15.–18. 09. 2008
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Ferngespräch
Einführung in die Windows Communication Foundation (WCF)

Thomas Haug

MATHEMA Software GmbH

Agenda

- Anforderungen
- Eigenschaften
- Architektur Überblick
  - Architekturüberblick
  - WCF ABC
  - Clientseitiges Programmiermodell
  - Serverseitiges Programmiermodell
- Fazit (und .Net 3.5 und Visual Studio 2008 Neuerungen)
- Literaturhinweis

Agenda

- Anforderungen
- Eigenschaften
- Architektur Überblick
  - Architekturüberblick
  - WCF ABC
  - Clientseitiges Programmiermodell
  - Serverseitiges Programmiermodell
- Fazit (und .Net 3.5 und Visual Studio 2008 Neuerungen)
- Literaturhinweis

WCF – 'Anforderungen'

- Clients mit ‚Backend' Services verbinden
- ‚Backend' Services an andere ‚Backend' Services verbinden
- Soll Technologie Neutral sein, insbesondere auf Netzwerkebene
- Bereitstellen von 'Querschnittsdiensten':
  - Service hosting
  - Instanzmanagement und Nebenläufigkeitssteuerung
  - Unterstützung für Transaktionen
  - Unterstützung für Security
  - Unterstützung für Reliability
  - Asynchrone Kommunikation

Agenda

- Anforderungen
- Eigenschaften
- Architektur Überblick
  - Architekturüberblick
  - WCF ABC
  - Clientseitiges Programmiermodell
  - Serverseitiges Programmiermodell
- Fazit (und .Net 3.5 und Visual Studio 2008 Neuerungen)
- Literaturhinweis

WCF - Eigenschaften (I)

- Baut auf .Net 2.0 auf (als Add-on verfügbar)
- Integraler Bestandteil von .NET 3.0 (Release November 2006)
- Nachrichtenstruktur entspricht SOAP Struktur
- Implementiert WS-* Standards (wie WS-AT, WS-Security, WS-Reliable Messaging, WS-Addressing)
- Einheitliches Programmiermodell, vereinigt MS Technologien:
  - .NET Remoting
  - ASP.NET Web Services
  - Enterprise Services (COM+)
  - MSMQ

WCF - Eigenschaften (II)

- Unterstützt Transaktionen über
  - OleTransactions (in Microsoft Umgebungen)
  - WS AtomicTransactions (im heterogenen Umfeld)
- Unterstützt Sicherheits-Mechanismen
  - Transport
    - Authentisierung, Integrität, Vertraulichkeit
  - Autorisierung
- Unterstützung von Reliability
  - Exactly-Once Semantik
  - Reihenfolge wird eingehalten

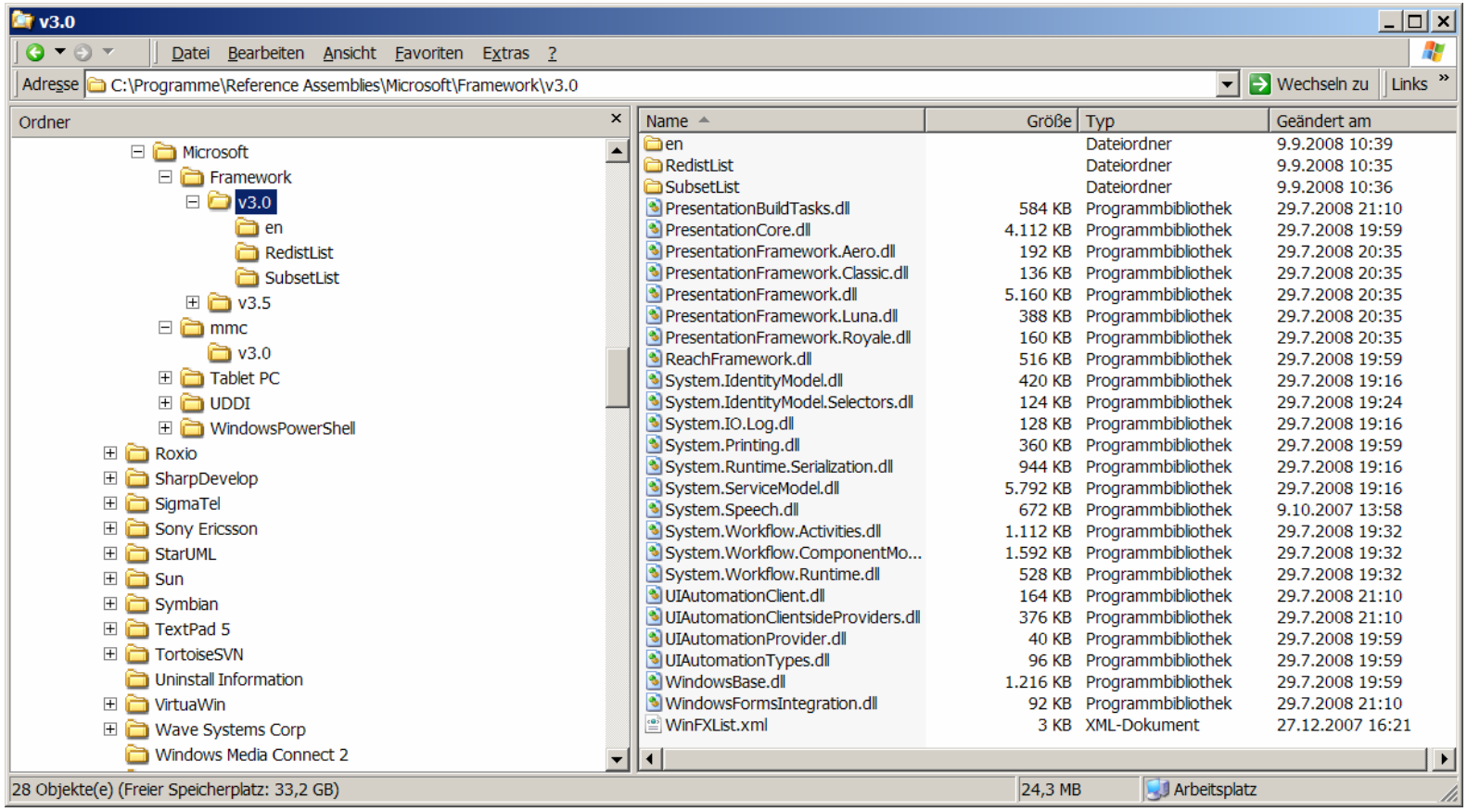
WCF - Eigenschaften (III)

- Unterstützt verschiedene Nachrichtenaustausch Muster (Message Exchange Patterns - MEPs)
  - Request/Reply (a.k.a half-duplex)
  - Duplex (mit Callback Vertrag)
  - One-way („Datagram“)
  - Messaging mit MSMQ

- “Kern-Namespace” ist `System.ServiceModel`
- `System.Runtime.Serialization` wird für Datenverträge verwendet

WCF - Eigenschaften (IV)

- Installation



WCF - 'Versprechungen' (I)

- Einfachheit
  - Ähnliche Komplexität wie andere Microsoft Verteilungstechnologien
  - Komplexität kann, wenn benötigt, gesteigert werden (z.B. Erweiterbarkeit des Frameworks)
- Wartbarkeit
  - Einheitliches API, d.h. es müssen keine unterschiedlichen Technologien wie ASP.Net, .Net Remoting erlernt werden.

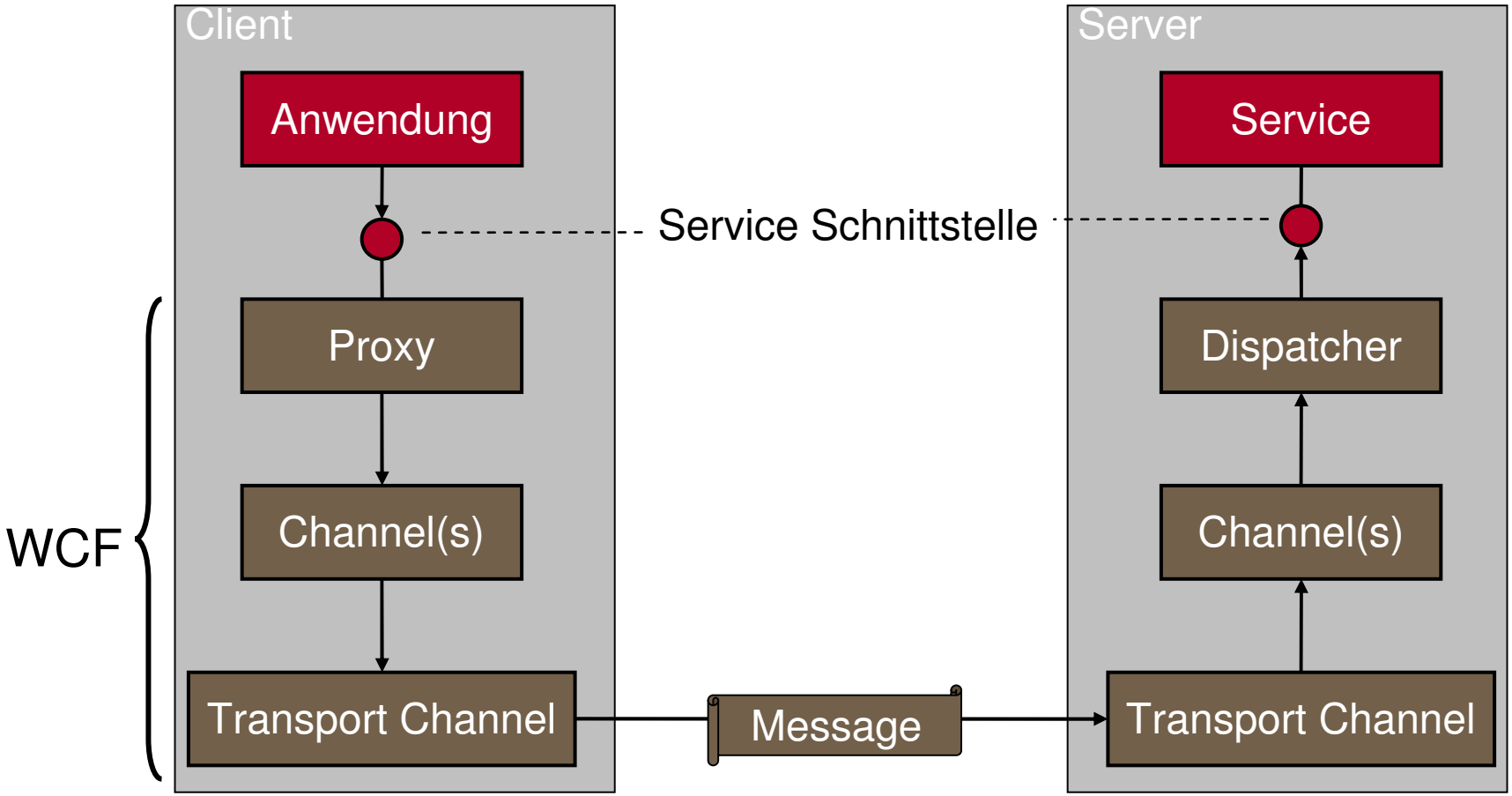
WCF - 'Versprechungen' (II)

- Flexibilität
  - Eine Vielzahl von 'Eigenschaften' des Servers lassen sich per Konfiguration einstellen und erweitern ohne das Code verändert werden muss
- Mächtigkeit
  - Die Eigenschaften aller MS Verteilung-Technologien wurden in ein Framework verschmolzen.
  - Interoperabilität und Service-Orientierung standen bei der Konzeption des Frameworks im Vordergrund

Agenda

- Anforderungen
- Eigenschaften
- Architektur Überblick
  - Architekturüberblick
  - WCF ABC
  - Clientseitiges Programmiermodell
  - Serverseitiges Programmiermodell
- Fazit (und .Net 3.5 und Visual Studio 2008 Neuerungen)
- Literaturhinweis

WCF - Architektur Überblick (I)

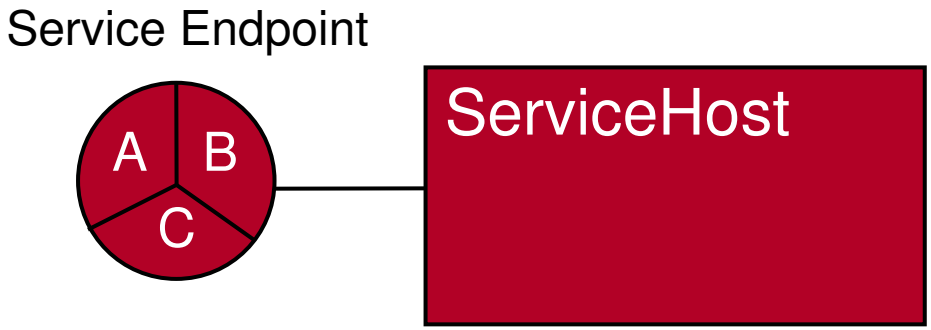


Agenda

- Anforderungen
- Eigenschaften
- Architektur Überblick
  - Architekturüberblick
  - WCF ABC
  - Clientseitiges Programmiermodell
  - Serverseitiges Programmiermodell
- Fazit (und .Net 3.5 und Visual Studio 2008 Neuerungen)
- Literaturhinweis

WCF – Endpoints

- Services werden über sog. Endpoints bereitsgestellt
- Endpoints bestehen aus drei Bestandteile
  - Address Wo ist der Service zu finden
  - Binding Wie ist der Dienst anzusprechen
  - Contract Welche Operationen bietet der Dienst an



WCF – Endpoint Adressen

- Beschreibt den ‚Ort' eines Service
- Legt das Transport-Protokoll fest, WCF unterstützt hierbei
  - HTTP
  - TCP
  - IPC
  - MSMQ
  - Peer network
- Beispiel

```
http://localhost:8080/MyService
net.tcp://localhost:8888/Myservice
net.pipe://localhost/NamedPipe
net.msmq://hostName/ServiceName
```

WCF – Endpoint Bindings (I)

- Legt das ‚Kommunikationsmuster' zwischen Client und Server fest:
  - Nachrichten Enkodierung (Binär, Text oder MTOM)
  - Timeout Verhalten
  - Zuverlässigkeit (Reliability)
  - Transaktionales Verhalten (Propagieren und Nutzen von Transaktionsinformationen)
  - Security
- Kann programmatisch oder per Konfiguration festgelegt werden

WCF – Endpoint Bindings (II)

- WCF (3.0) unterstützt 9 unterschiedliche Typen:
  - NetTcpBinding
  - NetNamedPipeBinding
  - NetMsmqBinding
  - MsmqIntegrationBinding
  - NetPeerTcpBinding
  - BasicHttpBinding
  - WS*(2007)*HttpBinding
  - WSDualBinding
  - WS*(2007)*FederationHttpBinding
- .Net 3.5 WebHttpBinding mit XML (POX) und JSON Formattern, des Weiteren RSS und ATOM
- WCF unterstützt das Definieren neuer sog. Custom Bindings

WCF – Endpoint Bindings und Interoperabilität

Binding Name	Transport	Encoding	Interop.
NetTcpBinding	TCP	Binary	nein
NetNamedPipeBinding	IPC	Binary	nein
NetMsmqBinding	MSMQ	Binary	nein
MsmqIntegrationBinding	MSMQ	Binary	nein
NetPeerTcpBinding	P2P	Binary	nein
BasicHttpBinding	HTTP / HTTPS	**Text** / MTOM	ja
WsHttpBinding	HTTP / HTTPS	**Text** / MTOM	ja
WsDualBinding	HTTP	**Text** / MTOM	ja
WsFederationHttpBinding	HTTP / HTTPS	**Text** / MTOM	ja
WebHttpBinding	HTTP / HTTPS	Xml / JSON	ja

WCF – Endpoint Contracts (I)

- `ServiceContract` am Beispiel

```
1:  using System.ServiceModel;
2:   namespace WCFDemo {
3:     [ServiceContract (Name= "..", Namespace = "..")]
4:     public interface ISimpleDemoService {
5:        [OperationContract (IsOneWay = false)]
6:        [FaultContract(typeof(ArgumentException)]
7:        void CallMe(string inputparam);

8:        [OperationContract]
9:        void StoreCustomer(Kunde kunde);
10:     }
11:  }
```

- Schnittstelle muss mindestens eine Methode mit `OperationContract` Attribut aufweisen

WCF – Endpoint Contracts (II)

- `DataContract` am Beispiel

```
1:  using System.Runtime.Serialization;
2:  namespace WCFDemo {
3:     [DataContract]
4:     public class Kunde {
5:        [DataMember]
6:        private string Name;
7:        private int KundenID;
8:     }
9:  }
```

- In .Net 3.5 kann auf diese Attribute verzichtet werden

WCF – Endpoint Contracts (III)

- Exkurs: Fault Contract Implementierung im Service

```
1:   public void CallMe(string inputparam) {
2:      if (inputparam.Equals("")) {
3:         throw new FaultException<ArgumentException>(
                     new ArgumentException("Param wrong",
                                           "inputparam"));
4:      }
        Exception ist im
        Vertrag definiert

5:      if (inputparam == null) {
6:         throw new FaultException<ArgumentNullException>(
                     new  ArgumentNullException("…"));
7:      }
8:      …
9:   }
        Exception ist nicht
        im Vertrag definiert
```

WCF – Endpoint Contracts (IV)

- Exkurs: Fault Contract Implementierung im Client

```
 1:  try {
 2:     client.CallMe("");
 3:  }
 4:  catch (FaultException<ArgumentException> argExc) {
 5:     Console.WriteLine("ArgumentException has
                          been caught : {0}",
                          argExc.Message);

 6:  }
 7:  try {
 8:     client.CallMe(null);
 9:  }
10:  catch (FaultException argEx) {
11:     Console.WriteLine("Unknown FaultException has been
                           caught : {0}",argEx.Message);
12.  }
```

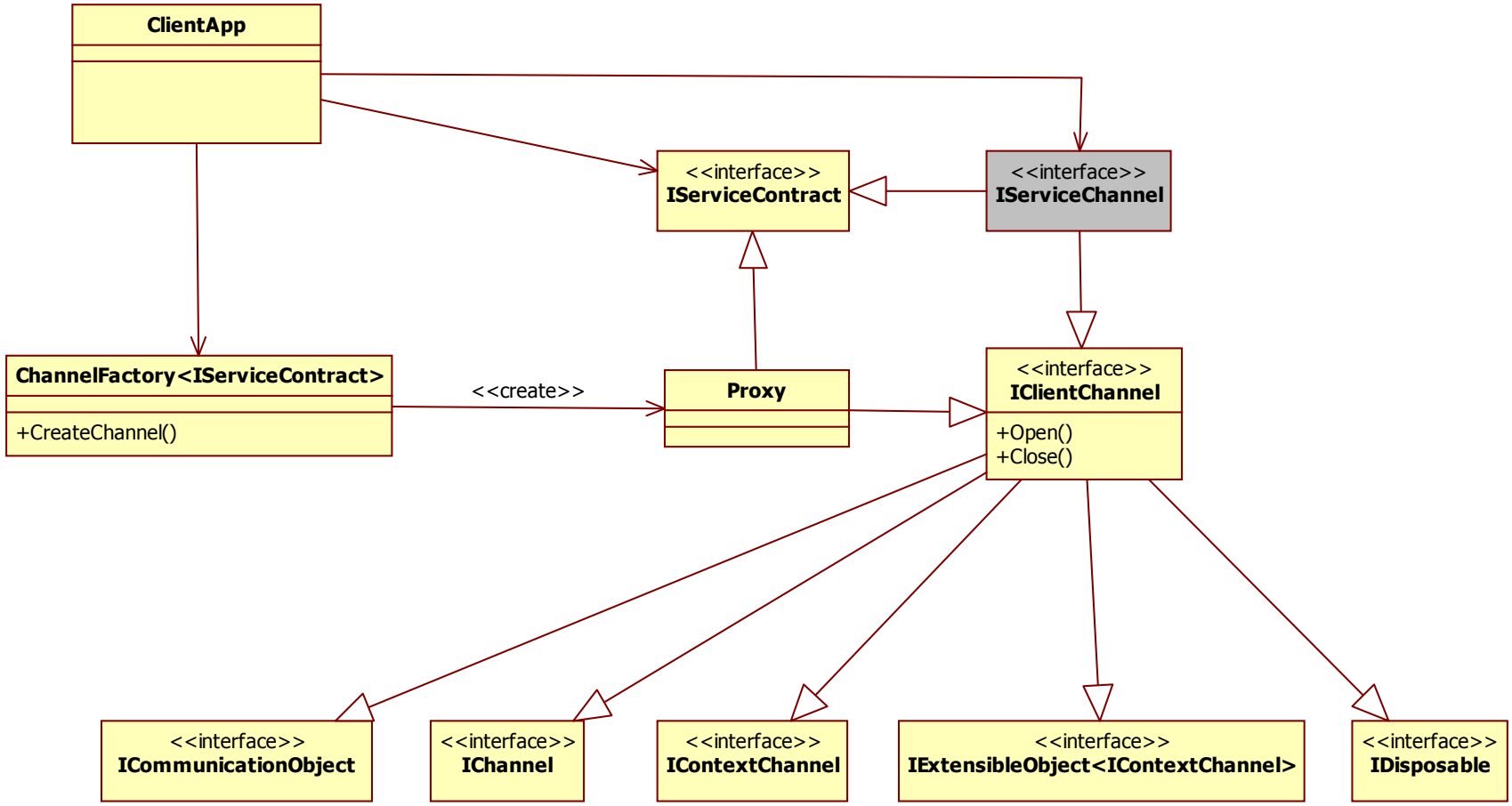
Exception ist im Vertrag definiert

Exception ist nicht im Vertrag definiert, deshalb generische FaultException

Agenda

- Anforderungen
- Eigenschaften
- Architektur Überblick
  - Architekturüberblick
  - WCF ABC
  - Clientseitiges Programmiermodell
  - Serverseitiges Programmiermodell
- Fazit (und .Net 3.5 und Visual Studio 2008 Neuerungen)
- Literaturhinweis

WCF – Client Architektur (I)



WCF - Client Architektur (II)

- `ChannelFactory`
  - Ist für die Erzeugung der grundlegenden WCF Laufzeitumgebung verantwortlich
- `IClientChannel`
  - Bietet die Kernfunktionalität zur Kommunikation mit dem (entfernten) Service an und erweitert die folgenden Schnittstellen

- `ICommunicationObject` : Bietet Methoden zum Steuern der Kommunkation
- `IContextChannel` : bietet Methoden zum Setzen von Timeouts, Lesen der SessionID
- `IExtensibleObject`: Bietet Methoden, um den InstanceContext zu erweitern

WCF - Client Architektur (III)

- Beispiel

```
 1:  using System.ServiceModel;
 2:  namespace WCFClient {
 3:     class Client {
 4:        static void Main(string[] args) {
 5:           IMyChannel client;
 6:           try {
 7:              EndpointAddress eAddress =
                                    new EndpointAddress(...);
 8:              NetTcpBinding tcpBinding =
                                    new NetTcpBinding();

 9:              client =
      ChannelFactory<IMyChannel>.CreateChannel(tcpBinding,
                                                  eAddress);

10:              client.CallMe("Hallo, ein client ruft");
11:              client.Close();
```

WCF - Client Architektur (IV)

- Beispiel Client per Konfiguration (I)

```
 1: <system.serviceModel>
 2:   <bindings>
 3:     <netTcpBinding>
 4:         <binding name="NetTcpBinding_ISimpleDemoService"
 5:             <!-- die Konfiguration …. -->
 6:         </binding>
 7:     </netTcpBinding>
 8:   </bindings>
 9:   <client>
10:     <endpoint address="net.tcp://localhost:8888/TcpBinding"
                  binding="netTcpBinding"
                  bindingConfiguration="NetTcpBinding_ISimpleDemoService"
                  contract="DemoService.ISimpleDemoService"
                  name="NetTcpBinding_ISimpleDemoService">
11:           <identity>
12:               <userPrincipalName value="MATHEMA\haug" />
13:           </identity>
14:     </endpoint>
15:   </client>
16: </system.serviceModel>
```

WCF - Client Architektur (V)

- Beispiel Client per Konfiguration (II)

```
1:  SimpleDemoServiceClient client = null;
2:  client = new SimpleDemoServiceClient("NetTcpBinding_ISimpleDemoService");
3:  client.CallMe("Hallo ein client ruft");
```

Agenda

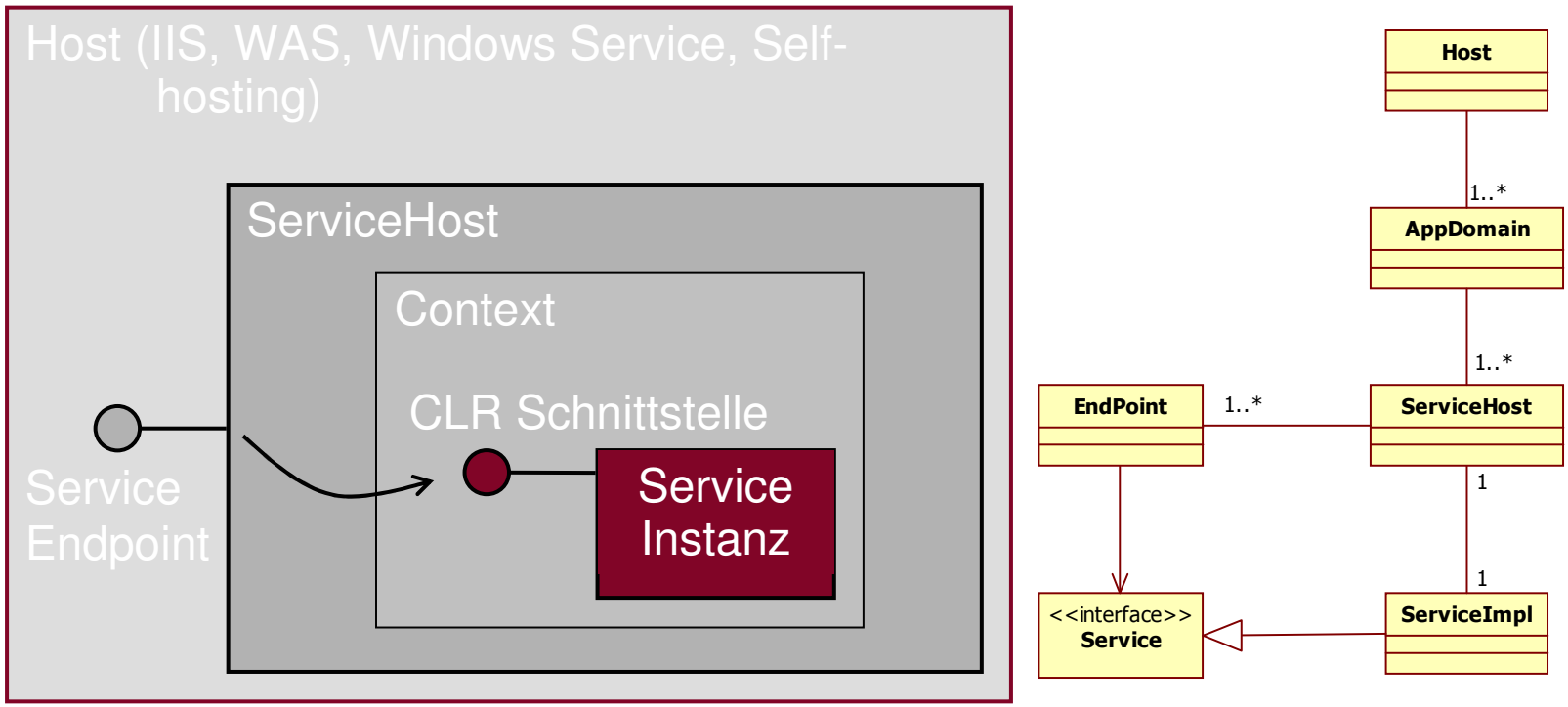
- Anforderungen
- Eigenschaften
- Architektur Überblick
  - Architekturüberblick
  - WCF ABC
  - Clientseitiges Programmiermodell
  - Serverseitiges Programmiermodell
    - Hosting
    - Service bereitstellen
    - Service Typen
    - Nebenläufigkeit
    - Throttling
- Fazit (und .Net 3.5 und VS 2008 …)
- Literaturhinweis

Agenda

- Anforderungen
- Eigenschaften
- Architektur Überblick
  - Architekturüberblick
  - WCF ABC
  - Clientseitiges Programmiermodell
  - Serverseitiges Programmiermodell
    - Hosting
    - Service bereitstellen
    - Service Typen
    - Nebenläufigkeit
    - Throttling
- Fazit (und .Net 3.5 und VS 2008 …)
- Literaturhinweis

WCF - Serverseitiges Hosting

- Der Host stellt die Laufzeitumgebung für die WCF Dienste zur Verfügung



WCF – Einen Service bereitstellen (I)

- Programmatisches Erzeugen von Endpunkten

```
1: ServiceHost serviceHost =
                new ServiceHost( typeof(DemoServiceImpl);

2:  EndpointAddress address = new EndpointAddress(
        new Uri( "net.tcp://localhost:8888/MyService“ ));

3:  Binding binding = new NetTcpBinding();

4:  ContractDescription contract =
  ContractDescription.GetContract(typeof(ISimpleDemoService));

5:  ServiceEndpoint endPoint = new ServiceEndpoint(contract,
                                                   binding,
                                                   address);
6:  serviceHost.Description.Endpoints.Add(endPoint);

7:  serviceHost.Open(); // auf eingehende Requests lauschen
```

WCF – Einen Service bereitstellen (II)

- Erzeugen von Endpunkten per Konfiguration

```
<system.serviceModel>
   <services>
        <service name="WCFDemo.DemoServiceImpl">
            <endpoint
                address = "http://localhost:8000/MyService"
                binding = "basicHttpBinding"
                contract = "WCFDemo.ISimpleDemoService"/>
       </service>
    </services>
    <bindings>
       <basicHttpBinding>
            <binding name="basicHTTP"
                     transferMode="Buffered"
                     messageEncoding="Mtom">
            </binding>
       </basicHttpBinding>
    </bindings>
</system.serviceModel>
```

WCF – Einen Service bereitstellen (III)

- Erzeugen von Endpunkten per Konfiguration (II)

```
1: Uri tcpAddress =
                    new Uri("net.tcp://localhost:7777");

2:  ServiceHost serviceHost =
                new ServiceHost(typeof(DemoServiceImpl),
                                tcpAddress);

3:  serviceHost.Open();
```

WCF – Metadaten bereitstellen und abfragen

- Servicebeschreibungen (Metadaten) werden mittels WSDL beschrieben
- Können über HTTP-GET oder über sog. MEX Endpoints bereitgestellt werden
  - Beispiel für HTTP-GET Variante

```
1:  ServiceMetadataBehavior smdBehavior =
                  new ServiceMetadataBehavior();
2:  smdBehavior.HttpGetEnabled = true;
3: serviceHost.Description.Behaviors.Add(smdBehavior);
```

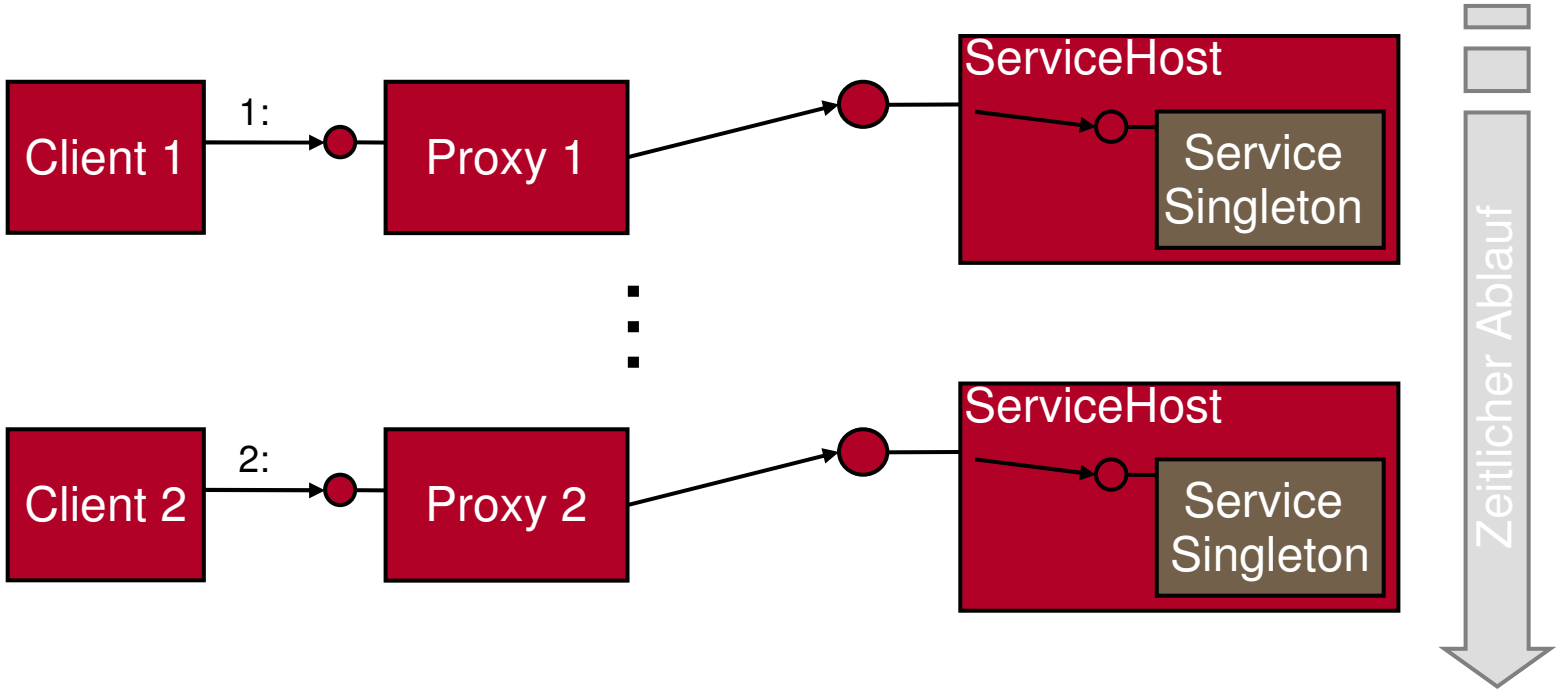
  - Abrufen über z.B.
    `http://localhost:8000/MyService?wsdl`

Agenda

- Anforderungen
- Eigenschaften
- Architektur Überblick
  - Architekturüberblick
  - WCF ABC
  - Clientseitiges Programmiermodell
  - Serverseitiges Programmiermodell
    - Hosting
    - Service bereitstellen
    - Service Typen
    - Nebenläufigkeit
    - Throttling
- Fazit (und .Net 3.5 und VS 2008 …)
- Literaturhinweis

WCF – `Single` Services (I)

- Jeden Service Aufruf verwendet die gleiche Service Instanz



WCF – `Single` Services (II)

- Single Services werden mittels `InstanceContextMode.Single` deklariert

- Beispiel Service Implementierung

```
1:  [ServiceBehavior(InstanceContextMode =
                InstanceContextMode.Single)]
2:  public class DemoServiceImpl :
                          ISimpleDemoService,
                          IDisposable {
      // ... Service Implementierung
3:... }
```

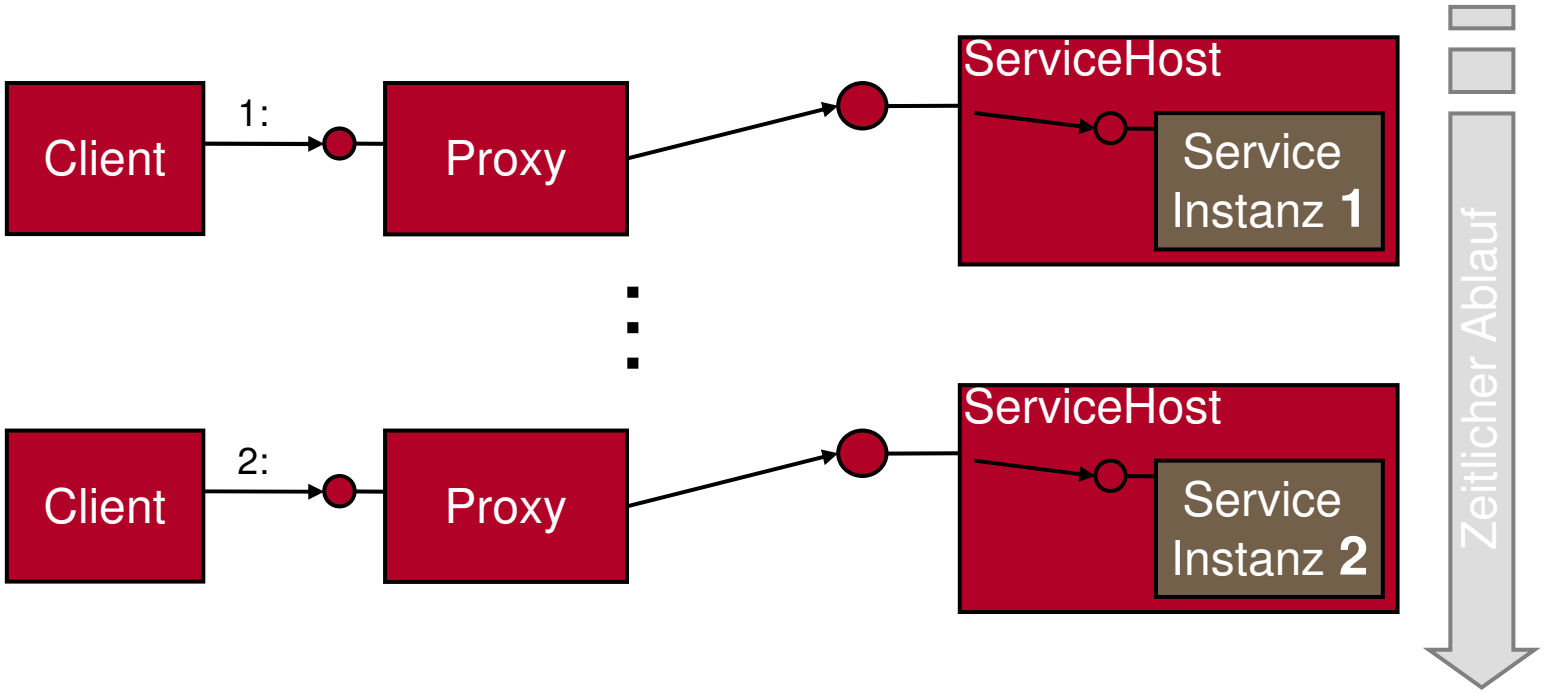
WCF – `Single` Services (III)

- Instanz wird beim Herunterfahren des ServiceHosts zerstört
- Das Singleton muss sich ‘selbstständig’ um die Synchronisation gemeinschaftlich genutzter Ressourcen kümmern!
- ServiceHost kann direkt mit Singleton Instanz initialisiert werden

```
1: DemoServiceImpl singleton =
                        new DemoServiceImpl();
2: singleton.initialisiereDich(...);
3: ServiceHost host =
     new ServiceHost(singleton);
```

WCF – `PerCall` Services (I)

- Für jeden Service Aufruf wird eine neue Instanz erzeugt



WCF – `PerCall` Services (II)

- Single Services werden mittels
  `InstanceContextMode.PerCall`
  deklariert
- Beispiel Service Implementierung

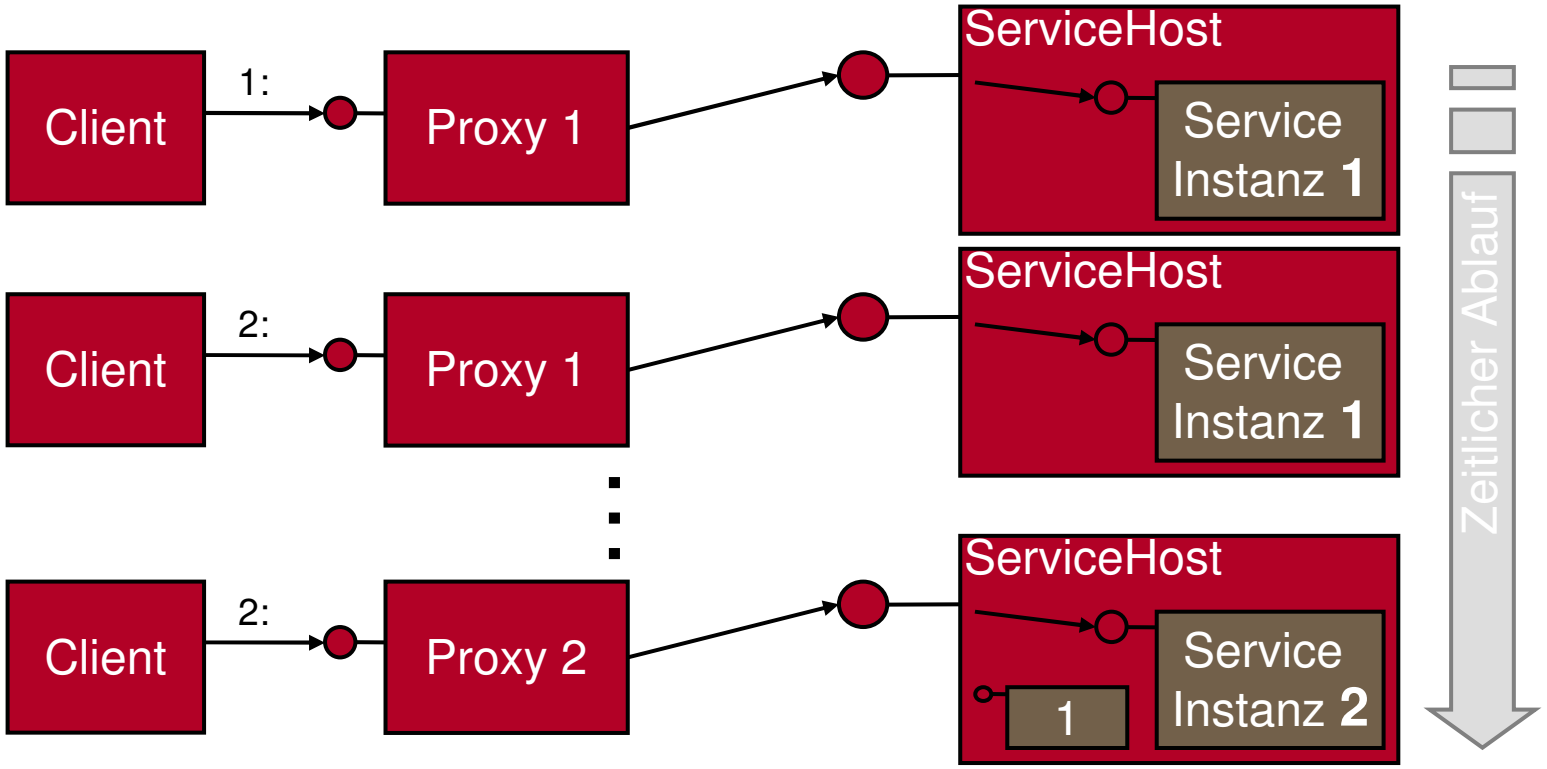
```
1:  [ServiceBehavior(InstanceContextMode =
        InstanceContextMode.PerCall)]
2:  public class DemoServiceImpl :
                    ISimpleDemoService,
                    IDisposable {
      // ... Service Implementierung
 3:  }
```

WCF – `PerCall` Services (III)

- Instanz wird nach Methodenaufruf zerstört
  - Deshalb keine Background Threads und keine async Callbacks zu Instanz verwenden
- Vergleichbar mit EJB Stateless Session Beans, aber kein Pooling

WCF – `PerSession` Services (I)

- Für jeden neuen Client-seitigen Proxy (channel) wird eine neue Instanz erzeugt



WCF – `PerSession` Services (II)

Single Services werden mittels

`InstanceContextMode.PerSession`

deklariert

- Beispiel Service Implementierung

```
1:  [ServiceContract(SessionMode =
          SessionMode.Allowed |
          SessionMode.Required)
2:  public interface ISimpleDemoService { ... }

3:  [ServiceBehavior(InstanceContextMode =
        InstanceContextMode.PerSession)]
4:  public class DemoServiceImpl :
                              ISimpleDemoService,
                              IDisposable {..}
```

WCF – `PerSession` Services (III)

- `SessionMode`
  - Wird im `ServiceContract` festgelegt
  - `SessionMode.Allowed`
    - Ist Default Konfiguration
    - Transportsessions oder deren Simulation bei WS* Bindings werden erlaubt, sind aber nicht erforderlich
  - `SessionMode.NotAllowed`
    - Es werden keine Transportsessions oder deren Simulation erlaubt, z.B TCP und IPC Binding kann dann nicht mehr verwendet werden
  - `SessionMode.Required`
    - Es ist eine TransportSession oder deren Simulation erforderlich, z.B. WS* ohne Security oder ohne Reliability ist nicht erlaubt

WCF – `PerSession` Services (IV)

- Ist der Default InstanceContext Modus
- Zur Unterstützung von Sessions müssen Binding, Service-Behaviour und der SessionMode im ServiceContract korrekt gesetzt sein.
  - HttpBasicBinding unterstützt keine Sessions

WCF – `PerSession` Services (V)

- Binding und Session Mode Kombinationen

Binding	**Instance Context**	**SessionMode**	**‚Ergebnis Modus‘**
netTcpBinding	PerSession	Allowed/Required	PerSession
netNamedPipeBinding	PerSession	Allowed/Required	PerSession
basicHttpBinding	PerSession	Allowed/NotAllowed	**PerCall**
Ws* ohne Security/ ohne Reliability	PerSession	Allowed/NotAllowed	**PerCall**
Ws* mit Security oder Reliability	PerSession	Allowed/Required	PerSession
Ws* mit Security oder Reliability	PerSession	NotAllowed	**PerCall**

→ Bei falscher Kombination wird die Semantik des Service verändert!

WCF – `PerSession` Services (VI)

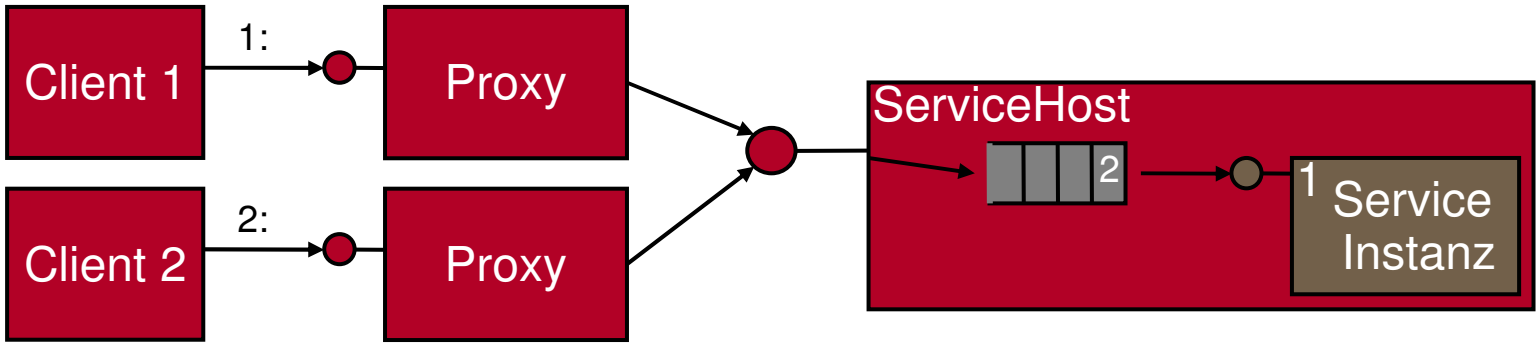
- Sessions beenden
  - Am Proxy wird die `Close()` Methode aufgerufen
  - Eine mit `OperationContract.isTerminating = true` markierte Methode wird aufgerufen (`SessionMode.Required` erforderlich)
  - Timeout
- Sobald die Session terminiert, wird die Service Instanz zerstört.
- Die Service Methoden können auf die SessionID zugreifen
  - `OperationContext.Current.SessionId`
- Entsprechen EJB Stateful Session Beans

Agenda

- Anforderungen
- Eigenschaften
- Architektur Überblick
  - Architekturüberblick
  - WCF ABC
  - Clientseitiges Programmiermodell
  - Serverseitiges Programmiermodell
    - Hosting
    - Service bereitstellen
    - Service Typen
    - Nebenläufigkeit
    - Throttling
- Fazit (und .Net 3.5 und VS 2008 …)
- Literaturhinweis

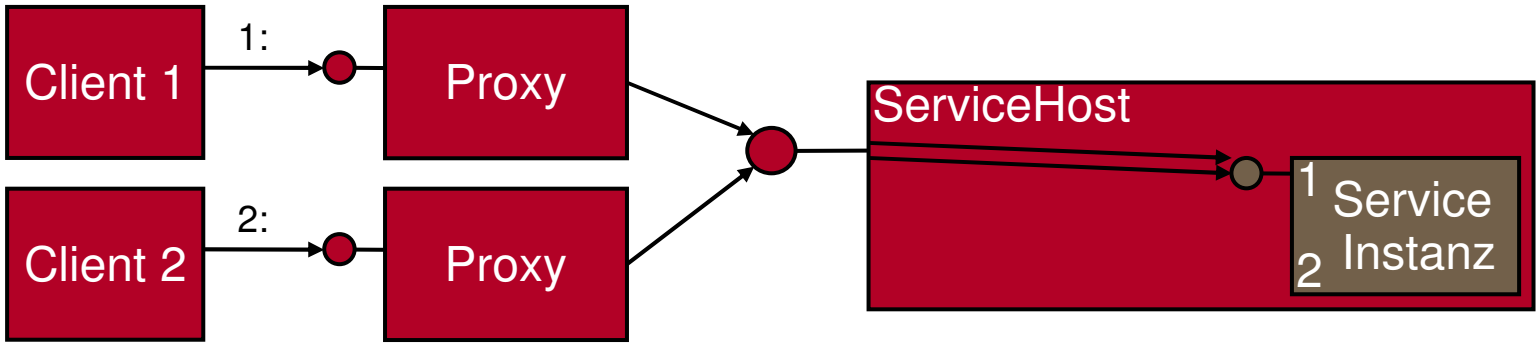
WCF – Nebenläufigkeit in Services (I)

- `ConcurrencyMode.Single` (Default)
  - WCF stellt automatische Synchronisierung der ankommenden Aufrufe sicher
  - Nur ein „*WCF-WorkerThread*“ darf sich in der Service Instance befinden
  - Timeout gilt auch für Aufrufe, die in Warteschlange liegen



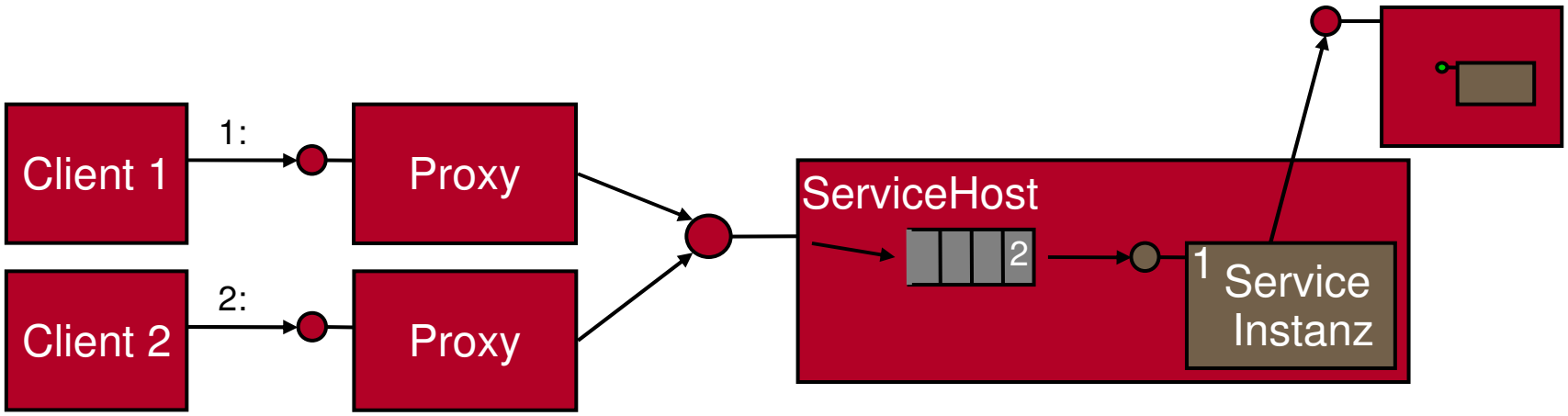
WCF – Nebenläufigkeit in Services (II)

- `ConcurrencyMode.Multiple`
  - Ankommende Aufrufe können parallel die Service Instanz betreten
  - Service Entwickler muss sich um die Synchronisation von Zuständen und gemeinschaftlich genutzte Ressourcen kümmern



WCF – Nebenläufigkeit in Services (III)

- ConcurrencyMode.Reentrant
  - WCF stellt automatische Synchronisierung der ankommenden Aufrufe sicher
  - Nur ein *„WCF-WorkerThread“* darf sich in der Service Instance befinden
  - Sollte die Service Instanz einen anderen Service rufen, kann der nächste Aufrufer die Instanz betreten



WCF – Nebenläufigkeit in Services (IV)

- Wird über das ConcurrencyMode Behavior gesteuert

```
1:  [ServiceBehavior(InstanceContextMode =
                     InstanceContextMode.PerSession,
                     ConcurrencyMode=
                         ConcurrencyMode.Multiple)]
2:  public class DemoServiceImpl :
                                 ISimpleDemoService,
                                 IDisposable { ..}
```

- Service Typen und Concurrency
  - `PerCall` Services laufen immer im `ConcurrencyMode.Single`, auch wenn ein abweichender Modus eingestellt wurde
  - `Single` Services und `PerSession` Services können den Concurrency Modus frei wählen.

Agenda

- Anforderungen
- Eigenschaften
- Architektur Überblick
  - Architekturüberblick
  - WCF ABC
  - Clientseitiges Programmiermodell
  - Serverseitiges Programmiermodell
    - Hosting
    - Service bereitstellen
    - Service Typen
    - Nebenläufigkeit
    - Throttling
- Fazit (und .Net 3.5 und VS 2008 …)
- Literaturhinweis

WCF – ‚Throttling' in Services (I)

- Dient zum Einstellen von
  - Maximale Anzahl der nebenläufigen Sessions pro Service (default = 10)
    - Hat keine Auswirkungen bei Session-losen Verbindungen (wie BasicHttp)
  - Maximale Anzahl der nebenläufigen Aufrufen an alle Service Instanzen eines Typs (default 16)
  - Maximale Anzahl an Service Instanzen (Context Objekten) (default unbegrenzt)
    - Hängt stark von InstanceContextMode ab

WCF – ‚Throttling' in Services (II)

- Kann per Konfiguration oder im Programmcode eingestellt werden
- Throttling Per Konfiguration

```
 1: <configuration>
 2:    <system.serviceModel>
 3:       <services>
 4:          <service name="WCFDemo.DemoServiceImpl"
              behaviourConfiguration = "Throttle" >
 5:          </service>
 6:       </services>
 7:       <behaviours>
 8:          <behaviour name = "Throttle">
 9:             <serviceThrottling
10:                maxConcurrentCalls = "100"
11:                maxConcurrentSessions = "1000"
12:                maxConcurrentInstances = "100" />
```

Agenda

- Anforderungen
- Eigenschaften
- Architektur Überblick
  - Architekturüberblick
  - WCF ABC
  - Clientseitiges Programmiermodell
  - Serverseitiges Programmiermodell
- Fazit (und .Net 3.5 und Visual Studio 2008 Neuerungen)
- Literaturhinweis

WCF – Fazit

- Einstieg in WCF ist einfach und schnell
- WCF bietet eine Vielzahl von Möglichkeiten (siehe Bindings, unterstützte (Netzwerk-) Protokolle)
- WCF biete eine Menge an wichtigen Diensten wie Security, Transaktion Handhabung, die sich nahtlos in das Grundframework einbetten
- Definiert kein so restriktives Programmiermodell, wie z.B. EJB, dies hat Vorteile (mehr Freiheit) als auch Nachteile (Threading Probleme)

WCF – Fazit (II)

- 'Bedachtloses' Konfigurieren/Programmieren von Diensten kann Seiteneffekte haben, z.B. `PerSession` Services werden zu `PerCall` Services

- Exception Handling ist meines Erachtens gewöhnungsbedürftig und zum Teil kompliziert (z.B. wann kann ich Proxy noch verwenden)

→ Fazit: WCF ist wie jedes mir bekannte Kommunikationsframework nur mit Erfahrung 'beherrschbar‘ und von der Komplexität mit CORBA vergleichbar

WCF – Neuerungen in .Net 3.5

- Unterstützung von RSS und ATOM Syndication
- Neues Binding für Representational State Transfer (REST)
- WCF und Workflow Foundation (WF) wachsen zusammen, WF Dienste als WCF Services anbieten
- Implementierung der aktuellen OASIS WS* Spezifikationen
  - WS Atomic Transaction 1.1, WS-ReliableMessaging 1.1, WS-SecureConversation und Web Services Coordination (WS-Coordination) 1.1.
- Web 2.0 und AJAX fähige WCF Services

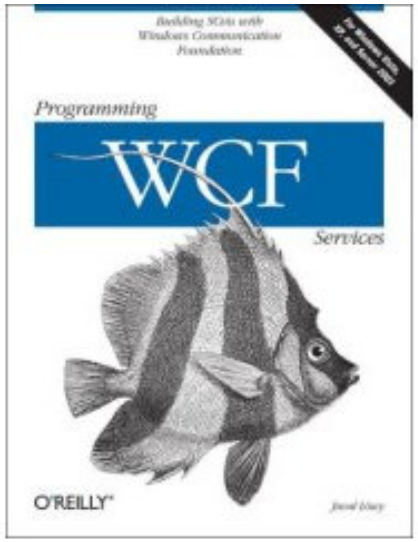
WCF –Neuerungen in Visual Studio 2008

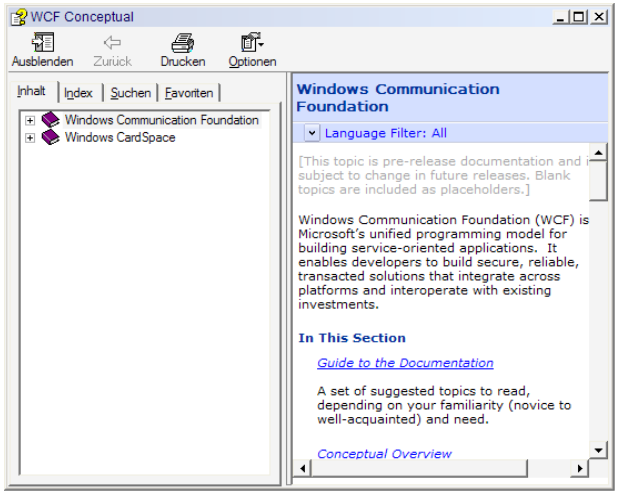
- Auswahl des Zielframeworks (2.0,3.0 und 3.5) möglich
- Service-Referenzen:
  - Settings (z.B. asynchrone Stub-Methoden automatisch generieren)
  - Handhabung von Collections/Arrays
  - Aktualisieren von Proxyverweisen
- WCF bereitgestellter Host, in den WCF Assemblies „deployt“ werden können
  - Aufruf: WcfSvcHost.exe /service:MyService.dll /config:App.config
  - Erspart die Einwicklung eines Hosts zur EntwicklungszeitTestClient
- WCF Testclient
  - Aufruf: WcfTestClient.exe http://locahost:9000

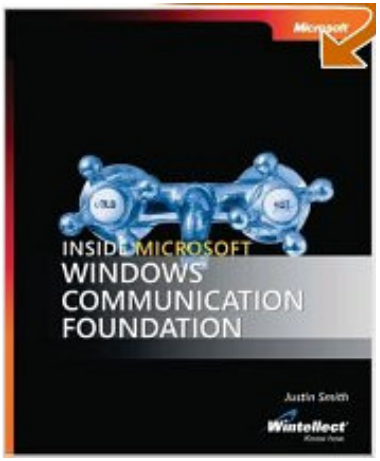
Agenda

- Anforderungen
- Eigenschaften
- Architektur Überblick
  - Architekturüberblick
  - WCF ABC
  - Clientseitiges Programmiermodell
  - Serverseitiges Programmiermodell
- Fazit (und .Net 3.5 und Visual Studio 2008 Neuerungen)
- Literaturhinweis

WCF – Literatur







- Web Ressourcen
  - http://wcf.netfx3.com/
  - http://www.thatindigogirl.com/

15.–18. 09. 2008
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Vielen Dank!

Thomas Haug
MATHEMA Software GmbH