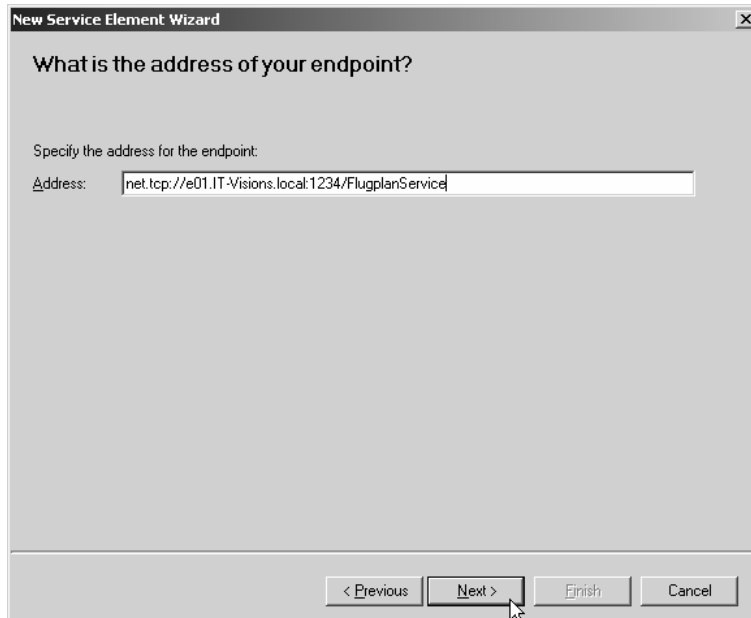


Abbildung 29.18 Festlegung der Adresse des Endpunktes

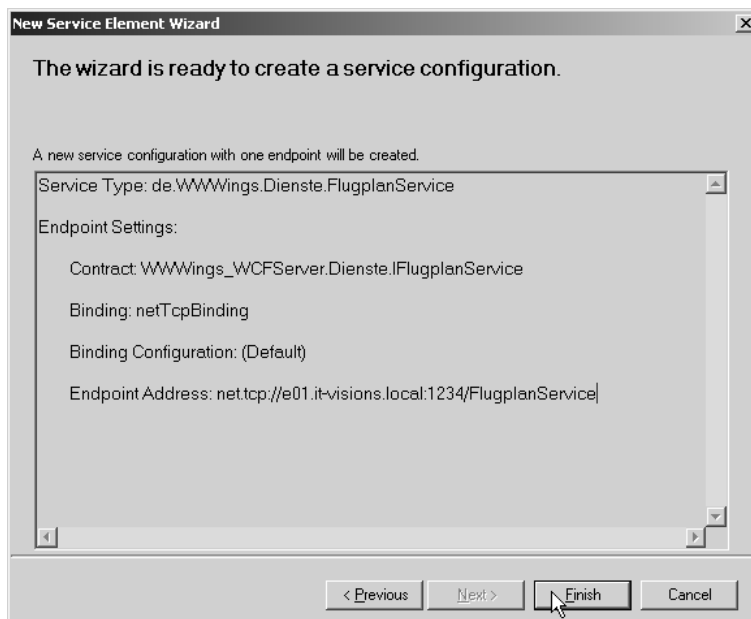


Abbildung 29.19 Zusammenfassung der Ergebnisse des Assistenten, der anschließend einen WCF-Dienst mit einem Endpunkt anlegt

Die im Assistenten vorgenommenen Einstellungen findet man nach Beendigung des Assistenten im Zweig *Services/Endpoints*. Es ist sinnvoll, aber nicht zwingend notwendig, dem Endpunkt einen Namen zu geben. Nachträgliche Änderungen können direkt hier vorgenommen werden. Dabei sind die Optionen (z.B. auch zur Auswahl der Bindung) vielfältiger und detaillierter als im Assistenten. Alternativ zum Weg über den Assistenten kann man Endpunkte auch direkt anlegen, indem man im Kontextmenü des Zweigs *Services* den Eintrag *New Service* wählt und dann im Zweig *Endpoints* den Eintrag *New Service Endpoint*.

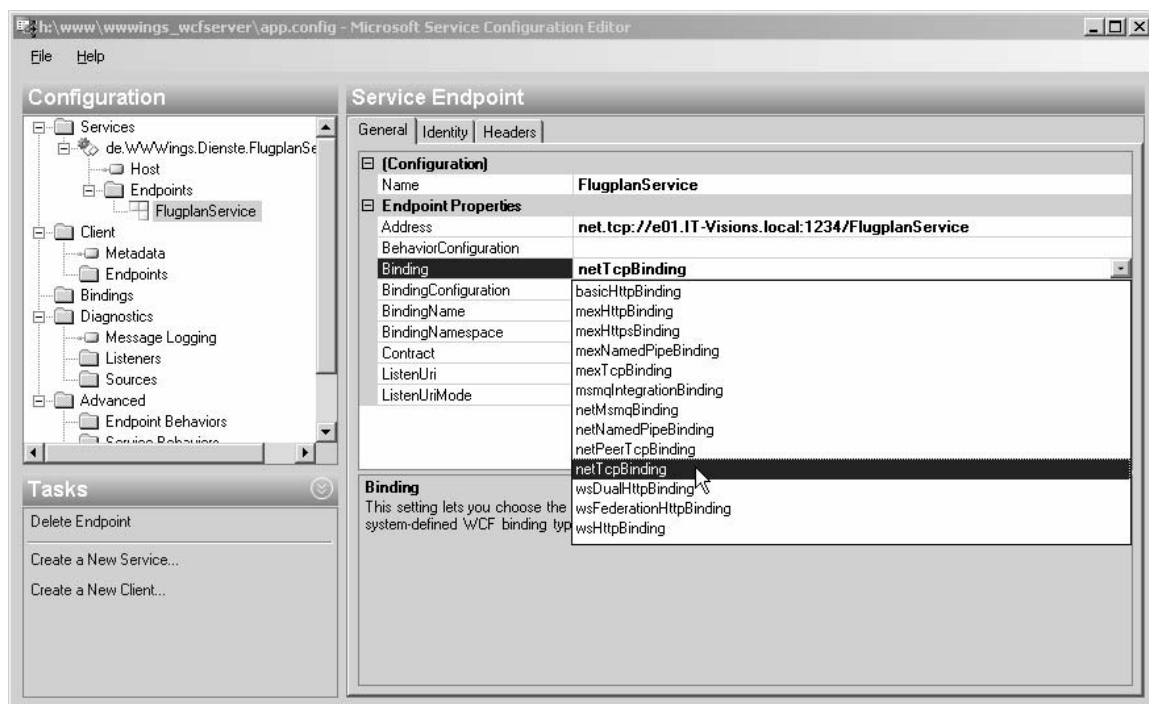


Abbildung 29.20 Detailkonfiguration für einen WCF-Endpoint

HINWEIS Zu einem Dienst kann es mehrere Endpunkte geben. So kann ein WCF-Dienst gleichzeitig sowohl über Named Pipes und TCP als auch über HTTP zur Verfügung gestellt werden.

Codebasierte Konfiguration

Alle Konfigurationseinstellungen kann man auch codebasiert vornehmen. Dazu folgt ein Beispiel für die Erzeugung eines WCF-Dienstes mit einem HTTP-basierten Host. Als Binding kommt `WSHttpBinding` zum Einsatz:

```
// --- Alternative: Code-basierte Konfiguration
// Festlegen der URL
Uri URL = new Uri("http://localhost:84/FlugplanserviceCodebasierteKonfiguration");
// ServiceHost erzeugen
ServiceHost FlugplanService = new ServiceHost(typeof(de.WWWings.Dienste.FlugplanService), URL);
// Endpunkt erzeugen
FlugplanService.AddServiceEndpoint(typeof(de.WWWings.Dienste.IFlugplanService), new WSHttpBinding(),
```

```
FlugplanServiceCodebasierteKonfiguration");
// Dienst starten
FlugplanService.Open();
```

Listing 29.7 Beispiel für die codebasierte Konfiguration in der WCF

HINWEIS Wenn es sowohl eine Konfigurationsdatei als auch eine codebasierte Konfiguration gibt, dann hat die codebasierte Konfiguration Vorrang (Leitsatz: »Code ist König«).

Bereitstellen eines Metadatenendpunktes

Die bisher vorgenommene Dienstkonfiguration reicht aus, um einen Dienst anzusprechen, nicht jedoch, um einen Proxy für einen Dienst zu generieren. Für die Proxygenerierung werden Metadaten benötigt. Während bei ASP.NET-Webservices diese Metadaten durch Anhängen von *?wsdl* an den URL immer automatisch zur Verfügung standen, müssen sie bei WCF explizit aktiviert werden.

WCF verwendet zum Metadaten austausch den Standard WS-Metadata Exchange (MEX). Über MEX werden WSDL-Metadaten gesendet, die von der Klasse `System.ServiceModel.Description.WsdlExporter` automatisch erzeugt werden. Eigene Metadatenformate sind implementierbar auf Grundlage der Basisklasse `System.ServiceModel.Description.MetadataExporter`.

Die Metadaten benötigen einen eigenen Endpunkt, wobei die Portnummer die gleiche sein darf wie bei dem Dienst; der URL kann sich in dem freien Bezeichner nach der Portangabe unterscheiden. Als Bindung stehen `mexTcpBinding`, `mexNamedPipeBinding`, `mexHttpBinding` und `mexHttpsBinding` zur Verfügung. Als Contract ist immer die in WCF vordefinierte Schnittstelle `System.ServiceModel.Description.IMetadataExchange` anzugeben.

Beispiel

Abbildung 29.21 zeigt die Deklaration eines Metadatendienstes für den zuvor deklarierten Dienst `FlugplanService`.

```
<system.serviceModel>
  <services>
    <service name="de.WWWings.Dienste.FlugplanService"
      behaviorConfiguration="Metadaten">
      <!-- Dienst -->
      <endpoint address="net.tcp://localhost:1234/FlugplanService"
        binding="netTcpBinding" bindingConfiguration="" name="FlugplanService"
        contract="de.WWWings.Dienste.IFlugplanService" />
      <!-- METADATEN -->
      <endpoint address="net.tcp://localhost:1234/WWWings/metadaten/Flugplanservice"
        binding="mexTcpBinding" bindingConfiguration="" contract="IMetadataExchange" />
    </service>
  </services>
  <!-- Verhalten -->
  <behaviors>
    <serviceBehaviors>
      <behavior name="Metadaten">
        <serviceMetadata />
      </behavior>
    </serviceBehaviors>
  </behaviors>
</system.serviceModel>
```

Abbildung 29.21 WCF-Dienst mit Metadatendienst

TIPP Die aus ASMX bekannte Syntax zum Bezug von WSDL (*?wsdl*) kann auch in der WCF genutzt werden, indem man dem Metadatenendpunkt eine Adresse gibt, die auf *?wsdl* endet. Für HTTP-Bindungen lässt sich dies auch über `httpGetEnabled="true"` aktivieren.

Konfiguration der Fehlerübermittlung

Ein WCF-Dienst übermittelt standardmäßig keine Details zu aufgetretenen Fehlern an den Client. Man kann diese Funktion jedoch aktivieren, indem man das konfigurierte Dienstverhalten um den Eintrag

```
<serviceDebug includeExceptionDetailInFaults="true"/>
```

ergänzt. Wichtig: Dieses Element muss unterhalb von `<serviceBehaviors>` stehen (vgl. vorheriger Abschnitt).

Hosting (WCF-Server-Prozess / WCF-Anwendungsserver)

WCF-Dienste müssen in einem Windows-Prozess gehostet werden, der eine .NET-Application Domain bereitstellt. Dieser Prozess wird als *WCF-Server* oder *WCF-Service-Host* oder *WCF-Anwendungsserver* bezeichnet.

Ebenso wie bei .NET Remoting stehen als Hostingoptionen die Internet Information Services (IIS), der Anwendungsserver COM+ (alias .NET Enterprise Services), ein Windows-Dienst oder eine einfach von Hand zu startende Konsolenanwendung zur Verfügung. Ab Windows Vista gibt es darüber hinaus einen leichtgewichtigen Standard-Hostprozess mit Namen Windows (Process) Activation Service (WAS) (früherer Name: *Webhost*). Der WAS ist das Basiselement für die IIS Version 7.0 und kann auch in der WCF eingesetzt werden, um HTTP-, TCP-, MSMQ- und Named Pipe-Dienste zu hosten.

Hosting im WCF-Testclient

Der WCF-Testclient wurde bereits im Abschnitt »Werkzeuge« beschrieben.

Hosting in einer Konsolenanwendung

In .NET-Anwendungstypen wie beispielsweise Konsolenanwendungen und Systemdiensten muss eine Instanz von `System.ServiceModel.ServiceHost` für jeden WCF-Dienst explizit instanziiert werden. Im Konstruktor der Klasse `ServiceHost` ist entweder eine Instanz der Dienstklasse (bei Singleton-Diensten) oder der Typ (bei Single-Call-Diensten) anzugeben. Der zweite Parameter bezeichnet die Basisadresse des WCF-Dienstes. Die Basisadresse würde es ermöglichen, die Dienstkonfiguration auf relevante Adressen (zu der Basisadresse) zu beschränken.

Beispiel

Das folgende Beispiel zeigt die Implementierung eines WCF-Servers als Konsolenanwendung. Der Server hostet einen einzigen WCF-Dienst, der wahlweise als Singleton-Objekt oder als Single-Call-Objekt gestartet werden kann. Wichtig ist, dass der Dienst den passenden `InstanceContextMode` in `[ServiceBehavior]` deklariert.

```

class EinfacherWWingsWCFServiceHost
{
    const bool Singleton = false;
    static ServiceHost Host;
    public static void StartService()
    {
        Uri baseAddress = new Uri("net.tcp://localhost:1234/WWings/");
        if (Singleton)
        { // Singleton
            Host = new ServiceHost(new de.WWings.Dienste.FlugplanService(), baseAddress);
        }
        else
        { // Single-Call
            Host = new ServiceHost(typeof(de.WWings.Dienste.FlugplanService), baseAddress);
        }
        Host.Open();
    }
    internal static void StopService()
    {
        if (Host.State != CommunicationState.Closed)
            Host.Close();
    }
}
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("WCF-Server wird gestartet...");
        de.WWings.WCFServer.EinfacherWWingsWCFServiceHost.StartService();
        Console.WriteLine("WCF-Server ist jetzt gestartet!");
        Console.ReadLine();
        Console.WriteLine("WCF-Server wird beendet...");
        de.WWings.WCFServer.EinfacherWWingsWCFServiceHost.StopService();
        Console.WriteLine("WCF-Server ist jetzt beendet!");
    }
}

```

Listing 29.8 Implementierung eines WCF-Service-Hosts für einen WCF-Dienst

ACHTUNG Bei der Instanziierung der Klasse `ServiceHost` erhält man eine `System.InvalidOperationException`, wenn die Annotationen `[ServiceContract]` und `[OperationContract]` falsch verwendet wurden oder keine Endpunkte für die an die `ServiceHost`-Instanz übergebenen Klassen vorhanden sind (Fehlertext: »Service has zero Application (non-infrastructure) Endpoints.«).

Hosting in den Internet Information Services (IIS)

Beim Hosting eines WCF-Dienstes in den IIS (oder WAS) wird der WCF-Dienst in Form einer WCF-Dienstdatei (.svc) auf dem Webserver bereitgestellt. Dabei wurde das Konzept von .asmx-Dateien (vgl. Kapitel 28 zu den ASP.NET-basierten XML-Webservices) übernommen, das heißt, die .svc-Datei selbst enthält nicht mehr als einen Verweis auf eine Klasse, die den eigentlichen Dienst implementiert. Eine .svc-Datei erstellt man in Visual Studio 2008 in einem Webprojekt (Websitemodell oder Webanwendungsmodell, vgl. [HS01]) über die Elementvorlage WCF-Dienst (*WCF Service*).


```
<%@ ServiceHost
Language="C#"
Debug="true"
Service="FlugplanService"
CodeBehind="~/App_Code/FlugplanService.cs" %>
```

Listing 29.9 Standardinhalt einer .svc-Datei

Das obige Beispiel zeigt eine .svc-Datei, die in einem Websitemodell-Projekt angelegt wurde. Die .svc-Datei verweist dabei in der Standardeinstellung auf eine Klasse, die im */App_Code*-Verzeichnis der Webanwendung liegt. Diese Klasse wiederum entspricht exakt der zuvor in diesem Kapitel beschriebenen Dienstklasse (einschließlich der Annotationen [ServiceContract] und [OperationContract] entweder in der Klasse selbst oder in einer Schnittstellendefinition einer Schnittstelle, welche die Klasse implementiert).

WICHTIG In Praxisprojekten sollten Sie nicht die von der Elementvorlage erzeugte Standardstruktur verwenden, sondern die Dienstklasse in eine eigene Bibliothek (DLL-Assembly) auslagern. Wenn Sie dann aus dem Projekt die DLL-Assembly referenzieren, dann reduziert sich der Inhalt der .svc-Datei wie nachfolgend dargestellt:

```
<%@ ServiceHost
Debug="true"
Service="de.WWWings.WCFServices.FlugplanService" %>
```

Listing 29.10 Angepasster Inhalt einer .svc-Datei

TIPP In ASP.NET-basierten Webservices kann man direkt auf die Sitzungsverwaltung von ASP.NET über das Objekt *Page.Session* zugreifen. Die Sitzungsverwaltung (und andere ASP.NET-Funktionen wie z.B. Impersonifizierung) kann in der WCF genutzt werden, wenn man einen WCF-Dienst in den Internet Information Services betreibt und den ASP.NET-Kompatibilitätsmodus in der Konfiguration aktiviert:

```
<system.serviceModel>
<serviceHostingEnvironment aspNetCompatibilityEnabled="false"/>
</system.serviceModel>
```

Listing 29.11 Aktivieren des ASP.NET-Kompatibilitätsmodus

Danach kann man im Programmcode auf *System.Web.HttpContext.Current* und dort z.B. auf das Unterobjekt *Session* zugreifen.

Konfigurationsdateien

Beim Anlegen einer .svc-Datei wird in Visual Studio 2008 automatisch auch ein Eintrag in der *web.config*-Datei mit der Definition eines Endpunkts für das *WSHttpBinding* angelegt. Das folgende Listing zeigt diese Standardkonfiguration für einen WCF-Dienst in den IIS.

Von der Konfigurationsdatei, die für einen Konsolenhost verwendet wurde, unterscheidet sich die Konfigurationsdatei für das IIS-Hosting nur in einem Punkt: Mit dem XML-Attribut *httpGetEnabled="true"* wird die Beschaffung der Metadaten über ein angehängtes *?wsdl* im URL aktiviert (vgl. Kapitel 28 zu den ASP.NET-basierten XML-Webservices).

WICHTIG Die IIS 5.x und 6.0 unterstützen nur HTTP-basierte Bindungen. Der Versuch, eine TCP-basierte Bindung zu konfigurieren, führt zum Fehler »The protocol 'net.tcp' is not supported.«. TCP und andere Protokolle sind erst möglich im WAS bzw. in den IIS 7.0 unter Windows Vista und Windows Server 2008.

```
<configuration>
  <appSettings/>
  <connectionStrings/>
  <system.serviceModel>
    <services>
      <service name="de.WWWings.Dienste.FlugplanService" behaviorConfiguration="FlugplanServiceBehaviors">
        <endpoint address="http://localhost:8082/WebUI_CSVB/Webservice/Flugplanservice.svc"
          binding="basicHttpBinding" bindingConfiguration="" contract="de.WWWings.Dienste.IFlugplanService" />
        <endpoint contract="IMetadataExchange" binding="mexHttpBinding" address="metadaten" />
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="FlugplanServiceBehaviors" >
          <serviceMetadata httpGetEnabled="true" />
          <serviceDebug includeExceptionDetailInFaults="true"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

Abbildung 29.22 Konfiguration für einen WCF-Dienst in den IIS

TIPP Wenn Sie WCF-Dienste in Unterverzeichnissen ablegen, reicht es, eine lokale *web.config*-Datei anzulegen und dort die WCF-Endpunktkonfiguration einzutragen. Dies vermeidet das »Zumüllen« der Anwendungskonfigurationsdateien, das man bei anderen WCF-Hosts vorfindet.

Abruf der Metadaten

Je nach Einstellung des Dienstverhaltens sind die Metadaten über das Anhängen von *?wsdl* oder des im Endpunkt genannten Begriffs (hier: Metadaten) zu beziehen. Nur die erste Variante funktioniert aus dem Webbrowser heraus. Die zweite Variante kann aber von den WCF-Proxygeneratoren (WCF-Erweiterung für Visual Studio bzw. *svcutil.exe*) verwendet werden.

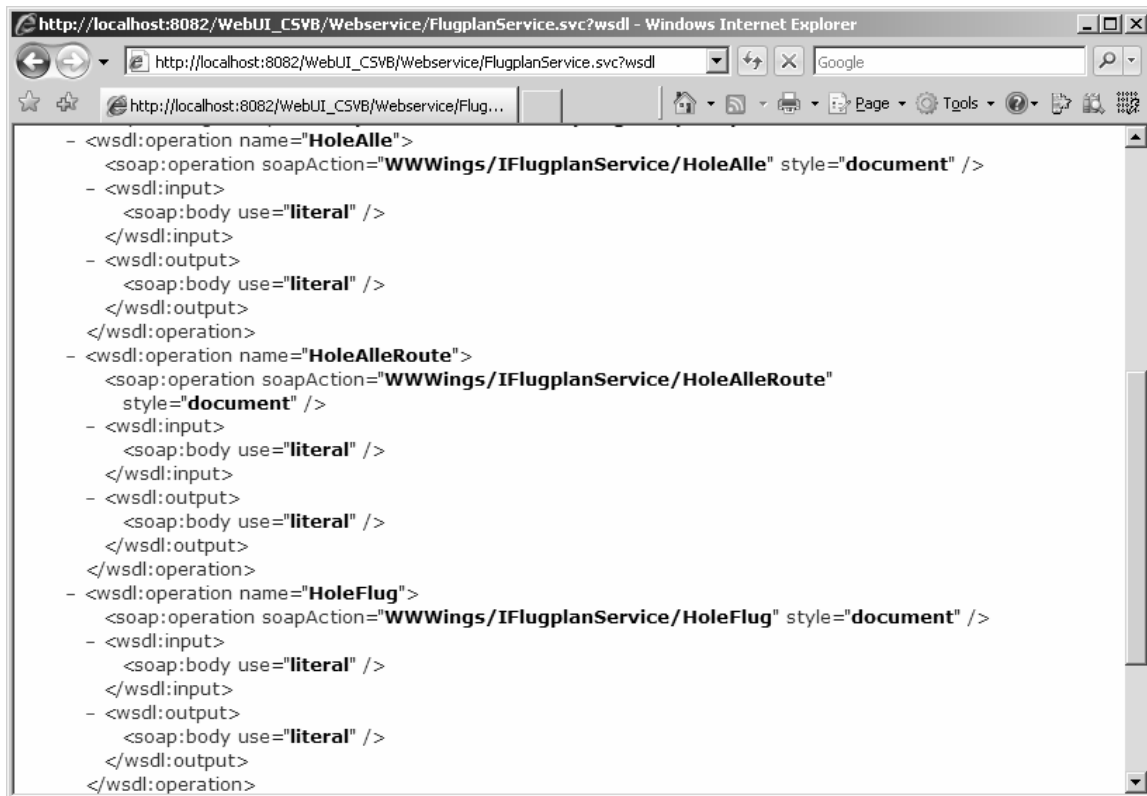


Abbildung 29.23 WSDL des WCF-Dienstes

Generierung eines IIS-basierten WCF-Hosts in Visual Studio 2008

Visual Studio 2008 bietet ab Service Pack 1 für WCF-Dienste einen Assistenten, der einen IIS-basierten WCF-Host erzeugt. Diese Funktion ist erreichbar über das Menü *Build/Publish* oder *Publish* im Kontextmenü des Projekts im Projektmappen-Explorer – aber nur für WCF-Dienstprojekte! Der Assistent zeigt einen Dialog (Abbildung 29.24) und erzeugt dann am angegebenen Ziel eine *.svc*-Datei und eine zugehörige *web.config*-Datei. Außerdem wird das Dienstprojekt als Assembly in das *bin*-Verzeichnis gelegt.

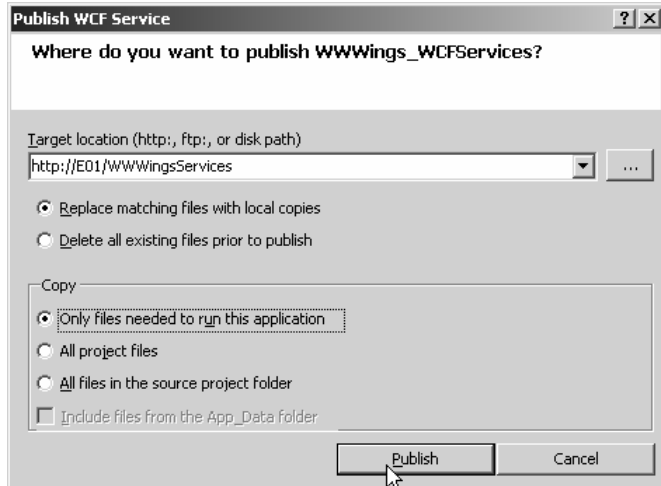


Abbildung 29.24 Dialog zum Veröffentlichen eines WCF-Dienstes in Visual Studio 2008 Service Pack 1

HINWEIS Statt des Assistenten für einen IIS-basierten WCF-Host würde man sich als WCF-Entwickler lieber einen universellen Hostassistenten wünschen, der auch Konsolen- und Windows-Systemdienst-basierte WCF-Hosts erzeugen kann. Zumindest in der bis zum Reaktionsschluss vorliegenden Version des Service Pack 1 von Visual Studio 2008 gab es einen solchen Assistenten aber nicht.

Test des Dienstes

In Visual Studio 2008 kann man – anders als in Visual Studio Version 2005 – auch die Funktion *In Browser anzeigen* (*View in Browser*) auf einer WCF-Datei aufrufen. Dadurch zeigt sich eine Informationsseite. Es gibt hier aber – anders als bei ASP.NET-basierten Webservices – keinen Testclient.

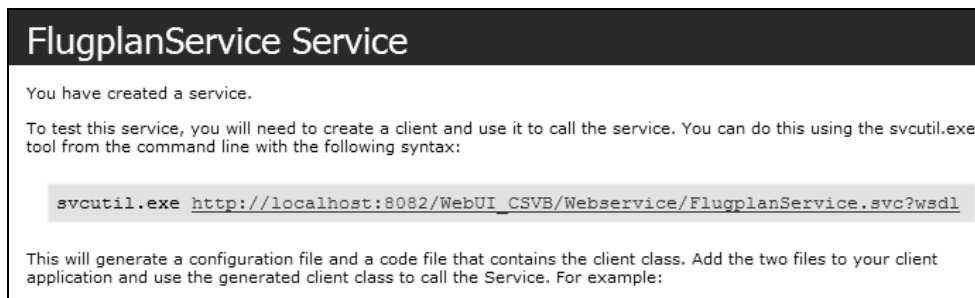


Abbildung 29.25 Wenn der Dienst erfolgreich angelegt wurde, erscheint eine solche Meldung

Best Practices: Selbsttest

Nach dem Start eines WCF-Servers empfiehlt es sich immer, einen Selbsttest durchzuführen. Dabei implementiert man eine Methode, die möglichst viele Daten von der Datenbank abrufen und prüft, ob dies erfolgreich ausführbar ist. Diese Methode ruft man sowohl assemblyintern als auch extern (das heißt über WCF) auf. In

der World Wide Wings-Anwendung realisiert der WCF-Server die Klasse `TestService` mit einer Methode `Ping()`, die unter anderem ein `DBInfo`-Objekt mit der Anzahl der Flüge, Passagiere und Buchungen in der Datenbank zurückgibt. Die `Ping()`-Methode wird nach der Instanziierung des `ServiceHost`-Objekts sowohl intern als auch extern (sowohl über Named Pipes als auch über TCP) aufgerufen. Dazu besitzt der WCF-Server selbst wieder eine `Service`-Referenz auf den Dienst `TestService`.

```

Shortcut to WWWings_WCFServer.exe
WCF-Server wird gestartet...
Neue Instanz der Fassade FlugplanService!
Neue Instanz der Fassade FlugplanverwaltungService!
Neue Instanz der Fassade Buchungsservice
Neue Instanz der Fassade PassagierverwaltungService!
Starte Dienst FlugplanService
- Endpunkt: net.tcp://localhost:1234/WWWings/FlugplanService (NetTcpBinding)
- Endpunkt: net.pipe://localhost/WWWings/Pipes/FlugplanService (NetNamedPipeBinding)
- Endpunkt: net.tcp://localhost:1234/WWWings/metadaten/FlugplanService (MetadataExchangeTcpBinding)
- Endpunkt: http://localhost/Webservices/FlugplanService.svc (BasicHttpBinding)
Starte Dienst FlugplanverwaltungService
- Endpunkt: net.tcp://localhost:1234/WWWings/FlugplanverwaltungService (NetTcpBinding)
- Endpunkt: net.pipe://localhost/WWWings/Pipes/FlugplanverwaltungService (NetNamedPipeBinding)
- Endpunkt: net.tcp://localhost:1234/WWWings/metadaten/FlugplanverwaltungService (MetadataExchangeTcpBinding)
Starte Dienst Buchungsservice
- Endpunkt: net.tcp://localhost:1234/WWWings/Buchungsservice (NetTcpBinding)
- Endpunkt: net.pipe://localhost/WWWings/Pipes/Buchungsservice (NetNamedPipeBinding)
- Endpunkt: net.tcp://localhost:1234/WWWings/metadaten/Buchungsservice (MetadataExchangeTcpBinding)
Starte Dienst PassagierverwaltungService
- Endpunkt: net.tcp://localhost:1234/WWWings/PassagierverwaltungService (NetTcpBinding)
- Endpunkt: net.pipe://localhost/WWWings/Pipes/PassagierverwaltungService (NetNamedPipeBinding)
- Endpunkt: net.tcp://localhost:1234/WWWings/metadaten/PassagierverwaltungService (MetadataExchangeTcpBinding)
Starte Dienst TestService
- Endpunkt: net.tcp://localhost:1234/WWWings/TestService (NetTcpBinding)
- Endpunkt: net.pipe://localhost/WWWings/Pipes/TestService (NetNamedPipeBinding)
- Endpunkt: net.tcp://localhost:1234/WWWings/metadaten/TestService (MetadataExchangeTcpBinding)
WCF-Server ist jetzt gestartet!
Interner Selbsttest (ohne Protokollstack): OK (232/165/22)
Externer Selbsttest (mit TCP-Protokollstack): OK (232/165/22)
Externer Selbsttest (mit Named Pipes): OK (232/165/22)

```

Abbildung 29.26 Der WCF-Server führt beim Starten drei Selbsttests durch

Erstellung eines WCF-Clients

Dieser Abschnitt beschreibt das Erstellen eines WCF-Clients. Als Client für einen WCF-Dienst kann grundsätzlich jeder .NET-Anwendungstyp zum Einsatz kommen.

Vorüberlegungen

Der Client benötigt von dem Server entweder

- die Assembly, welche die Dienste und die Datenklasse implementiert, oder
- Proxyklassen für Dienste und Datenklassen.

HINWEIS Im Konzept der serviceorientierten Architekturen (SOA) wird das zweite Modell klar bevorzugt aufgrund der Entkopplung von Client und Server und der dadurch möglichen Plattformunabhängigkeit der Kommunikation. In diesem Buch wird auch aus Platzgründen nur das zweite Modell verwendet.

Erstellen eines Proxys

Der Client benötigt zum Zugriff auf den WCF-Dienst einen Proxy und Konfigurationseinträge. Diese können auf verschieden Art erstellt werden:

- mit dem Kommandozeilenwerkzeug *SvcUtil.exe* aus dem Windows SDK 6.0 oder
- mit Visual Studio Version 2005, sofern die WCF-Erweiterung installiert ist, oder
- mit Visual Studio 2008 (ohne Zusatzinstallation).

Der Visual Studio 2008-Assistent zur Proxygenerierung ist über den Kontextmenüeintrag *Add Service Reference* erreichbar. Dieser Eintrag ist nicht zu verwechseln mit dem Eintrag *Add Web Reference*, der einen Proxy für klassische Webservices erzeugt und daher nur einsetzbar ist, wenn der WCF-Dienst via HTTP bereitgestellt und mit SOAP serialisiert wird.

Grundsätzlich kann man über *Add Service Reference* alle WCF-Dienste ansprechen. In das Dialogfenster ist die Adresse des Metadatendienstes einzugeben. Darüber bezieht Visual Studio dann die ABC-Daten. Visual Studio meldet im Ausgabefenster:

```
Attempting to download metadata from 'net.tcp://e01.it-visions.local:1234/WWWings/metadaten/FlugplanService' using WS-Metadata Exchange.  
Generating files...
```

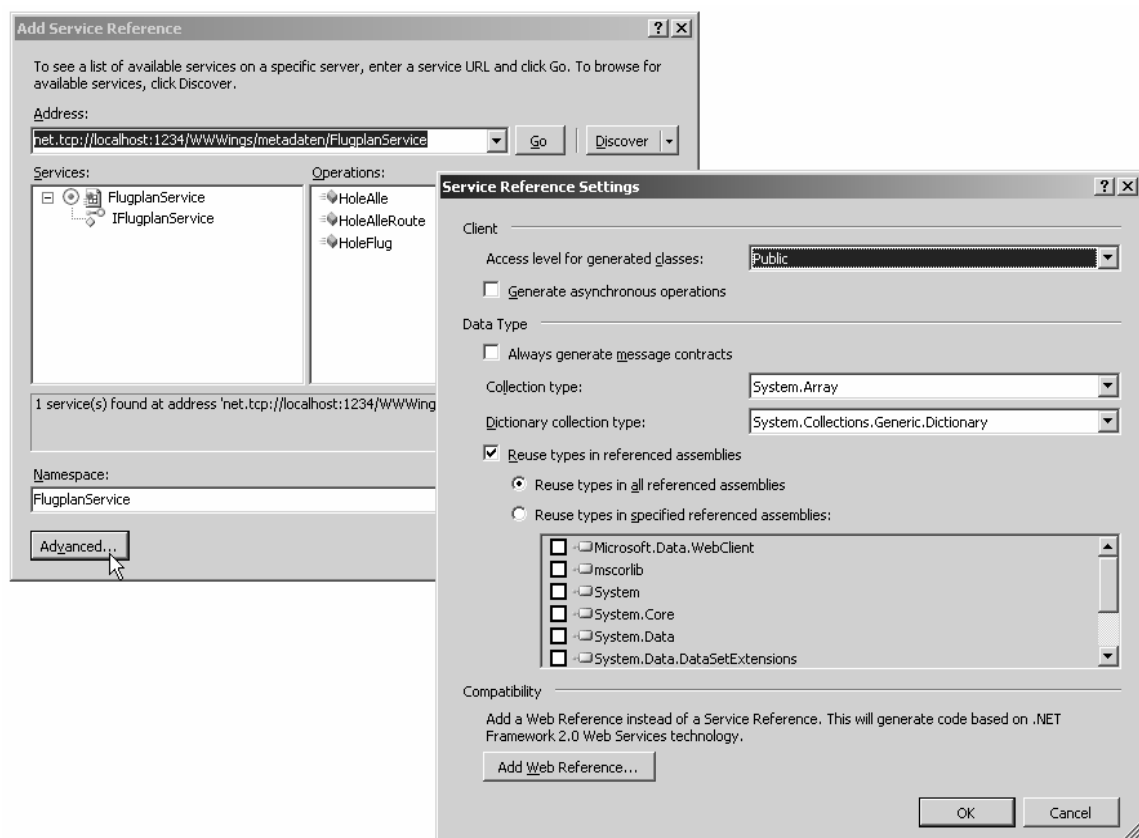


Abbildung 29.27 Erstellen einer Referenz zu einem WCF-Dienst in Visual Studio 2008

Im Ergebnis werden folgende Elemente erzeugt:

- Eine Programmcode-Datei mit einer Klasse für den Dienstproxy und gegebenenfalls weiteren Datenklassen, wenn der Dienst komplexe Datentypen austauscht, sowie Rückruffschnittstellen, wenn der Dienst Duplexkommunikation anbietet (Duplex wird später in diesem Kapitel besprochen). Die generierte Proxyklasse verwendet als Basisklasse `System.ServiceModel.ClientBase`. Die Datenklassen erben von `System.MarshalByRefObject`, das heißt, sie werden immer serialisiert.
- Eine `.map`-Datei, in der Visual Studio den Standort der Metadaten und die Adresse der Dienstendpunkte speichert.
- Einträge in der Anwendungskonfigurationsdatei, die für die Laufzeit des Clients die Adresse der Dienstendpunkte, die Bindung und Kommunikationsparameter wie Timeoutzeiten festlegen.
- Assemblyreferenzen zu `System.ServiceModel.dll` und `System.Runtime.Serialization.dll`.

TIPP

Wenn sich die Schnittstelle eines Dienstes ändert, muss man den Proxy aktualisieren. Zur Aktualisierung des Proxys muss man die Dienstreferenz nicht löschen, sondern man kann die Aktualisierung über den Kontextmenüeintrag *Update Service Reference* vornehmen.

Konfigurationseinstellungen

Das folgende Listing zeigt ein Beispiel für durch den Proxygenerator angelegte Konfigurationseinstellungen:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <bindings>
      <netTcpBinding>
        <binding name="FlugplanService" closeTimeout="00:01:00" openTimeout="00:01:00"
          receiveTimeout="00:10:00" sendTimeout="00:01:00" transactionFlow="false"
          transferMode="Buffered" transactionProtocol="OleTransactions"
          hostNameComparisonMode="StrongWildcard" listenBacklog="10" maxBufferPoolSize="524288"
          maxBufferSize="65536" maxConnections="10" maxReceivedMessageSize="65536">
          <readerQuotas maxDepth="32" maxStringContentLength="8192" maxArrayLength="16384"
            maxBytesPerRead="4096" maxNameTableCharCount="16384" />
          <reliableSession ordered="true" inactivityTimeout="00:10:00"
            enabled="false" />
          <security mode="Transport">
            <transport clientCredentialType="Windows" protectionLevel="EncryptAndSign" />
            <message clientCredentialType="Windows" />
          </security>
        </binding>
      </netTcpBinding>
    </bindings>
    <client>
      <endpoint address="net.tcp://localhost:1234/WWings/FlugplanService"
        binding="netTcpBinding" contract="WCFClient.WCFServer.IFlugplanService"
        name="FlugplanService_TCP" />
      <endpoint address="net.tcp://localhost:1234/FlugplanService"
        binding="netTcpBinding" bindingConfiguration="FlugplanService"
        contract="WCFClient.WCFServer.IFlugplanService" name="FlugplanService">
        <identity>
          <userPrincipalName value="hs@IT-Visions.local" />
        </identity>
      </endpoint>
    </client>
  </system.serviceModel>
</configuration>
```

```

    </identity>
  </endpoint>
</client>
</system.serviceModel>
</configuration>

```

Listing 29.12 Die durch den Assistenten generierte Konfiguration für einen WCF-Client

Die meisten Einstellungen davon braucht man aber nicht, denn sie geben nur die Standardeinstellungen wieder. In dem konkreten Fall würde auch die nachstehende Minimalversion reichen. Der Assistent macht es durch die umfangreicheren Einträge allerdings einfacher, die Konfiguration anzupassen.

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <client>
      <endpoint address="net.tcp://localhost:1234/FlugplanService"
        binding="netTcpBinding"
        contract="WCFClient.WCFServer.IFlugplanService" name="FlugplanService">
      </endpoint>
    </client>
  </system.serviceModel>
</configuration>

```

Listing 29.13 Reduzierte Konfiguration für einen WCF-Client

Auswahl der Konfiguration

Ein WCF-Dienst kann mehrere Endpunkte besitzen, also beispielsweise sowohl per Named Pipe als auch per TCP erreichbar sein. Bei der Generierung des Proxys werden dann mehrere alternative Clientkonfigurationen erstellt:

```

<endpoint address="net.tcp://localhost:1234/WWings/Flugplanservice"
  binding="netTcpBinding" bindingConfiguration="Flugplanservice_TCP"
  contract="de.WWings.Flugplanservice.IFlugplanService" name="Flugplanservice_TCP" />
<endpoint address="net.pipe://localhost/WWings/Pipes/Flugplanservice"
  binding="netNamedPipeBinding" bindingConfiguration="Flugplanservice_PIPE"
  contract="de.WWings.Flugplanservice.IFlugplanService" name="Flugplanservice_PIPE" />

```

Der Entwickler muss entweder innerhalb der Anwendungskonfigurationsdatei für jeden Typ die Konfiguration auf einen Endpunkt reduzieren oder aber im Programmcode den Namen der gewünschten Konfiguration bei der Instanziierung des Proxys angeben:

```

Flugplanservice.FlugplanServiceClient c = new
Flugplanservice.FlugplanServiceClient("Flugplanservice_TCP");

```

Aufruf eines WCF-Servers

Wenn man eine Proxyklasse erzeugt hat, kann man den WCF-Dienst so ansprechen, als wäre er eine lokale Klasse:


```
WCFServer.FlugplanServiceClient c = new WCFServer.FlugplanServiceClient();
WCFServer.Flug f = c.HoleFlug(101);
Console.WriteLine(f.flugNr + " fliegt von " + f.abflugOrt + " nach " + f.zielOrt);
c.close();
```

Listing 29.14 Aufruf einer Methode in einem WCF-Dienst [/ConsoleUI/WCFClient.cs]

WICHTIG Anders als bei ASMX-Webservices ist bei der WCF der abschließende `close()`-Aufruf wichtig, da einige WCF-Bindungen (z.B. TCP) eine ständige Verbindung zwischen Client und Server herstellen. Ohne das explizite Schließen der Verbindung würde diese bis zum Ende der auf dem Server eingestellten Timeoutzeit offen bleiben. Die ständige Verbindung wird nicht beim Instanzieren des Proxys, sondern erst beim ersten Aufruf hergestellt.

Eigenschaften der Proxyklassen

Bei den erzeugten Proxyklassen für die Datenklasse ist zu beachten:

- Die Proxyklasse besitzt nicht die Methode der Originalklasse.
- Die Proxyklasse besitzt nicht die in den Property-Attributen der Originalklasse enthaltene Logik.
- Die Proxyklasse besitzt keinen Hinweis auf die von der Originalklasse explizit implementierten Schnittstellen.
- Die Proxyklasse basiert nicht auf von der Originalklasse deklarierten Annotationen.
- Listen werden als Array abgebildet (Ausnahme: Silverlight liefert `ObservableCollection<T>`).

Die erzeugte Proxyklasse implementiert aber die Schnittstelle `INotifyPropertyChanged` für das Datenbinding.

Aktualisieren des Clients

Client und Server müssen in der Konfigurationsdatei die gleichen Bindinginformationen besitzen. Wenn man z.B. einen Server von `WSHttpBinding` auf `BasicHttpBinding` umstellt, aber die Konfigurationsdatei des Clients nicht aktualisiert, kommt es zum Fehler: »Content Type text/xml; charset=utf-8 was not supported by service http://e45/WCFService2/Service.svc. The client and service bindings may be mismatched.«. Für die Aktualisierung des Clients kann man in Visual Studio die Funktion *Update Service Reference* im Kontextmenü einer bestehenden Service Reference nutzen. Man muss die Service Reference nicht löschen und neu anlegen.

HINWEIS Wenn sich die Adresse eines Metadatenendpunktes ändert, können Sie nicht die Funktion *Update Service Reference* verwenden. In diesem Fall nutzen Sie die Funktion *Configure Service Reference* im gleichen Kontextmenü eines bestehenden *Service Reference*-Eintrags im Projektmappen-Explorer.

Steuerung der Proxyklasse

Visual Studio bietet für die Proxyklasse natürlich Eingabeunterstützung per IntelliSense an. Bei der Nutzung von IntelliSense fällt auf, dass die Proxyklasse `FlugplanServiceClient` mehr Mitglieder als nur die von dem WCF-Dienst bereitgestellten Methoden `HoleFlug(Nummer)`, `HoleAlle()` und `HoleAlleRoute(von, nach)` anbietet.

Dies sind die Attribute und Methoden der Basisklasse `System.ServiceModel.ClientBase`. Sie dienen der programmgesteuerten Konfiguration der Proxyklasse. Hier kann man unter anderem folgende Eigenschaften auslesen oder steuern:

- `Endpoint.Address` bietet die Möglichkeit, vor dem Aufruf der Methode den URL des WCF-Servers zu setzen. Der `WebServiceClient` kann also den anzusprechenden Server zur Laufzeit auswählen, wenn mehrere Server den gleichen Dienst anbieten.
- Über `Endpoint.Binding.SendTimeout` und `Endpoint.Binding.ReceiveTimeout` legt man fest, wie lange sich der Proxy beim Senden geduldet bzw. wie lange er maximal auf eine Antwort wartet. Erfolgt die Reaktion nicht in der gegebenen Zeit, erzeugt die WCF-Laufzeitumgebung einen Fehler vom Typ `System.TimeoutException`.
- Im Attribut `State` findet man den aktuellen Zustand des Proxys.

Im folgenden Beispiel wird der URL explizit angegeben und dem WCF-Server wird nur eine Sekunde Zeit zum Antworten gegeben:

```
WCFServer.FlugplanServiceClient c = new WCFServer.FlugplanServiceClient();
c.Endpoint.Address = new System.ServiceModel.EndpointAddress("net.tcp://e01.it-
visions.local:1234/FlugplanService");
c.Endpoint.Binding.ReceiveTimeout = new TimeSpan(0, 0, 1);
WCFServer.Flug f = c.HoleFlug(101);
```

Listing 29.15 Aufruf einer Webmethode [/ConsoleUI/WCFClient.cs]

Erweitern der generierten Klassen

Die generierten WCF-Clientklassen sind alle als partielle Klassen angelegt, bestehen aber nur aus einem Teil. Dies gibt dem Nutzer der Klassen die Möglichkeit, eigene Ergänzungen hinzuzufügen.

Beispiel

In dem folgenden Beispiel erhält die generierte Datenklasse `Passagier` ein zusätzliches Property-Attribut, das den Vor- und Nachnamen zusammensetzt.

HINWEIS Die serverseitige Implementierung besitzt ein solches Property-Attribut bereits. In der Welt der vertragsbasierten serviceorientierten Kommunikation stehen solche Attribute im Client aber nicht zur Verfügung, da sie auf Programmcode basieren, und dieser wird nicht an den Client übergeben.

```
/// <summary>
/// Manuell erstellte ergänzende Implementierung für Passagier-Klasse
/// </summary>
public partial class Passagier
{
    public string GanzerName
    {
        get { return this._Vorname + " " + this._Nachname; }
    }
}
```

Listing 29.16 Ergänzung zu einer Proxydatenklasse

REST-basierte WCF-Dienste

In der ersten Version der WCF gab es nur SOAP-basierte Webservices. Ab .NET 3.5 werden auch REST-basierte Webservices unterstützt.

HINWEIS REST steht für »Representational State Transfer« und ist ein technischer Begriff, der das Kernprinzip des HTTP-Protokolls bezeichnet. Per HTTP werden Ressourcen von URLs (HTTP-Verben GET oder POST) abgerufen oder Ressourcen verändert (PUT, DELETE). Der Begriff REST stammt aus der Dissertation von Roy Fielding, einem der Autoren der HTTP-Spezifikation, aus dem Jahre 2000.

Bei einem REST-basierten Webservice werden aufzurufende Methoden durch den URL festgelegt und nicht durch SOAP-Datenstrukturen im HTTP-Header. REST ist nicht nur prägnanter bei der Datenübertragung, sondern auch einfacher zu implementieren.

Voraussetzung für die Nutzung von REST ist die Referenzierung der Assembly *System.ServiceModel.Web.dll*. Der Namensraum *System.ServiceModel.Web* enthält die wichtige Annotation *[WebGet]* und *[WebInvoke]* sowie die Klasse *WebServiceHost*. Die Klasse *WebServiceHost* bietet gegenüber der Klasse *ServiceHost*, die im Allgemeinen verwendet wird, ein paar Grundeinstellungen, die man sonst manuell anlegen müsste.

Das erste Listing zeigt die Deklaration eines REST-basierten WCF-Dienstes. Das zweite Listing demonstriert das codebasierte Hosting (natürlich ist eine deklarative Konfiguration auch möglich).

```
/// <summary>
/// WCF-Dienst für Flugplan im REST-Stil.
/// PerCall == Zustandlosigkeit
/// </summary>
[ServiceBehavior(InstanceContextMode = InstanceContextMode.PerCall)]
[ServiceContract]
public class FlugplanServiceREST
{
    public FlugplanServiceREST()
    {
        // Ausgaben nur zu Testzwecken!!!
        Console.WriteLine("Neue Instanz der Fassade FlugplanServiceREST!");
    }

    [OperationContract]
    [WebGet(UriTemplate = "Flug?Nr={FlugNr}")]
    public de.WWWings.Flug HoleFlug(long FlugNr)
    {
        System.Console.WriteLine("HoleFlug wird aufgerufen mit Nr " + FlugNr);
        return de.WWWings.FlugBLManager.HoleFlug(FlugNr);
    }

    [OperationContract]
    [WebGet(UriTemplate = "AlleFluege")]
    public de.WWWings.FlugMenge HoleAlle()
    {
        return de.WWWings.FlugBLManager.HoleAlle();
    }
}
```

```
[WebGet(UriTemplate = "route?von={von}&{nach}=nach")]
public de.WWWings.FlugMenge HoleAlleRoute(string von, string nach)
{
    return de.WWWings.FlugBLManager.HoleAlle(von, nach);
}
}
```

Listing 29.17 Beispiel für einen REST-basierten Webservices

```
// --- REST-Service Hosting
// Festlegen der URL
Uri URL = new Uri("http://localhost:85/");
// ServiceHost erzeugen
WebServiceHost REST_Host = new WebServiceHost(typeof(de.WWWings.Dienste.FlugplanServiceREST), URL);
// Endpunkt erzeugen
REST_Host.AddServiceEndpoint(typeof(de.WWWings.Dienste.FlugplanServiceREST), new WebHttpBinding(),
"RESTHost");
// Dienst starten
REST_Host.Open();
```

Listing 29.18 Hosting eines REST-basierten Webservices

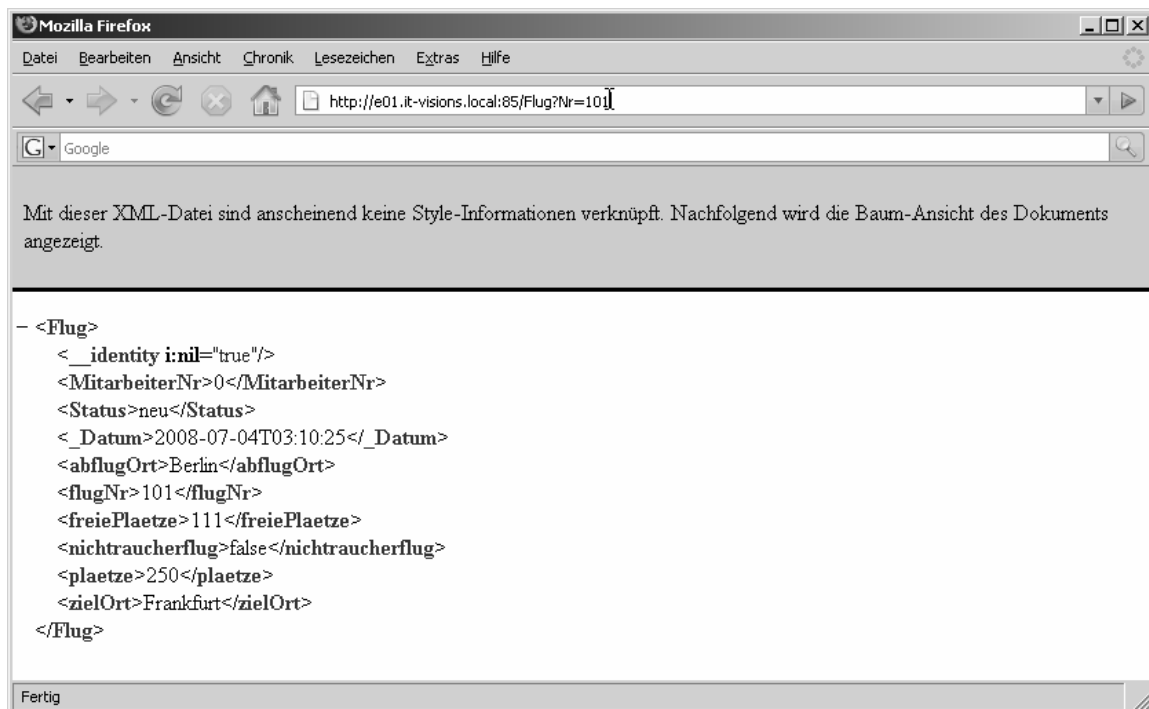


Abbildung 29.28 Beispiel für den Aufruf eines REST-basierten Webservices

Persistente WCF-Dienste (Durable Services)

Persistente Dienste (*Durable Services*) sind WCF-Dienste, die zwischen zwei Methodenaufrufen ihren Zustand in einer Datenbank persistieren können. Dies ist besonders geeignet für lang andauernde Kommunikationsbeziehungen, also zwei Computern, die über einen längeren Zeitraum miteinander in Kontakt stehen. Während dieser Zeit soll auf dem Server ein Zustand verwaltet werden. Der Client kann zwischenzeitlich sogar beendet werden und dennoch später die Kommunikation wieder aufgenommen werden. Dazu persistiert der Server den Zustand in einer Datenbank. Die notwendigen *Token* zur Identifizierung des Clients werden automatisch ausgetauscht.

Microsoft hat diese Idee aus der Windows Workflow Foundation (WF) übernommen (siehe Kapitel zur Persistenz im Kapitel über Windows Workflow Foundation im Buch [HS02]) und bietet sie seit .NET 3.5 auch für die WCF an. Allerdings ist das Datenbankschema ein anderes (viel einfacher). Die persistenten Dienste benötigen nur eine Tabelle (*InstanceData*) und fünf gespeicherte Prozeduren.

HINWEIS Anders als bei dem Persistenzdienst in der WF hat man bei dem WCF-Persistenzdienst die Wahl zwischen binärer Serialisierung und XML-Serialisierung (`serializeAsText="true|false"`). Die WF kann nur binäre Serialisierung. Wie bei der WF liefert Microsoft nur einen Provider für den Microsoft SQL Server, man kann sich aber eigene Provider schreiben.

Folgende Punkte sind zu beachten:

- Mithilfe der im .NET Framework 3.5 Redistributable mitgelieferten SQL-Skripts (*SqlPersistenceProviderSchema.sql* und *SqlPersistenceProviderLogic.sql*) muss man die Datenbankstruktur und die zugehörigen gespeicherten Prozeduren anlegen. Zu beachten ist, dass die Skripts die Datenbank selbst nicht anlegen. Man muss erst eine leere Datenbank mit beliebigem Namen erzeugen und die Skripts legen dann dort die benötigten Elemente an.

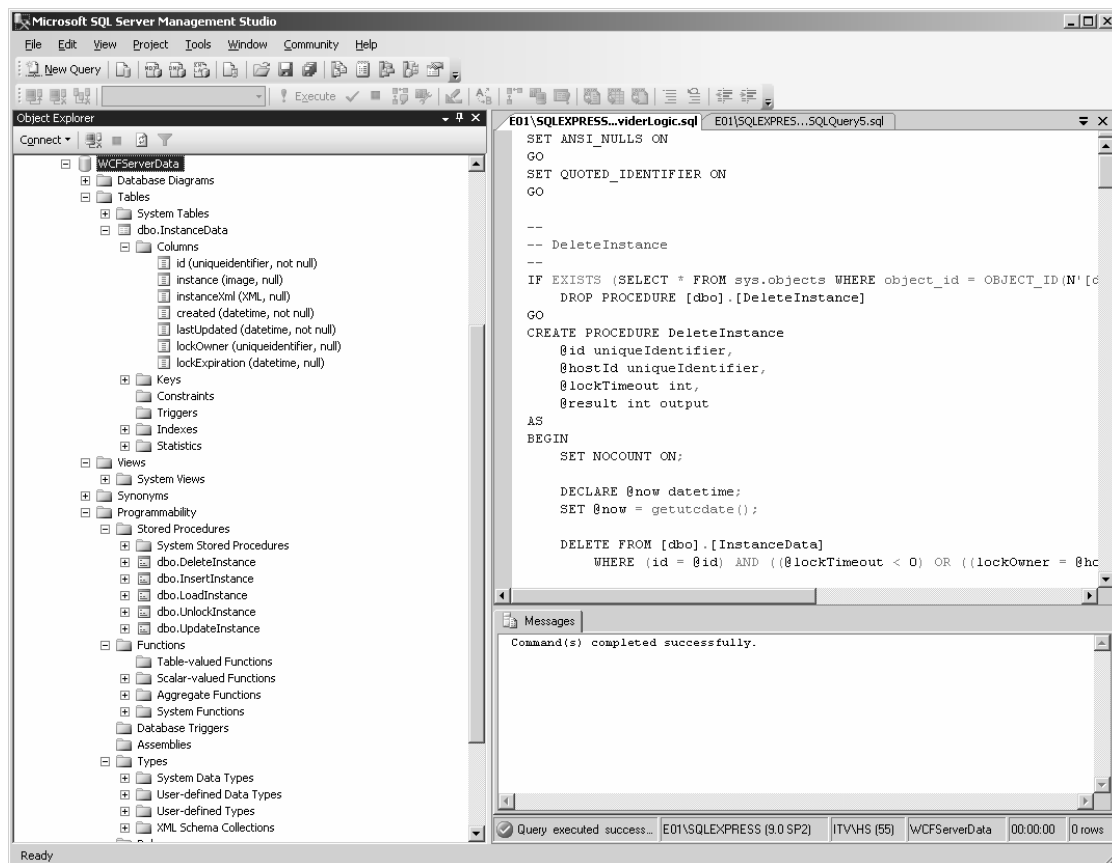


Abbildung 29.29 Anlegen der Datenbank für die persistenten WCF-Dienste

- Eine Dienstklasse, die persistent werden soll, muss mit [DurableService] (Namensraum: System.ServiceModel.Description, Assembly: *System.WorkflowServices.dll*) annotiert sein. Die Schnittstellendefinition bleibt unverändert.
- Jede Methode, nach deren Ende ein Speichervorgang erfolgen soll, muss mit [DurableOperation] annotiert sein. Durch die Zusätze *CanCreateInstance* und *CompletesInstance* kann man die Lebensdauer einer Sitzung steuern.
- In der Konfigurationsdatei muss man die Verbindung zur Datenbank festlegen:

```
<ConnectionStrings>
<add name="CS_WCFServerData" ConnectionString="Data Source=ServerName;Initial
Catalog=WCFServerData;Integrated Security=SSPI"/>
</ConnectionStrings>
```

- In der Konfigurationsdatei muss man den WCF-Persistenzdienst als Dienstverhalten aktivieren:

```
<!-- Persistente Dienste -->
<behavior name="Persistenz">
  <persistenceProvider type="System.ServiceModel.Persistence.SqlPersistenceProviderFactory,
System.WorkflowServices, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"
    ConnectionStringName="CS_WCFServerData"
    persistenceOperationTimeout = "00:00:10"
    lockTimeout="00:01:00"
    serializeAsText="false"/>
</serviceMetadata/>
<serviceDebug includeExceptionDetailInFaults="True" />
</behavior>
```

- Der WCF-Dienst muss mit einer speziellen *Kontextbindung* (z. B. *basicHttpContextBinding*, *wsHttpContextBinding*) konfiguriert werden. Eine Kontextbindung erlaubt den Austausch einer ID zwischen Client und Server, die der Wiedererkennung der Sitzung dient (vgl. Sitzungs-ID in ASP.NET). Außerdem muss man das oben definierte Dienstverhalten zuordnen:

```
<!-- ZUSTANDSDIENST -->
<service behaviorConfiguration="Persistenz" name="de.WWWings.Dienste.Zustandsdienst">
  <endpoint address = "http://localhost:89/WWWings/Zustandsdienst" binding="wsHttpContextBinding"
contract="de.WWWings.Dienste.IZustandsdienst"/>
  <endpoint address="http://localhost:89/WWWings/Zustandsdienst/mex" binding="mexHttpBinding"
contract="IMetadataExchange"/>
</service>
```

- Im Client ist zunächst nichts weiter zu beachten. Einen Proxy kann man wie gewohnt über *Add Service Reference* erstellen und nutzen wie jeden anderen WCF-Dienst. Der Client eines persistenten Dienstes hat aber die zusätzliche Möglichkeit, das vom Server gesendete Identifizierungstoken lokal zu persistieren und zu einem späteren Zeitpunkt in einer neuen Instanz des Clients wieder zu verwenden. Auf diese Weise kann die Kommunikation fortgesetzt werden, selbst wenn der Client zwischenzeitlich beendet war. Ein Beispiel dazu finden Sie in dem Programmcode zu diesem Buch.

Table - dbo.InstanceData		E01\SQLEXPRESS...viderLogic.sql	E01\SQLEXPRESS...SQLQuery5.sql	E01\SQLEXPRESS...iderSchema.sql			
	id	instance	instanceXml	created	lastUpdated	lockOwner	lockExpiration
	5210-186ffd791c59	NULL	<Zustandsdienst xmlns="http://sche...	09.06.2008 23:...	09.06.2008 23:...	NULL	NULL
	5e17a0c9-f5bc-...	<Binary data>	NULL	09.06.2008 23:...	09.06.2008 23:...	6836b5ad-0fc4-4...	09.06.2008 23:13:41
	50e060a0-f300-...	NULL	<Zustandsdienst xmlns="http://sche...	09.06.2008 23:...	09.06.2008 23:...	420d813c-73a5-4...	09.06.2008 23:11:39
	153a32e9-f810-...	NULL	<Zustandsdienst xmlns="http://sche...	09.06.2008 23:...	09.06.2008 23:...	a69deed0-db05-...	09.06.2008 23:10:04
	592e9ef2-d6c5-...	NULL	<Zustandsdienst xmlns="http://sche...	09.06.2008 23:...	09.06.2008 23:...	6211b181-6c67-4...	09.06.2008 23:12:37
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Abbildung 29.30 Zustand der Datenbank mit vier persistierten Dienstinstanzen, davon eine mit binärer Serialisierung

WCF-Sicherheit

WCF-Sicherheit ist ein Thema, über das man eigene dicke Bücher schreiben kann. Hier folgt nur ein sehr kurzer Überblick, bei dem lediglich der Fall der integrierten Windows-Sicherheit behandelt wird. Die Darstellung von Sicherungsmaßnahmen auf Basis digitaler X.509-Zertifikate zur Verwendung in heterogenen Umgebungen würde den Rahmen dieses Buchs sprengen.

Sicherheitsmechanismen

WCF unterstützt zahlreiche Sicherheitsmechanismen. Dazu gehören:

- Verschlüsselung des Datenverkehrs zur Bewahrung von Geheimnissen,
- Sicherstellen der Integrität von Nachrichten,
- Authentifizierung von Client und/oder Server,
- Festlegen und Prüfen der Identität von Client und Server.

Sicherheitsmodi

Sicherheitsfunktionen können auf der Transportebene (z.B. Nutzung von Secure Socket Layer – SSL) und/oder auf der Nachrichtenebene (z.B. Nutzung von WS-Security) verwendet werden. Nachrichtenbasierte Sicherheit bietet mehr Flexibilität, ist aber in der Regel auch aufwendiger zu konfigurieren. In einem gemischten Modus, der von `WSHttpBinding` unterstützt wird, können auch Nachrichtenebene (zur Client-authentifizierung) und Transportebene (für Serverauthentifizierung, Integrität und Vertraulichkeit) gemischt werden. Nur `NetMsmqBinding` unterstützt auch die gleichzeitige Anwendung. Alle Bindungen außer `BasicHttpBinding` bieten in ihrer Grundkonfiguration bereits Sicherheitsfunktionen.

Der Sicherheitsmodus wird festgelegt durch das Attribut `mode` im Element `<security>` unterhalb von `<binding>`.

Authentifizierungsverfahren

Tabelle 29.3 zeigt die von der WCF unterstützten Authentifizierungsverfahren auf Transport- und Nachrichtenebene.

	Transport	Nachricht (Message)
Anonym (None)	X	X
http-Basic nach RFC 2617 (Kennwort als Base64-kodiert)	X	
http-Digest nach RFC 2617 (Kennwort als Hash)	X	
X.509-Zertifikate	X	X
Benutzername/Kennwort		X
Windows (NTLM/Kerberos)	X	X
Windows CardSpace		X

Tabelle 29.3 WCF-Authentifizierungsverfahren

Das Authentifizierungsverfahren `clientCredentialType` ist festzulegen in den Elementen `<transport>` und/oder `<message>` unterhalb von `<security>`.

Beispiel

Das folgende Beispiel aktiviert für eine Bindung die Transportsicherheit auf Basis von Windows-Authentifizierung (NTLM oder Kerberos, abhängig von der Umgebung) und legt fest, dass damit der Datenverkehr verschlüsselt und signiert wird. Sicherheit auf Nachrichtenebene wird nicht angewendet. Dies entspricht der Standardeinstellung für das `NetTcpBinding`.

```
<binding name="Sicher">
  <security mode="Transport">
    <transport clientCredentialType="Windows" protectionLevel="EncryptAndSign" />
    <message clientCredentialType="None" />
  </security>
</binding>
```

Listing 29.19 Sicherheitseinstellung für einen »sicheren« Endpunkt

Das zweite Beispiel legt fest, dass keine Sicherheitsfunktionen verwendet werden.

```
<binding name="Unsicher">
  <security mode="None" />
</binding>
```

Listing 29.20 Sicherheitseinstellung für einen »unsicheren« Endpunkt

HINWEIS	Auf der Clientseite werden die gleichen Einstellungen verwendet.
----------------	--

Übermittlung der Identität

Verlangt der Server eine Authentifizierung, muss der Client eine Identität übermitteln. Im Fall der Windows-Authentifizierung wird – sofern der Client auf `clientCredentialType="Windows"` konfiguriert ist – im Standardfall die Identität des Benutzers, unter dem der Client läuft, übermittelt.

Durch eine Einstellung des Proxys kann der Client jedoch eine andere Identität festlegen:

```
// Andere Identität
proxy.ClientCredentials.Windows.ClientCredential.UserName = "Meier";
proxy.ClientCredentials.Windows.ClientCredential.Domain = "it-visions.local";
proxy.ClientCredentials.Windows.ClientCredential.Password = "geheim";
```

Listing 29.21 Identitätsfestlegung für einen WCF-Aufruf

Der Client kann auch bestimmen, wie der Server die Identität verwenden darf. `Identification` bedeutet, dass der Server die Identität nur zur Authentifizierung verwenden darf. Mit `Impersonation` darf der Server die Identität selbst annehmen und unter dieser agieren:

```
// Festlegung der Art der Verwendung der Identität
proxy.ClientCredentials.Windows.AllowedImpersonationLevel =
System.Security.Principal.TokenImpersonationLevel.Identification;
```

Listing 29.22 Festlegung, wie der Server die Identität verwenden darf

Ermitteln der aktuellen Identität

Aus einem WCF-Dienst heraus kann man die aktuelle Identität des Dienstes ermitteln:

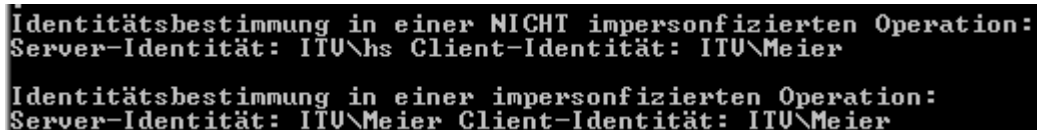
- `System.Security.Principal.WindowsIdentity.GetCurrent().Name` ist der Name des Benutzerkontos, unter dem die aktuelle WCF-Operation läuft.
- `ServiceSecurityContext.Current.WindowsIdentity.Name` ist der Name des Benutzerkontos, unter dem der Client sich authentifiziert hat.

Nutzung der Identität

Normalerweise nutzt der Server die Identität nur zur Authentifizierung. Er kann aber eine Operation damit impersonifizieren, das heißt die Identität des Clients für die Ausführung der Methode übernehmen. Dazu ist die Methode mit `[OperationBehavior(Impersonation = ImpersonationOption.Required)]` zu annotieren:

```
// Impersonifizierung
[OperationBehavior(Impersonation = ImpersonationOption.Required)]
public string GetServerIdentity_Impersonated()
{
    // Return the InstanceContextMode of the service
    string ausgabe = PingInfo.GetIdentity(); // Eigene Hilfsroutine, siehe Projekt!
    Console.WriteLine("Identitätsbestimmung in einer impersonifizierten Operation:");
    Console.WriteLine(ausgabe);
    return ausgabe;
}
```

Listing 29.23 Beispiel für eine Operation, die impersonifiziert wird



```
Identitätsbestimmung in einer NICHT impersonifizierten Operation:
Server-Identität: ITU\hs Client-Identität: ITU\Meier

Identitätsbestimmung in einer impersonifizierten Operation:
Server-Identität: ITU\Meier Client-Identität: ITU\Meier
```

Abbildung 29.31 Aufruf einer nicht impersonifizierenden und einer impersonifizierenden Variante des obigen Beispiels

Zugriffsrechte

Ein WCF-Dienst darf festlegen, wer die Operationen aufrufen darf. Dies erfolgt am einfachsten mit der Annotation `[PrincipalPermission]`. Die Annotation ist auf der Methode zu verwenden, welche die WCF-Operation implementiert.

Beispiele

- Nur der Benutzer *Hschwichtenberg* darf die Operation aufrufen:

```
[PrincipalPermission(SecurityAction.Demand, Name="HSchwichtenberg")]
```

- Alle Mitglieder der Gruppe *WCFBenutzer* dürfen die Methode aufrufen:

```
[PrincipalPermission(SecurityAction.Demand, Role="WCFBenutzer")]
```

WICHTIG Der Versuch, von anderen Benutzeridentitäten aus eine derart annotierte Operation aufzurufen, führt zum Fehler »Access is denied«.

Protokollierung

Die WCF enthält bereits ein System zur Überwachung der Kommunikation zwischen zwei Endpunkten. Dabei setzt die WCF auf der in der .NET-Klassenbibliothek vorhandenen Protokollierungsinfrastruktur auf, die folgende Listener definiert:

- `System.Diagnostics.DefaultTraceListener`: Protokollierung in das Ausgabefenster von Visual Studio,
- `System.Diagnostics.EventLogTraceListener`: Protokollierung in das Windows-Ereignisprotokoll,
- `System.Diagnostics.ConsoleTraceListener`: Protokollierung in das Konsolenfenster,
- `System.Diagnostics.DelimitedListTraceListener`: Protokollierung in eine Datei mit Trennzeichen,
- `System.Diagnostics.XmlWriterTraceListener`: Protokollierung in eine Datei in XML-Form.

Beispiel

Im Service Configuration Editor sind folgende Einstellungen notwendig, um die Protokollierung in eine Datei zu aktivieren:

- Unter *Diagnostics/Message Logging* sind allgemeine Einstellungen erforderlich, z.B. `logEntireMessage = true`.
- Unter *Sources* ist `System.ServiceModel` als eine neue Quelle zu definieren.
- Unter *Listeners* ist ein neuer Listener zu erzeugen. Dabei sind ein Dateiname bei `InitData` und die zuvor angelegte Quelle anzugeben (Abbildung 29.32).

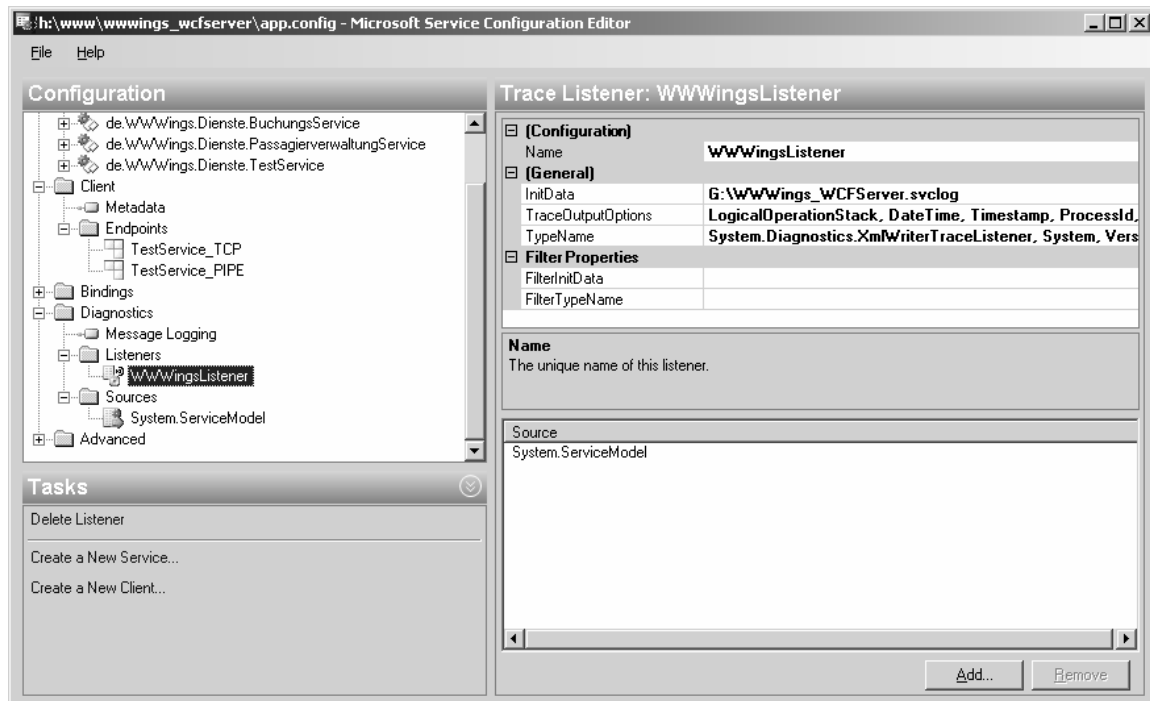


Abbildung 29.32 Konfiguration der Protokollierung in einer Textdatei

```
<sources>
  <source name="System.ServiceModel" switchValue="Information,ActivityTracing"
    propagateActivity="true">
    <listeners>
      <add type="System.Diagnostics.DefaultTraceListener" name="Default">
        <filter type="" />
      </add>
      <add name="WWWingsListener">
        <filter type="" />
      </add>
    </listeners>
  </source>
</sources>
<sharedListeners>
  <add initializeData="G:\app_trace\log1.svclog" type="System.Diagnostics.XmlWriterTraceListener,
System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
    name="WWWingsListener" traceOutputOptions="LogicalOperationStack, DateTime, Timestamp, ProcessId,
ThreadId, Callstack">
    <filter type="" />
  </add>
</sharedListeners>
</system.diagnostics>
</ConnectionStrings>
```

```

<add name="CS_WWWings" ConnectionString="Data
Source=.\SQLEXPRESS;AttachDbFilename=h:\www\Datenbanken\WorldWideWings.mdf;Integrated Security=True;"
providerName="System.Data.SqlClient"/>
</ConnectionStrings>

<system.serviceModel>
  <diagnostics>
    <messageLogging logEntireMessage="true" />
  </diagnostics>
  ...
</system.serviceModel>

```

Listing 29.24 Erzeugte Konfigurationseinträge

Die erzeugte Datei kann mit dem Programm Service Trace Viewer (*SvcTraceViewer.exe*), das im Windows SDK enthalten ist, betrachtet werden (Abbildung 29.33).

The screenshot shows the Microsoft Service Trace Viewer application. The left pane displays a list of activities with columns for Activity, # Traces, and Duration. The right pane shows a detailed view of a selected activity, including its description, level, thread ID, process name, time, and trace identifier. The 'Basic Information' section shows details like Activity Name, Related Activity Name, Time, Level, Source, Process, Thread, Computer, and Trace Identifier/Code. The 'Diagnostics Information' section shows properties like Name and Value. The 'Stack Trace' section shows the method call stack.

Activity	# Traces	Duration
Open ServiceHost 'de.WWWings.Dienste.TestService'.	11	15ms
Open ServiceHost 'de.WWWings.Dienste.Passagierverwaltung'.	11	0ms
Listen at 'net.pipe://localhost/WWWings/Pipes/Passagier...'.	3	0ms
Listen at 'net.pipe://localhost/WWWings/Pipes/TestService'.	4	494ms
Open ClientBase. Contract type: 'WWWings.WCFServer.Sel...'.	6	31ms
Construct ChannelFactory. Contract type: 'WWWings.WCFServer.Sel...'.	5	0ms
Process action 'http://de.ITVisions.Samples.WCF/ITTestSevi...'.	18	78ms
Receive bytes on connection 'net.tcp://localhost:1234/'.	9	78ms
Processing message 1.	5	15ms
Execute 'de.WWWings.Dienste.ITTestService.Ping'.	4	0ms
Close ClientBase. Contract type: 'WWWings.WCFServer.Sel...'.	5	0ms
Process action 'http://de.ITVisions.Samples.WCF/ITTestSevi...'.	5	15ms
Open ClientBase. Contract type: 'WWWings.WCFServer.Sel...'.	5	15ms
Construct ChannelFactory. Contract type: 'WWWings.WCFServer.Sel...'.	5	0ms
Receive bytes on connection 'net.pipe://localhost:1234/'.	23	1s
Processing message 2.	5	15ms
Execute 'de.WWWings.Dienste.ITTestService.Ping'.	4	15ms
Process action 'http://de.ITVisions.Samples.WCF/ITTestSevi...'.	17	62ms
Processing message 3.	5	0ms
Execute 'de.WWWings.Dienste.ITTestService.Ping'.	4	0ms
Receive bytes on connection 'net.tcp://localhost:1234/'.	9	125ms
Processing message 4.	5	15ms
Process action 'http://de.ITVisions.Samples.WCF/ITTestSevi...'.	10	15ms
Execute 'de.WWWings.Dienste.ITTestService.Ping'.	4	15ms
Receive bytes on connection 'net.tcp://localhost:1234/'.	9	15ms
Process action 'http://de.ITVisions.Samples.WCF/ITTestSevi...'.	10	15ms
Processing message 5.	5	0ms
Execute 'de.WWWings.Dienste.ITTestService.Ping'.	4	0ms
Receive bytes on connection 'net.tcp://localhost:1234/'.	9	15ms
Process action 'http://de.ITVisions.Samples.WCF/ITTestSevi...'.	10	15ms
Processing message 6.	5	0ms
Execute 'de.WWWings.Dienste.ITTestService.Ping'.	4	0ms
Receive bytes on connection 'net.tcp://localhost:1234/'.	17	1s
Process action 'WWWings.PassagierverwaltungService/Hol...'.	10	546ms
Execute 'de.WWWings.Dienste.PassagierverwaltungService...'.	4	531ms
Processing message 7.	5	0ms
Receive bytes on connection 'net.tcp://localhost:1234/'.	32	10s
Processing message 8.	5	15ms
Process action 'http://de.ITVisions.Samples.WCF/ITTestSevi...'.	10	0ms
Execute 'de.WWWings.Dienste.ITTestService.Ping'.	4	0ms
Process action 'http://de.ITVisions.Samples.WCF/ITTestSevi...'.	10	15ms
Processing message 9.	5	0ms
Execute 'de.WWWings.Dienste.ITTestService.Ping'.	4	0ms
Processing message 10.	5	0ms
Execute 'de.WWWings.Dienste.ITTestService.Ping'.	4	0ms
Processing message 11.	5	0ms
Process action 'http://de.ITVisions.Samples.WCF/ITTestSevi...'.	10	0ms

Abbildung 29.33 Analyse der Protokolldatei

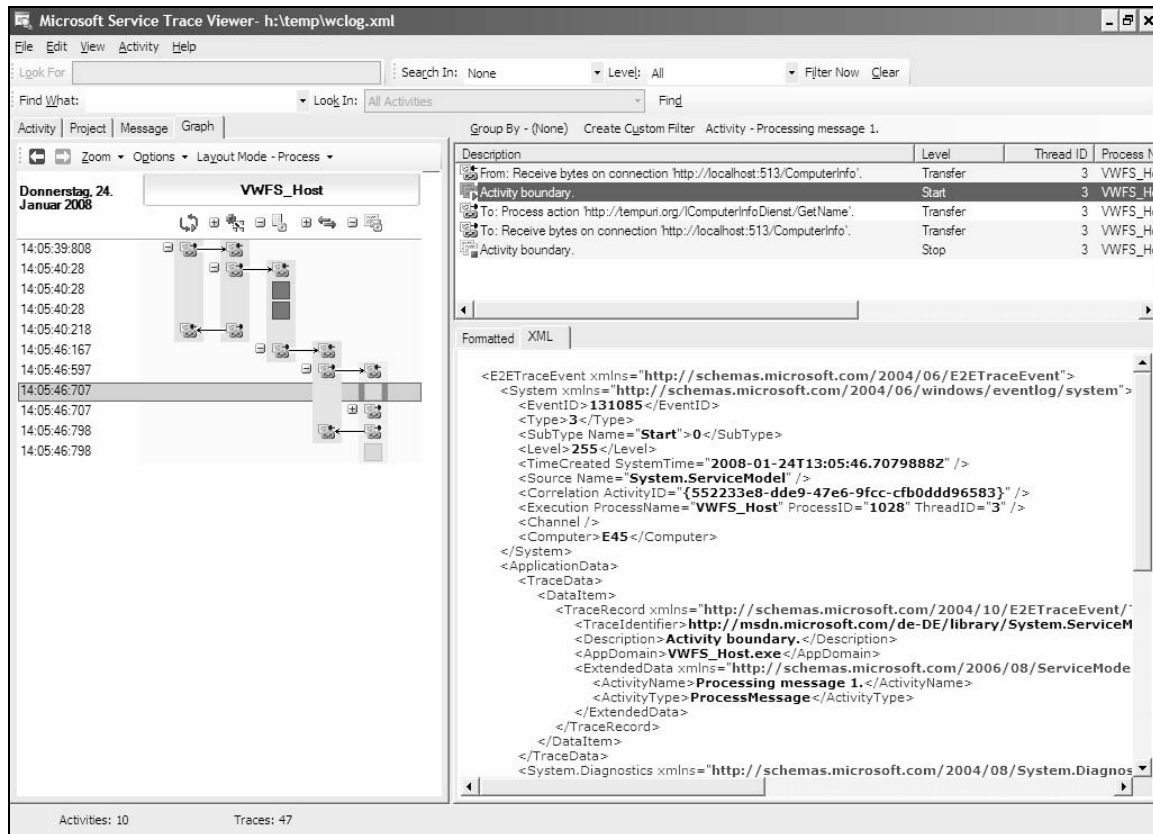


Abbildung 29.34 Grafische Darstellung des Kommunikationsablaufs

HINWEIS Protokolldateien können bei vielen Clients sehr schnell sehr groß werden (unter Umständen mehrere GByte). Der Service Trace Viewer unterstützt aber das partielle Laden großer Protokolldateien für einen bestimmten Zeitraum (Abbildung 29.35). Das Programm stellt automatisch fest, dass eine Datei groß ist, und bietet dann das partielle Laden an.

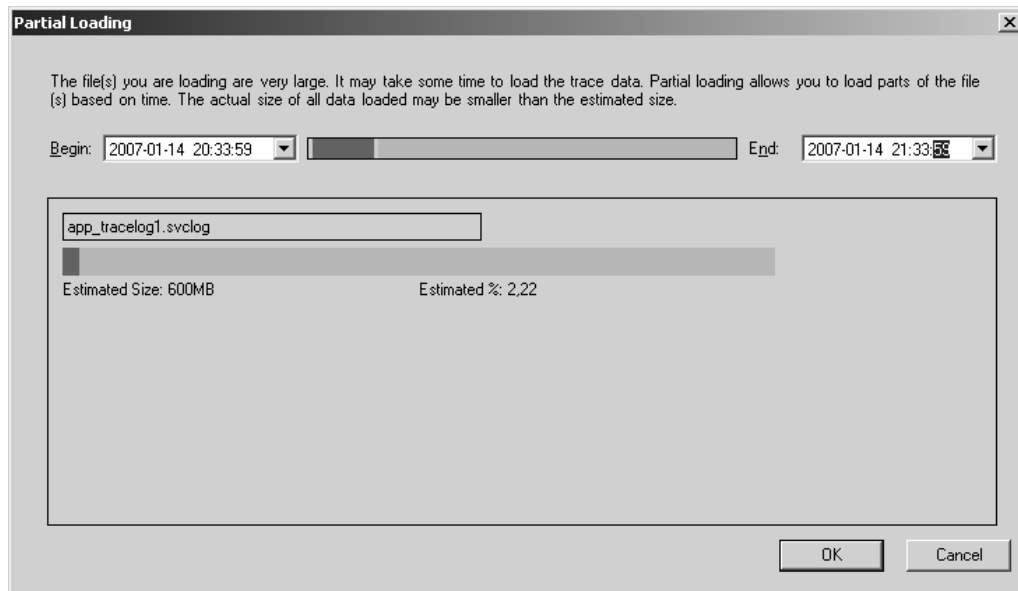


Abbildung 29.35 Partielles Laden von Protokolldateien mit dem Service Trace Viewer

Weitere Funktionen

Dieser Abschnitt erläutert noch kurz einige weitere Funktionen von WCF, die bisher nicht erwähnt wurden.

Funktionen des Service Model Metadata Utility Tools (svcutil.exe)

Das Kommandozeilenwerkzeug Microsoft Service Model Metadata Utility Tool (*svcutil.exe*) erfüllt viele verschiedene Aufgaben, dazu gehören unter anderem:

- Überprüfen einer Implementierung eines WCF-Dienstes:

```
svcutil H:\WWW\WWWings_WCFServer\bin\Debug\WWWings_WCFServer.exe /validate  
/servicename:de.WWWings.Dienste.FlugplanService
```

- Exportieren von WCF-Metadaten in Form von *.wsdl*- und *.xsd*-Dateien aus einer Assembly, die WCF-Dienste implementiert:

```
svcutil H:\WWW\WWWings_WCFServer\bin\Debug\WWWings_WCFServer.exe
```

- Herunterladen von WCF-Metadaten in Form von *.wsdl*- und *.xsd*-Dateien von einem Metadaten-Endpunkt:

```
svcutil /t:metadata net.tcp://localhost:1234/WWWings/metadaten/
```

- Erzeugung von WCF-Proxys und WCF-Client-Konfigurationseinstellungen aus Metadaten (aus Dateien oder von einem Metadatenendpunkt):

```
svcutil /t:code net.tcp://localhost:1234/WWWings/metadaten/ /language:cs
```

- Statische Generierung des Serialisierungscode für die XML-Serialisierung (steigert die Geschwindigkeit beim ersten Aufruf von WCF-Diensten):

```
svcutil /t:xmlserializer H:\WWW\WWWings_WCFServer\bin\Debug\WWWings_WCFServer.exe
```

ACHTUNG *svcutil.exe* überschreibt bestehende gleichnamige Dateien auf der Festplatte beim Generieren von Code- und XML-Dateien. Mit */mergeConfig* kann erreicht werden, dass die neuen Zeilen an bestehende Zeilen angehängt werden.

Ein-Weg-Kommunikation (unidirektionale Kommunikation, engl. One-Way)

Die Standardeinstellung in der WCF ist die Zwei-Wege-Kommunikation (alias Frage-Antwort-Kommunikation): Ein Client ruft eine Operation auf einem Server auf und wartet so lange, bis die Operation beendet ist und eine Antwort an den Client sendet. Die WCF unterstützt alternativ das Ein-Weg-Kommunikationsmuster, bei dem der Client nicht wartet, sondern direkt fortfährt, und der Server auch gar keine Antwort sendet.

Ein-Weg-Kommunikation wird deklariert durch das Attribut `IsOneWay` in der Annotation `[OperationContract]`. Die Methode darf keinen Rückgabewert haben.

WICHTIG Bei der Ein-Weg-Kommunikation erhält der Client weder einen Rückgabewert noch eine Fehlermeldung im Fall eines Verarbeitungsfehlers auf dem Server. Nur wenn die betreffende Operation gar nicht erst gefunden wurde, oder der Server nicht verfügbar ist, bekommt der Client eine Fehlermeldung.

Beispiel

Das folgende Beispiel zeigt die relevanten Fragmente aus der Implementierung eines Dienstes, mit dem sich der Client gegenüber dem Server als »lebendig« melden kann. Für den Client ist es dabei nicht wichtig, ob der Server die Nachricht korrekt verarbeiten konnte.

```
[OperationContract(IsOneWay = true)]  
void SendeLebenszeichen(string Client);
```

Listing 29.25 Ausschnitt aus der Schnittstelle `ITestService`

```
public void SendeLebenszeichen(string Client)  
{  
    Console.WriteLine("Lebenszeichen von Client: " + Client);  
}
```

Listing 29.26 Ausschnitt aus der Klasse `TestService`

Duplexkommunikation

Duplexkommunikation ist neben der Ein-Weg- und der Zwei-Wege-Kommunikation die dritte von WCF unterstützte Kommunikationsform. Bei der Duplexkommunikation erhält der Client keine sofortige Rückmeldung, sondern eine (oder sogar mehrere) spätere Rückmeldung(en) (Callbacks).

Für die Duplexkommunikation müssen folgende Voraussetzungen erfüllt sein:

- Die verwendete WCF-Bindung muss Duplexkommunikation unterstützen (z.B. `WSDualHttpBinding`, `NetTcpBinding` und `NetNamedPipeBinding`).
- Der Server deklariert zwei Schnittstellen (eine Dienstschnittstelle und eine Rückrufschnittstelle) mit jeweils einer Ein-Weg-Operation.
- Der Server muss die Dienstschnittstelle implementieren, der Client die Rückrufschnittstelle.
- Die Dienstschnittstelle muss die Annotation `[ServiceContract]` besitzen, in der im Attribut `CallbackContract` der Typ der Rückrufschnittstelle genannt ist.
- Innerhalb der aufzurufenden Operation in der Dienstschnittstelle beschafft man sich einen Zeiger auf den serverseitigen Dienstproxy für die clientseitige Rückrufschnittstellenimplementierung des Clients über `OperationContext.Current.GetCallbackChannel<IRueckrufSchnittstelle>()`.
- Die zweite Rückrufschnittstelle kann, muss aber nicht mit `[ServiceContract]` annotiert sein.
- Der Client muss die Rückrufschnittstelle implementieren und diese Implementierung bei der Instanziierung des Proxys im Konstruktor übergeben.

Beispiel

Das folgende Beispiel realisiert einen asynchronen beidseitigen Ping. Nachdem der Server einen Ping-Aufruf mit dem Namen des Clients erhalten hat, sendet er zehn Pings an den Client zurück.

```
/// <summary>
/// Rückrufschnittstelle für WCF-Duplex
/// </summary>
public interface IpingDuplexCallback
{
    [OperationContract(IsOneWay = true)]
    void ClientPing(string Server);
}
```

Listing 29.27 Rückrufschnittstelle

```
[OperationContract(IsOneWay = true)]
void AsyncPing(string Client);
```

Listing 29.28 Operation der Dienstschnittstelle

```
public void AsyncPing(string Client)
{
    Console.WriteLine("Async-Ping von Client: " + Client);
    // Hole Zeiger auf Implementierung des Server-Proxy für die Rückrufschnittstellen-Implementierung des
    Clients
```

```

IpingDuplexCallback client = OperationContext.Current.GetCallbackChannel<IpingDuplexCallback>();
// Rufe Client 10x auf
for (int a = 0; a < 10; a++)
{
    client.ClientPing("Rückruf #" + a + " von " + System.Environment.MachineName);
    System.Threading.Thread.Sleep(1000);
}
}

```

Listing 29.29 Implementierung der Operation der Dienstschnittstelle

```

class WCFDemos : TestService.ItestServiceCallback
{
    public void BspWCFDuplex()
    {
        TestService.TestServiceClient t = new de.WWings.TestService.TestServiceClient(new
        InstanceContext(this, "TestService_TCP");
        t.AsyncPing("KonsolenClient");
        Console.ReadLine();
        t.Close();
    }
    // Rückrufoperation
    public void ClientPing(string Server)
    { Console.WriteLine("Callback vom Server: " + Server); }
}

```

Listing 29.30 Implementierung des Clients

HINWEIS In dem obigen Beispiel bietet die Klasse, die den WCF-Dienst aufruft, gleichzeitig selbst die Rückrufoperation an. Daher übergibt man im Konstruktor des Proxys eine mit `this` gefüllte Instanz von `System.ServiceModel.InstanceContext`. Die Rückrufoperation könnte auch von einer anderen Klasse realisiert werden.

Die in dem Beispiel verwendete Schnittstelle `ItestServiceCallback` wird durch den Proxygenerator erstellt. Diese Schnittstelle entspricht auf der Serverseite `IpingDuplexCallback`.

ACHTUNG Der Client darf den Proxy nicht schließen, solange er noch Rückrufe erwartet!

Asynchrone WCF-Aufrufe

Ein-Weg- und Duplexkommunikation sind nicht zu verwechseln mit dem asynchronen Aufruf von Zwei-Wege-Operationen. Beim asynchronen Aufruf arbeitet der Client direkt nach dem Aufruf einer Operation weiter, erwartet jedoch in einem anderen Thread genau einen Rückruf, wenn das Ergebnis des Aufrufs bereitsteht.

Der Proxygenerator für Webservices in Visual Studio (*Add Web Reference*) erstellt automatisch zu jeder Methode auch eine asynchrone Aufrufoption. Bei *Add Service Reference* für WCF-Dienste entstehen nicht automatisch asynchrone Aufrufe. Derzeit kann man diese nur mit dem Kommandozeilenwerkzeug `svcutil.exe` mit der Option `/async` generieren lassen. Dadurch entstehen zu jeder Operation zwei weitere Methoden mit den

vorangestellten Wörtern *Begin* und *End* im Namen. Die andere Alternative ist das manuelle Starten eines zweiten Threads vor dem Aufruf einer WCF-Operation, entweder mit einem asynchronen Methodenaufruf oder mit einer Instanz der Klasse Thread.

Sitzungen (Sessions)

Sitzungen (Sessions) bedeuten, dass jeder Instanz eines Clientproxys genau eine Instanz der Dienstklasse zugeordnet wird, und deren Zustand zwischen zwei Operationsaufrufen erhalten bleibt. Diese Option ist nur für Bindungen verfügbar, die Sitzungszustände unterstützen. Standardmäßig sind dies `NetTcpBinding` und `NetNamedPipeBinding`. Optional ist dies möglich für `WSHttpBinding` und `WSDualHttpBinding`. Technisch werden Sitzungen entweder durch einen permanenten Kanal (TCP, Named Pipes) oder durch die Protokolle WS-SecureConversation/WS-ReliableMessaging (http) abgebildet.

WICHTIG Ob Sitzungen aktiv oder nicht aktiv sind, hängt von den Standardeinstellungen der ausgewählten WCF-Systembindungen bzw. der eigenen Bindungsdefinition ab. Standardmäßig unterstützen `NetTcpBinding` und `WSHttpBinding` Sitzungen, während `BasicHttpBinding` keine Sitzungen bietet.

Eine WCF-Dienstklasse kann deklarieren, dass Sitzungen verwendet werden müssen:

```
[ServiceContract(... SessionMode = SessionMode.Required ... )]
```

Eine Sitzung beginnt, wenn die erste Operation aufgerufen wird (nicht schon bei der Instanziierung des Clientproxys). Wann eine Sitzung endet und eine neue Instanz erzeugt wird, wird gesteuert über `[OperationBehavior(ReleaseInstanceMode = ReleaseInstanceMode.AfterCall)]`. Erlaubte Werte sind `AfterCall`, `BeforeCall`, `BeforeAndAfterCall` und `None` (Standard == Steuerung nur über `InstanceContextMode`).

Der Entwickler der Klasse kann definieren, dass einige Operationen nicht zum Start einer Sitzung berechtigen.

Beispiel

Die Methode, welche die Anzahl der Operationsaufrufe zählt, darf nicht zuerst aufgerufen werden:

```
[OperationContract(IsInitiating = false)]  
public int GetOperationCount() { ... }
```

Ein mehrfacher Aufruf einer Methode, die initialisierend ist, ist kein Problem. Es muss immer ein `IsInitiating = true` geben. Standardmäßig ist dies bei allen Methoden gesetzt.

Der Entwickler der Klasse kann auch definieren, dass einige Operationen eine Sitzung beenden.

Beispiel

Die Methode `SitzungBeenden()` beendet die Sitzung. Dazu muss die Methode keine besondere Implementierung besitzen (sie kann leer sein!). Nach dem Aufruf einer Methode mit `isTerminating=true` kann der Client keine andere Methode mehr aufrufen:

```
[OperationContract(IsTerminating = true)]  
Public void SitzungBeenden() { ... }
```

HINWEIS Standardmäßig sind alle Methoden initialisierend. Der mehrfache Aufruf von Methoden, die initialisierend sind, ist kein Problem: Es gibt trotzdem nur eine Sitzung. Nach dem Aufruf einer Methode mit `isTerminating=true` kann der Client keine andere Methode mehr aufrufen.

ACHTUNG Da es eine Serverinstanz pro Client-Proxy-Instanz (nicht pro Client!) gibt, achten Sie darauf, dass der Client wirklich nur die Anzahl von Instanzen erzeugt, die Sie wollen. Es sollte keinesfalls immer der Proxy neu instanziiert werden!

Beispiele in World Wide Wings

Das World Wide Wings-Beispiel ist umfangreicher als die bisher in diesem Kapitel abgedruckten Listings. In dem WCF-Server in World Wide Wings gibt es insgesamt sechs verschiedene Dienste. Einige Dienste sind mit mehreren Endpunkten erreichbar:

- Ein Windows-Systemdienst, der auch als Konsolenanwendung gestartet werden kann, bietet Endpunkte für TCP, Named Pipes und http.
- Ein HTTP-Endpunkt in der WorldWideWings-Webanwendung.

Es existiert jeweils auch ein Metadatenendpunkt, der über TCP bzw. HTTP erreichbar ist.



Abbildung 29.36 WCF-Konfiguration für World Wide Wings

Der WCF-Server ist als Windows-Systemdienst mit der zusätzlichen Option zum Start als Konsolenanwendung implementiert, damit Protokollmeldungen über den Start des Servers und die einzelnen Aufrufe leicht sichtbar gemacht werden können. Das folgende Listing zeigt die Implementierung des WCF-Service-Hosts mit insgesamt vier Diensten. Der WCF-Service-Host startet die Dienste wahlweise als Single-Call- oder Singleton-Objekte:

```

Class WWWingsWCFServiceHost
{
    static System.Collections.Generic.List<ServiceHost> WCFDienste = new
System.Collections.Generic.List<ServiceHost>();
    /// <summary>
    /// Start des WCF-Servers
    /// </summary>
    public static void StartService()
    {
        bool Singleton = false;
        Uri baseAddress = new Uri("net.tcp://localhost:1234/WWWings/");
        if (Singleton)
        { // Singleton
            WCFDienste.Add(new ServiceHost(new de.WWWings.Dienste.FlugplanService(), baseAddress));
            WCFDienste.Add(new ServiceHost(new de.WWWings.Dienste.FlugplanverwaltungService(), baseAddress));
            WCFDienste.Add(new ServiceHost(new de.WWWings.Dienste.Buchungsservice(), baseAddress));
            WCFDienste.Add(new ServiceHost(new de.WWWings.Dienste.PassagierverwaltungService(), baseAddress));
        }
        else
        { // Single Call
            WCFDienste.Add(new ServiceHost(typeof(de.WWWings.Dienste.FlugplanService), baseAddress));
            WCFDienste.Add(new ServiceHost(typeof(de.WWWings.Dienste.FlugplanverwaltungService), baseAddress));
            WCFDienste.Add(new ServiceHost(typeof(de.WWWings.Dienste.Buchungsservice), baseAddress));
            WCFDienste.Add(new ServiceHost(typeof(de.WWWings.Dienste.PassagierverwaltungService), baseAddress));
        }

        // Protokollausgabe
        foreach (ServiceHost Host in WCFDienste)
        {
            Print("Starte Dienst " + Host.Description.Name, ConsoleColor.Yellow);
            foreach (System.ServiceModel.Description.ServiceEndpoint e in Host.Description.Endpoints)
            {
                Print(" – Endpunkt: " + e.Address + " (" + e.Binding.Name + ")", ConsoleColor.White);
            }
            Host.Open();
        }
    }
    /// <summary>
    /// Beenden des WCF-Servers
    /// </summary>
    internal static void StopService()
    {
        foreach (ServiceHost Host in WCFDienste)
        {
            if (Host.State != CommunicationState.Closed)
                Host.Close();
        }
    }
}

```

```

/// <summary>
/// Hilfsmethode für Protokollierung
/// </summary>
private static void Print(string Text, System.ConsoleColor Farbe)
{
    Console.ForegroundColor = Farbe;
    Console.WriteLine(Text);
    Console.ForegroundColor = ConsoleColor.Gray;
}
}

```

Listing 29.31 Implementierung des WCF-Servers [WCFServer/ServiceHost.cs]

```

c:\ file:///H:/WWW/WWWings_WCFServer/bin/Debug/WWWings_WCFServer.EXE
WCF-Server wird gestartet...
Neue Instanz der Fassade FlugplanService!
Neue Instanz der Fassade FlugplanverwaltungService!
Neue Instanz der Fassade Buchungsservice
Neue Instanz der Fassade PassagierverwaltungService!
Starte Dienst FlugplanService
- Endpunkt: net.tcp://localhost:1234/WWWings/Flugplanservice <NetTcpBinding>
- Endpunkt: net.pipe://localhost/WWWings/Pipes/Flugplanservice <NetNamedPipeBinding>
- Endpunkt: net.tcp://localhost:1234/WWWings/metadaten/Flugplanservice <MetadataExchangeTcpBinding>
- Endpunkt: http://localhost/Webservices/Flugplanservice.svc <BasicHttpBinding>
Starte Dienst FlugplanverwaltungService
- Endpunkt: net.tcp://localhost:1234/WWWings/FlugplanverwaltungService <NetTcpBinding>
- Endpunkt: net.pipe://localhost/WWWings/Pipes/FlugplanverwaltungService <NetNamedPipeBinding>
- Endpunkt: net.tcp://localhost:1234/WWWings/metadaten/FlugplanverwaltungService <MetadataExchangeTcpBinding>
Starte Dienst Buchungsservice
- Endpunkt: net.tcp://localhost:1234/WWWings/Buchungsservice <NetTcpBinding>
- Endpunkt: net.pipe://localhost/WWWings/Pipes/Buchungsservice <NetNamedPipeBinding>
- Endpunkt: net.tcp://localhost:1234/WWWings/metadaten/Buchungsservice <MetadataExchangeTcpBinding>
Starte Dienst PassagierverwaltungService
- Endpunkt: net.tcp://localhost:1234/WWWings/PassagierverwaltungService <NetTcpBinding>
- Endpunkt: net.pipe://localhost/WWWings/Pipes/PassagierverwaltungService <NetNamedPipeBinding>
- Endpunkt: net.tcp://localhost:1234/WWWings/metadaten/PassagierverwaltungService <MetadataExchangeTcpBinding>
WCF-Server ist jetzt gestartet!

```

Abbildung 29.37 Protokollmeldungen beim Start des WCF-Servers

Übliche Stolpersteine

In diesem Abschnitt werden einige typische Stolpersteine bei der Arbeit mit WCF diskutiert.

Typische Fehlermeldungen

Wenn beim Start einer .NET-Anwendung der Fehler »Configuration system failed to initialize/Unrecognized configuration section system.serviceModel.« auftritt, ist dies ein Hinweis darauf, dass WCF (also das .NET Framework 3.0 oder 3.5) auf dem System nicht installiert ist.

Erhalten Sie beim Start des WCF-Servers die Fehlermeldung »The contract name xy could not be found in the list of contracts implemented by the service abc.«, liegt das daran, dass auf der Klasse oder der Schnittstelle der Dienstklasse die Auszeichnung [ServiceContract] fehlt.

Wenn Sie beim Start des WCF-Servers die Fehlermeldung »ContractDescription xy has zero operations; a contract must have at least one operation.« erhalten, fehlen die Auszeichnungen mit [OperationContract].

Erhalten Sie beim Start des WCF-Servers die Fehlermeldung »Service has zero Application (non-infrastructure) Endpoints.«, gibt es weder in der Konfigurationsdatei noch im Programm eine Festlegung für Adresse und Bindung für eine Dienstklasse.

Standardbegrenzungen (Throttling)

Die WCF begrenzt in den Standardeinstellungen sowohl die Nachrichtengröße als auch die Anzahl der aktiven Sitzungen. Die wichtigsten Begrenzungen sind:

- Anzahl gleichzeitiger Sitzungen (maxConcurrentSessions): 10,
- Anzahl gleichzeitiger Aufrufe (maxConcurrentCalls): 16,
- Maximale Puffergröße (maxBufferSize): 64 KByte,
- Maximale Nachrichtengröße (maxReceivedMessageSize): 64 KByte,
- Maximale Größe einer Objektmenge (maxArrayLength): 16384.

Diese Einstellung muss man verändern in der XML-Konfiguration eines WCF-Dienstes unter:

configuration/system.serviceModel/behaviors/serviceBehaviors/serviceThrottling

HINWEIS

Microsoft betrachtet die oben genannten niedrigen Grenzen nicht als einen Fehler, sondern als eine Funktion, die vor Überlastungsangriffen schützt.

Angabe von http-Adressen

Beim Hosting in den IIS ist der Standort der .svc-Datei die physikalische Basisadresse. Die Adressangaben in der Endpunktconfiguration sind in der Regel relativ dazu angegeben. Bei der Übernahme einer solchen Konfiguration in einen Konsolen- oder Windows-Dienst-Host muss man die Adressen in den Endpunkten entweder absolut spezifizieren oder aber in der Sektion <Host> unterhalb von <Service> eine Basisadresse benennen:

```
<host>
  <baseAddresses>
    <add baseAddress="http://E01:86/WWWingsDienst"/>
  </baseAddresses>
</host>
```

Ohne Adressangabe kommt es zum Fehler »Could not find a base address that matches scheme http for the endpoint with binding BasicHttpBinding. Registered base address schemes are [].«.

Eindeutigkeit der Bindung

Bei der Instanziierung eines Proxys muss das zu verwendende Binding eindeutig sein. Wenn in der Konfigurationsdatei des Clients mehrere Bindings erscheinen (z.B. weil der Server mehrere anbietet), dann läuft die Instanziierung des Proxys auf den Server: »An endpoint configuration section for contract 'ServiceReference1.Iservice' could not be loaded because more than one endpoint configuration for that contract was found. Please indicate the preferred endpoint configuration section by name.« Hier gibt es zwei Möglichkeiten: entweder das Reduzieren der Konfigurationsdatei des Clients auf eine Endpunktkonfiguration oder aber – wenn der Nutzer des Clients die Wahl haben soll – die Angabe der gewünschten Konfiguration anhand ihres Namens beim Instanziiieren des Proxys:

```
ServiceReference1.ServiceClient c = new ConsoleApplication1.ServiceReference1.ServiceClient("TCP");
Dim c As ServiceReference1.ServiceClient = _
```

Gemeinsame Proxytypen (Proxy Type Sharing)

Normalerweise erzeugen die Proxygeneratoren für jeden WCF-Service eigene Proxyklassen. Es kann aber vorkommen, dass zwei WCF-Dienste als Parameter die gleiche Datenklasse *x* verwenden. Daraus würden auf dem Client zwei verschiedene Datenklassen *x1* und *x2* entstehen, die untereinander nicht mehr kompatibel wären.

Beispiel für das Problem

Das folgende Beispiel veranschaulicht das Problem.

Dienst	Operation	Generierte Datenklassen
FlugplanService	public de.WWWings.Flug HoleFlug(long FlugNr)	Flugplanservice.Flug
PassagierverwaltungService	public de.WWWings.PassagierSystem.Passagier HolePassagier(int Nummer)	PassagierverwaltungService.Passagier
BuchungsService	public Buchung BuchungErstellenMitObjekten(Flug f, Passagier p)	BuchungsService.Flug Buchungsservice.Passagier

Tabelle 29.4 Problem bei generierten Webservice-Proxyklassen

Das nachstehende Listing kann nicht funktionieren, weil der BuchungsService nicht in der Lage ist, die von dem FlugplanService und dem PassagierverwaltungService gelieferten Objekte zu verarbeiten. Sie haben zwar die gleiche Struktur, aber einen anderen Namensraum und damit einen anderen Namen:

```
Flugplanservice.FlugplanServiceClient fs = new
Flugplanservice.FlugplanServiceClient("Flugplanservice_TCP");
Flugplanservice.Flug f2 = fs.HoleFlug(101);
Console.WriteLine(f2.flugNr + " fliegt von " + f2.abflugOrt + " nach " + f2.zielOrt + " Anzahl freier
Plätze: " + f2.freiePlaetze);
```

```
PassagierverwaltungService.Passagier p = new de.WWWings.PassagierverwaltungService.Passagierverwaltung
    ServiceClient("PassagierverwaltungService_TCP").HolePassagier(97);
Console.WriteLine("Passagier heißt: " + p._Vorname + " " + p._Nachname);
// Das geht nicht!
// Buchungsservice.Buchung bu = bs.BuchungErstellenMitObjekten(f2, p);
```

Listing 29.32 Probleme mit dem Proxygenerator

Lösung für das Problem

Mit Visual Studio Version 2005 war das Problem nicht lösbar. In Visual Studio 2008 gibt es eine Lösung. Oft geht es aber besser mit dem Kommandozeilenwerkzeug *svcutil.exe*. Der Trick liegt darin, die Metadatenendpunkte aller beteiligten WCF-Dienste zusammen anzugeben, sodass die Proxy- und Datenklassen zusammen generiert werden. Zwar meldet das Kommandozeilenwerkzeug dann einen Fehler (»The global element 'http://schemas.datacontract.org/2004/07/de.WWWings:Flug' has already been declared.«), aber der erzeugte Code funktioniert dennoch.

Ein weiterer Vorteil der Kommandozeilengenerierung ist, dass die erzeugten Datenklassen den gleichen Namensraum bekommen, den sie auch auf dem Server haben. In dem nachstehenden Aufruf von *svcutil.exe* wird auf die Übernahme des Namensraums aber bewusst verzichtet und ein anderer Namensraum benannt, damit die Beispiele übersichtlicher werden und direkt erkennbar ist, ob lokale oder entfernte Klassen angesprochen werden:

```
Svcutil
/t:code
/language:cs
/config:app.config
/namespace:*,de.WWWings.WCF
/out:WWings_WCFProxies.cs
net.tcp://localhost:1234/WWings/metadaten/Flugplanservice
net.tcp://localhost:1234/WWings/metadaten/FlugplanverwaltungService
net.tcp://localhost:1234/WWings/metadaten/Buchungsservice
net.tcp://localhost:1234/WWings/metadaten/PassagierverwaltungService
```

Listing 29.33 Kommandozeilenbefehl zur Erzeugung von gemeinsamen Datenklassen für mehrere WCF-Dienste [ProxyGen.bat]

WICHTIG Die vorgenannten Befehle müssen alle in einer Zeile stehen. Sie wurden im Buch nur zur Übersichtlichkeit umgebrochen!

Möglichkeiten in Visual Studio 2008

In Visual Studio 2008 kann man beim Anlegen einer Referenz auf einen WCF-Dienst (*Add Service Reference*) unter *Advanced* festlegen, dass der Proxygenerator Datentypen aus einer bereits referenzierten Assembly wiederverwenden kann. Leider gilt die Wiederverwendung nur für referenzierte Assemblies im engeren Sinne, nicht für den Programmcode im eigenen Projekt. Das macht es sehr lästig. Diese Funktion in Visual Studio 2008 lässt sich aber dazu nutzen, für Datenklassen gar keine Proxyklassen zu erzeugen, sondern die Originalklassen vom Server zu nutzen. Dies widerspricht zwar etwas dem Prinzip der Entkopplung (und wird daher von einigen Autoren *SOA light* genannt), ist aber in reinen .NET-Szenarien durchaus adäquat. Das Verfahren bietet sogar den Vorteil, dass man eine Geschäftslogik auf einfache Weise wahlweise über die WCF oder rein lokal im gleichen Prozess betreiben kann.

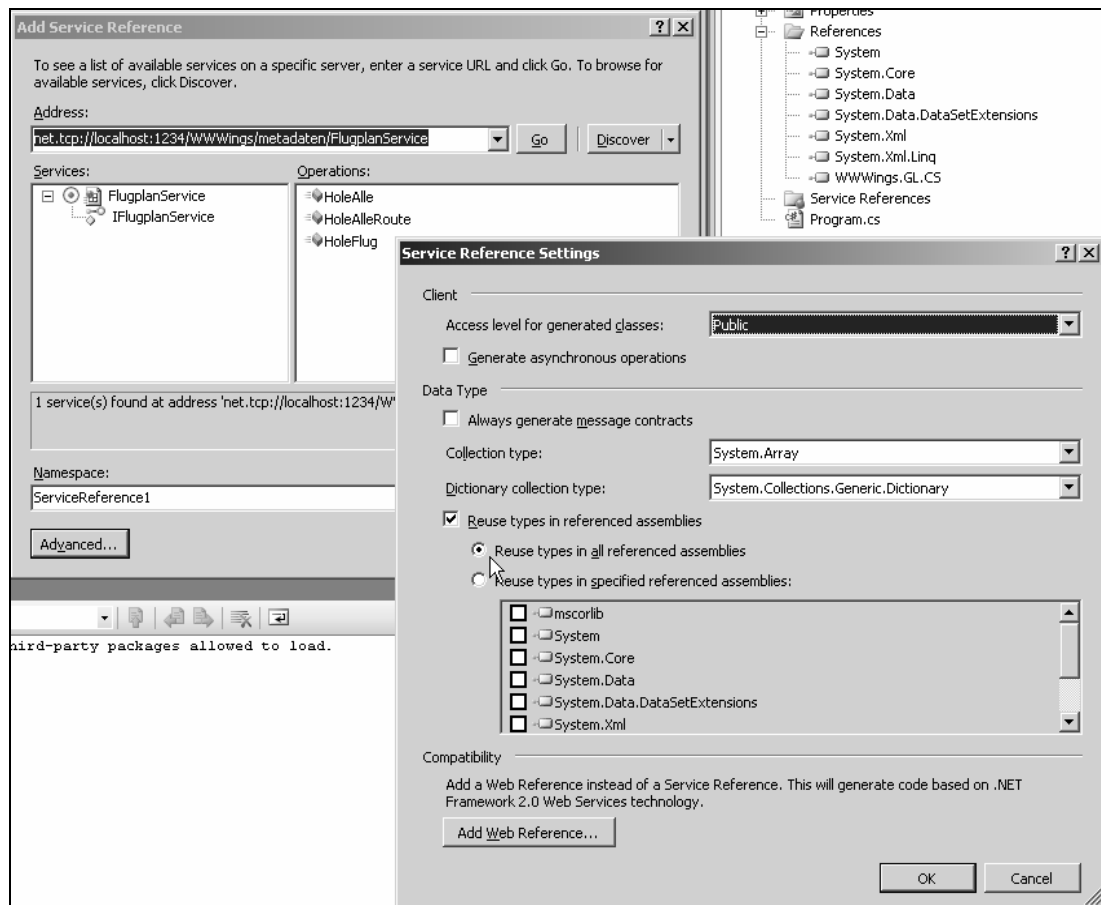


Abbildung 29.38 Wiederverwendung von Datenklassen in bereits referenzierten Assemblies

Ein auf diese Weise generierter Proxy für eine Dienstklasse sieht dann so aus wie in dem folgenden Listing in Ausschnitten gezeigt: Die Methoden geben Objekte einer Klasse zurück, die aus der vom Server bezogenen Geschäftslogikassembly stammen:

```
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.CodeDom.Compiler.GeneratedCodeAttribute("System.ServiceModel", "3.0.0.0")]
public partial class FlugplanServiceClient :
    System.ServiceModel.ClientBase<ConsoleApplication2.ServiceReference1.IflugplanService>,
    ConsoleApplication2.ServiceReference1.IflugplanService {

    ...

    public de.WWWings.Flug[] HoleAlle() {
        return base.Channel.HoleAlle();
    }

    public de.WWWings.Flug[] HoleAlleRoute(string von, string nach) {
        return base.Channel.HoleAlleRoute(von, nach);
    }

}
```

```
public de.WWWings.Flug HoleFlug(long FlugNr) {  
    return base.Channel.HoleFlug(FlugNr);  
}  
}
```

Listing 29.34 Generierter Clientproxy, der sich die Datenklassen mit dem Server teilt

Multithreading-Verhalten

Eine wichtige Einstellung in [ServiceBehavior] ist die Multithreading-Einstellung (ConcurrencyMode):

- Die Standardeinstellung ist `Single`, das heißt nur ein Thread befindet sich gleichzeitig in einer Dienstinstanz. Alle weiteren Aufrufe landen in der Warteschlange.
- Optional kann man `Multiple` einstellen. Dies bedeutet, dass mehrere Instanzen in einem Thread sein können, und der Entwickler selbst die Konsistenz gewährleisten muss.
- `Reentrant` ist eine Zwischenlösung, bei der ein zweiter Thread die Dienstinstanz nur betreten kann, wenn der Dienst einen anderen Dienst aufruft.

ACHTUNG Die Multithreading-Einstellungen sind besonders entscheidend für Singleton-Dienste. Wenn man diese Dienste im `ConcurrencyMode = Single` betreibt, bremsen sich die Clients gegenseitig sehr stark aus. Für Single Call-Dienste (PerCall) ist die Einstellung für `ConcurrencyMode` irrelevant, da es ohnehin immer eine neue Instanz bei jedem Aufruf gibt.

Leistung (Performanz)

Bei der Auswahl der Bindungen sollten Sie die Ergebnisse von Microsofts offiziellem WCF-Leistungstest [MSDN20] in Betracht ziehen. Zusammenfassend kann man sagen:

- WCF-Webservices sind etwas schneller als ASP.NET-basierte Webservices.
- WCF-Dienste mit Sicherheitsmechanismen auf Transportebene (SSL) sind langsamer als Dienste ohne Sicherheitsmechanismen.
- WCF-Dienste mit Sicherheitsmechanismen auf Nachrichtenebene (WS-Security) sind wesentlich langsamer als Dienste ohne Sicherheitsmechanismen.
- TCP-basierte Dienste sind wesentlich schneller als HTTP-basierte Dienste.

Weitere Möglichkeiten von WCF

Folgende Möglichkeiten von WCF können hier nur kurz erwähnt werden:

- Die Serialisierung kann über die Schnittstellen `ISerializable` und `IXmlSerializable` genau gesteuert werden.
- Wenn anstelle einer Instanz einer in der Operation genannten Datenklasse eine Instanz einer abgeleiteten Klasse übergeben werden soll, müssen die Klassen mit `[ServiceKnownType(typeof(Klasse))]` deklariert werden. Die Annotation wird zusammen mit `[OperationContract]` genannt.

- Wenn man vor der Anforderung steht, einen Typ serialisieren zu müssen, der die Voraussetzungen der Serialisierung nicht erfüllt, man diesen Typ aber nicht ändern kann, muss man ein so genanntes Serialisierungssurrogat entwickeln mit der Schnittstelle *IDataContractSurrogate*.
- Mithilfe so genannter Nachrichtenverträge kann man eine genaue Kontrolle über den Aufbau der zwischen Client und Server ausgetauschten Nachrichten erlangen. Damit lässt sich z.B. festlegen, welche Informationen im Nachrichtenkopf und welche im Nachrichteninhalte landen. Hierzu verwendet man die Annotationen *[MessageContract]*, *[MessageHeader]* und *[MessageBodyMember]*.
- Über *[FaultContract]* kann man spezielle Nachrichten für den Fehlerfall definieren.
- Noch mehr Kontrolle erhält man, indem man definiert, dass Operationen die Daten in Form von Stream- oder Message-Objekten liefern sollen.
- Ein Workflow Service ist ein WCF-Dienst, der mit der Windows Workflow Foundation (WF) implementiert ist, siehe Buch [HS02], Kapitel »Windows Workflow Foundation (WF)« (Aktivität *ReceiveActivity*).
- Im Standardmodus können Nachrichten erst vom Empfänger gelesen werden, wenn sie komplett übermittelt wurden. Für große Nachrichten kann der Streamingmodus aktiviert werden, sodass der Empfänger schon vor Eingang der kompletten Nachricht mit dem Verarbeiten beginnen kann.
- Für Transportprotokolle, die keine eingebaute Zuverlässigkeitskontrolle haben (also HTTP, aber nicht TCP), bietet die WCF die Unterstützung für den Standard *WS-ReliableMessaging*, mit dem Nachrichten bei Verlust wiederholt und doppelte Nachrichten eliminiert werden, und auch die Reihenfolge des Eingangs der Nachrichten sichergestellt wird.
- Unterstützung für Transaktionen mit *WS-AtomicTransaction* und *OLE Transactions*.
- Über die WCF-Bindungen *NetMsmqBinding* bzw. *MsmqIntegrationBinding* können Nachrichtenwarteschlangen auf Basis des Microsoft Message Queuing-Dienstes verwendet werden. Nachrichtenwarteschlangen kommen zum Einsatz, wenn nicht sichergestellt ist, dass der Server zum Zeitpunkt des Absendens verfügbar ist, es aber dennoch erforderlich ist, dass der Client die Nachricht unabhängig davon absenden kann.
- WCF-Dienste können durch einen WMI-Provider (WMI-Namensraum *root/ServiceModel*) überwacht werden.
- Schutz gegen Angriff durch Lastbeschränkungen und Verhindern von Replay-Angriffen.
- Weiterleiten von Aufrufen mit WS-Addressing.

Fazit zu WCF

Die WCF ist zweifelsfrei eine mächtige und sehr flexible Kommunikationsinfrastruktur, welche die effiziente Kommunikation zwischen zwei .NET-Anwendungen und die plattformübergreifende Kommunikation auf Basis von Webservices und WS-* Standard auf eine gemeinsame Basis stellt. Viele Kommunikationseigenschaften können zur Betriebszeit ohne Neukompilierung der Anwendung geändert werden.

Jedoch ist auch Kritik angebracht: Microsoft propagiert die WCF als Vereinigung und Nachfolgetechnologie von ASP.NET-Webservices (ASMX) und .NET Remoting. Auch wenn die WCF einige Konzepte von .NET Remoting übernommen hat, ist es allenfalls ein echter Nachfolger von ASMX.

Als problematisch anzusehen ist, dass WCF nicht kompatibel mit bestehenden .NET-Remoting-Endpunkten ist, das heißt, es existiert dafür kein entsprechendes WCF-Bindungsprofil. WCF mischt sich zwar nicht in die Kommunikation von .NET Remoting ein, sodass bestehende .NET Remoting-Anwendungen weiterhin funktionieren; jedoch ist eine Kommunikation zwischen einem WCF-Endpunkt und einem .NET Remoting-Endpunkt nicht ohne erhebliche Änderungen im Programmcode möglich.

Microsoft propagiert die WCF als technische Infrastruktur zum Aufbau von serviceorientierten Architekturen, in der kein Raum mehr für die von .NET Remoting geförderte enge Bindung durch verteilte Objekte ist. Wünschenswert wäre gewesen, den Entwickler hier nicht zu bevormunden und auch diese Art verteilter Systeme weiterhin zuzulassen.