Windows Communication Foundation (WCF) 4.5

Verteilte Anwendungen entwickeln

Vielen Dank

Ihr Trainer: Konstantin KLETZANDER

- Trainer @ PPEDV
- Social Media Kontakt:
  - XING
  - LinkedIn

- Fragen & Interessantes: http://blog.ppedv.de/

Feedback

- Ich brauche Sie, um dieses Training zu verbessern!
  - Was ist nicht klar?
  - Was fehlt Ihnen?
  - Welchen Schwierigkeiten begegnen Sie?
  - Bewerten Sie das Training auf: http://feedback.ppedv.de/

Agenda

- Einführung Serviceorientierte Architektur
- Überblick der Windows Communication Foundation (WCF)
- Erstellung eines SOAP Service mit WCF
- Hosting von WCF-Services
- Bekannte Fehler erkennen, Test und Troubleshooting von WCF-Diensten
- Erstellung eines WCF-Clients (SOAP)
- Erstellung eines Service und eines Clients mit REST
- Sicherheit

Wie stellt man sich technischen Problemen

- F&A
- Methode Google und StackOverflow

Einführung Serviceorientierte Architektur

Was ist ein Service?

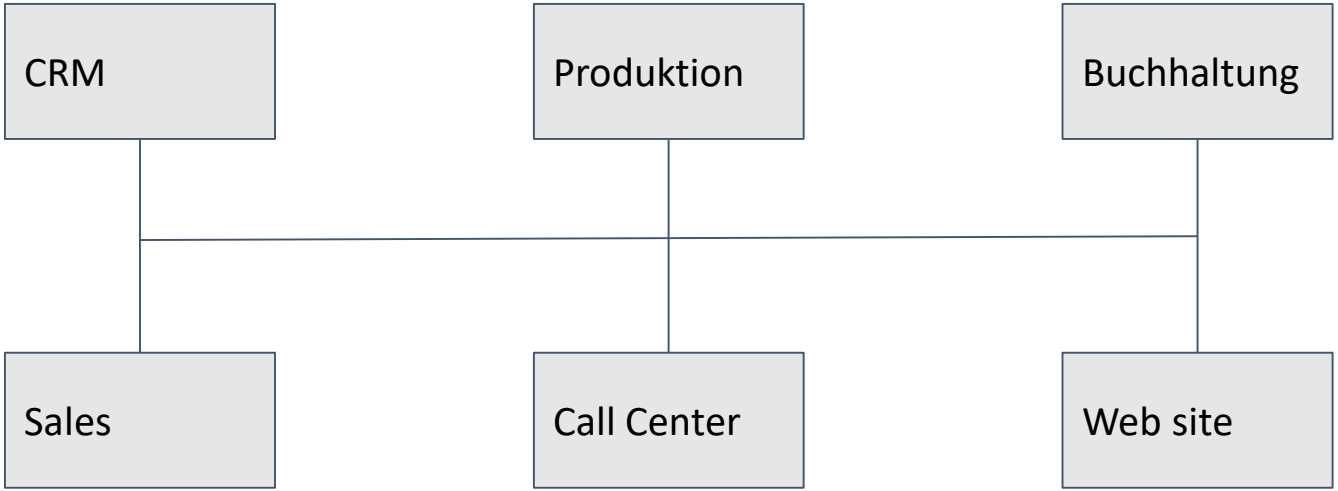
- Eine Funktionalität, die über ein strukturiertes *Messaging Schema* bereitgestellt werden kann
- Die Struktur des Messaging Schemas kann nahezu *alles* sein: SOAP, XML, JavaScript Object Notation, etc.
- Der Transport kann über verschiedene OSI-Modell Schichten erfolgen und die unterschiedlichen Protokolle verwenden: HTTP, TCP/IP, UDP, SMTP
- ...oder sogar Brieftauben ;)

Was ist ein Client?

- Konsumiert die vom Service zur Verfügung gestellte Funktionalität
- Console Application, Windows Forms, WPF, Xamarin, ASP.NET, ...

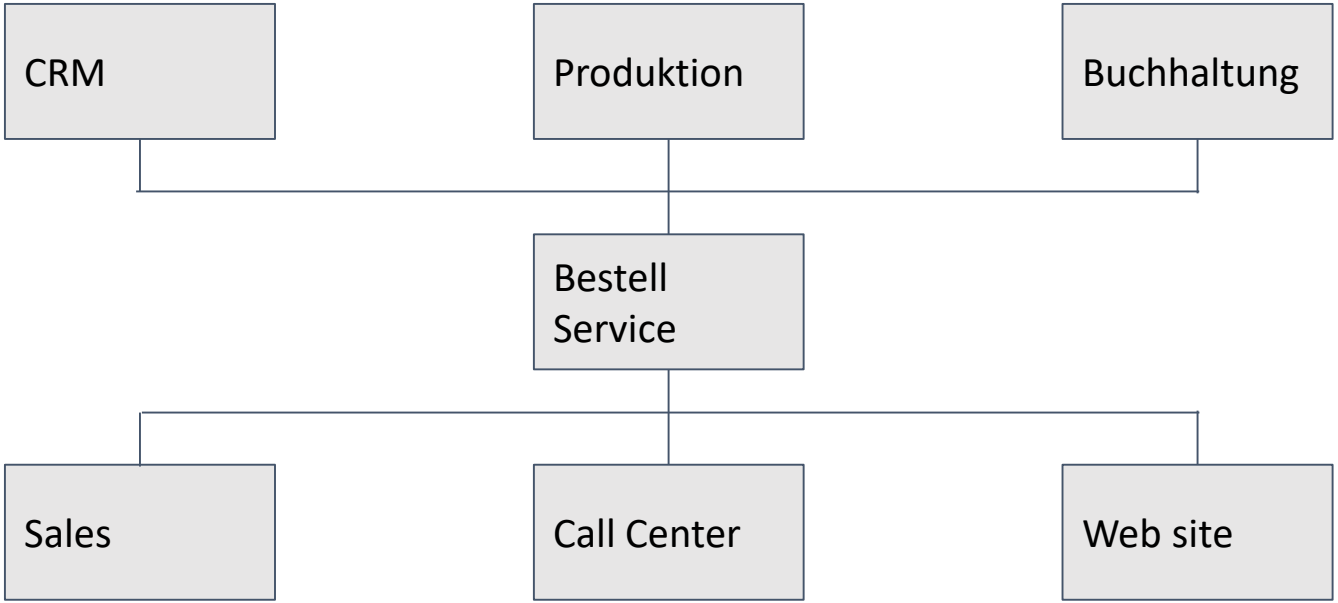
Warum einen Service verwenden?

- Business Beispiel



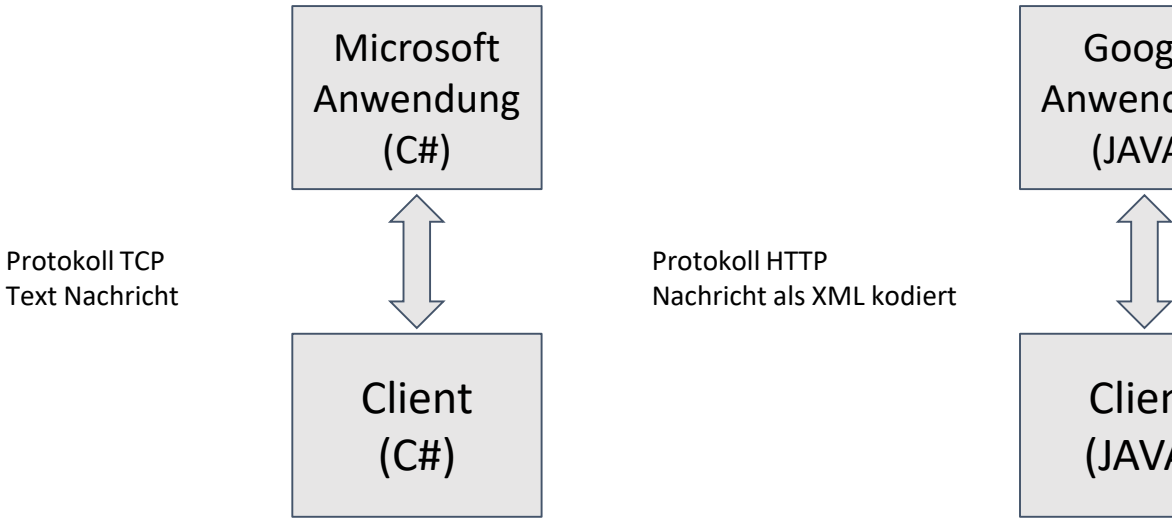
Warum einen Service verwenden?

- Business Beispiel



Warum verteilte Anwendungen?

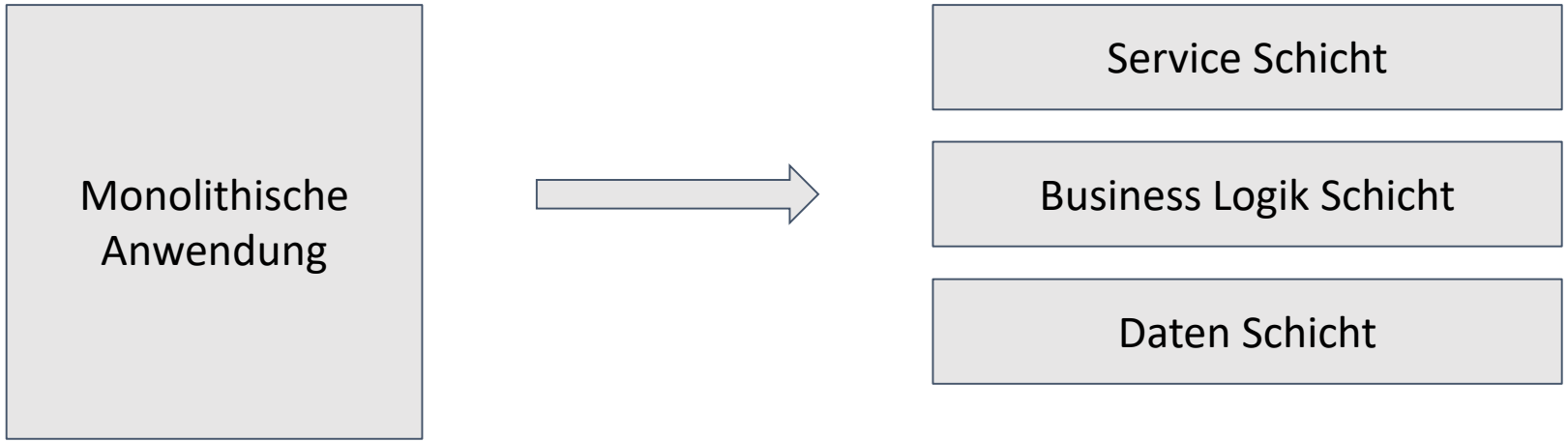
- Problem: Kein einheitlicher Standard, Komplizierte Client Erstellung



"Monolithische Architektur"

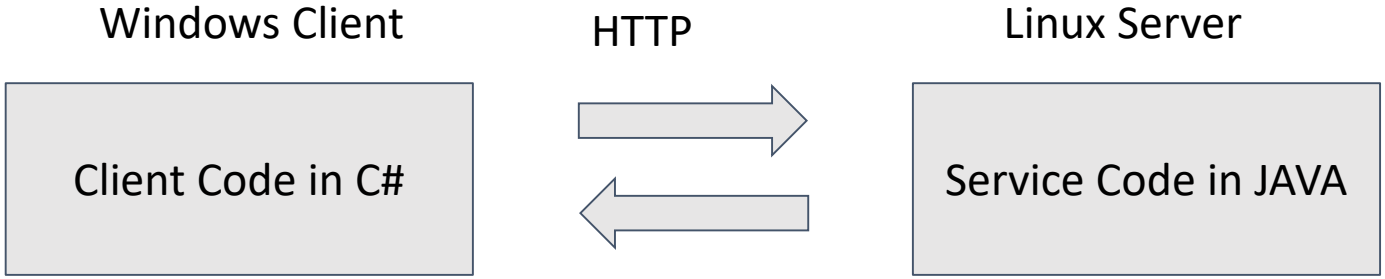
Architektur in Schichten

“Monolithische Architektur” => Mehrschichtige Architektur



Warum *Service-Orientierte-Architektur*?

- Technologie-Unabhängigkeit als Notwendigkeit
- Standardisierte Protokolle und Nachrichten Kodierung in der Kommunikation



Fundamentale Konzepte

- **Message:** Nachricht, die übermittelt wird
- **Endpoint:** Bestimmt wohin und woher einer Nachricht
- **Hosting:** Wo und wie der Service zur Verfügung gestellt wird
- **Metadata:** Erleichtert die Kommunikation zwischen Endpunkten
- **Client:** Anwendung zur Interaktion mit der Service-Funktionalität
- **Channel:** Medium das zur Kommunikation zwischen Client und Service dient

Exkurs: WS-*

- WS-* = Abkürzung für eine Vielzahl an *Webservice Spezifikationen*, wie:
  - WS-Security
  - WS-SecureCoversation
  - WS-Federation
  - …

- https://en.wikipedia.org/wiki/List_of_web_service_specifications

Exkurs: URI, URL, URN

- **URI**: Uniform Resource Indicator
  Zeichenkette zur Identifikation einer Ressource

- **URL**: Uniform Resource Locator
  Teilmenge der URI, gibt an wo die Ressource zu finden ist. Muss nicht HTTP sein, kann auch Z.B. FTP sein
  http://www.ppedv.de/programm.txt

- **URN**: Uniform Resource Name
  Kann auch Teilmenge einer URI sein, muss die Ressource genau spezifizieren
  urn:beispiele:programmservice

Was ist REST?

- REST = Representational State Transfer
- REST ist ein ressourcenorientierter Architekturstil
- Verwendung universeller Interfaces des HTTP (GET, POST, PUT, DELETE)
- Lokalisierung von Ressourcen mittels URI
- Plattformunabhängig

- Beispiel response: `{"id":1,"content":"Hello, World!"}`

Was ist SOAP?

- SOAP ist ein Kommunikationsprotokoll im Internet
- Protokoll zur Formatierung von Informationen mittels XML
- Kann Nachrichten austauschen oder Remote-Procedures aufrufen
- TCP oder HTTP
- Struktur definiert den Transport der Nachricht
- Plattformunabhängig

SOAP Nachricht

```
<?xml version = "1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV = "http://www.w3.org/2001/12/soap-envelope"
   SOAP-ENV:encodingStyle = "http://www.w3.org/2001/12/soap-encoding">

   <SOAP-ENV:Header>
      ...
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      ...
      <SOAP-ENV:Fault>
         ...
      </SOAP-ENV:Fault>
      ...
   </SOAP-ENV:Body>
</SOAP_ENV:Envelope>
```

- SOAP Envelope einer Nachricht = Name + Namespace + Header + Body (+ Fault)

SOAP Nachricht - Syntax Regeln

- Muss in XML kodiert sein
- Muss den SOAP Envelope Namensraum verwenden
- Muss den SOAP Encoding Namensraum verwenden
- Darf keine DTD Referenz enthalten
- Darf keine XML Processing Anweisungen enthalten

https://www.w3schools.com/xml/xml_soap.asp

Message Exchange Patterns

- Beschreibung des Nachrichtenflusses
  - Request-Reply, Point-to-Point, Publish-Subscribe, …

Weitere Informationen:
https://msdn.microsoft.com/en-us/library/aa480027.aspx

Serviceorientierte Architektur

SOA = Service Oriented Architecture = Service-Orientierte Architektur

1. Unabhängige, wiederverwendbare Komponenten (= Services), durch Message und *Contract* (= Vertrag) lose verbunden
2. Der Client muss nur den Contract kennen, um mit dem Service arbeiten zu können
3. Der Service teilt das Schema und den Contract, jedoch nicht die Implementierung
4. Der Service ist konfigurierbar
5. Die Konsumierung des Services ist unabhängig von der verwendeten Technologie des Clients

Zusammenfassung

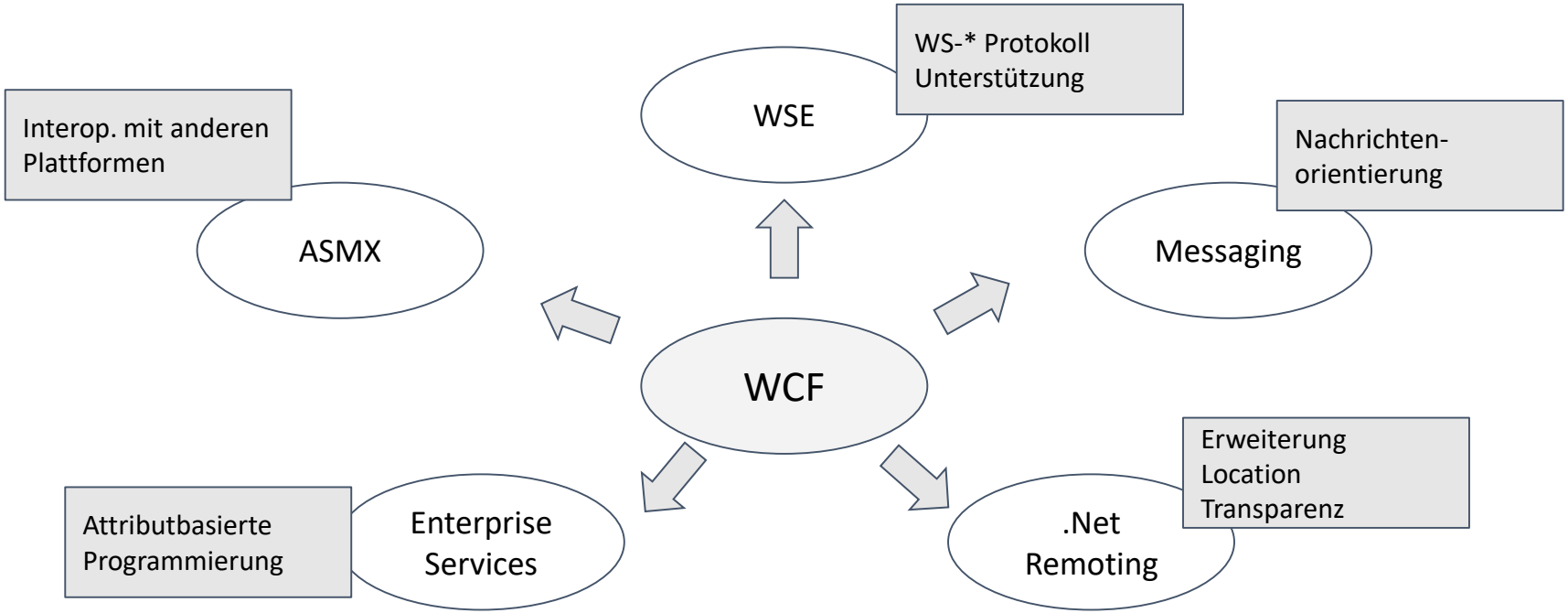
- 4 Argumente für SOA:
  - *Skalierbarkeit, Wartbarkeit, Interoperabilität, Flexibilität*

- Wann sollte SOA in Erwägung gezogen werden?
  - Der Service soll leicht von unterschiedlichen Clients konsumiert werden können
  - Kürzere Entwicklungszeit bei der Service Erstellung
  - Umfassende Konfigurationsmöglichkeiten für den Service - vor allem das Protokoll!
  - Möglichkeit zur Wartung des Services

- SOAP Nachrichten Testen mit SOAP UI: https://www.soapui.org

Überblick der Windows Communication Foundation (WCF)

Hintergründe



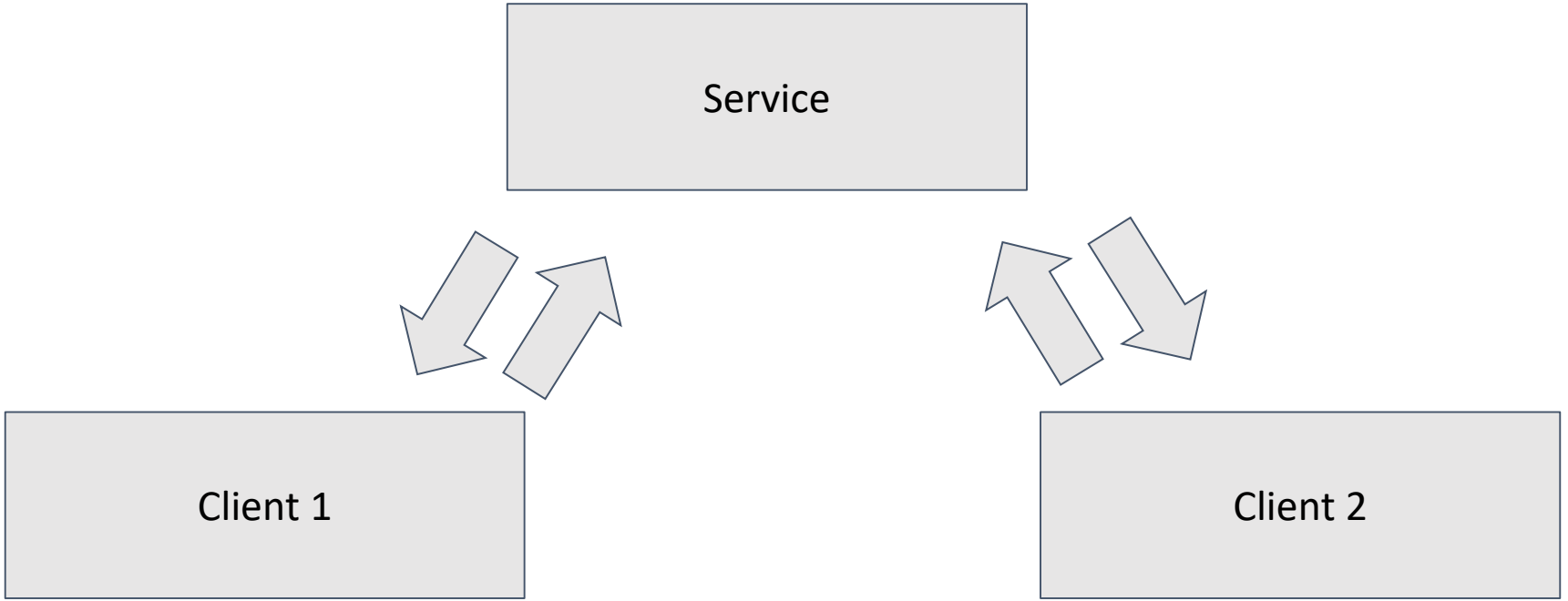
Was ist WCF?

- WCF = Windows Communication Foundation
- Framework zur Erstellung von verteilten Anwendungen
- Technologische Erstellungshilfe auf verschiedenen Endpunkten
- SOAP Messages
- Bestehend aus 3 Komponenten
  - Service
  - Client
  - Host

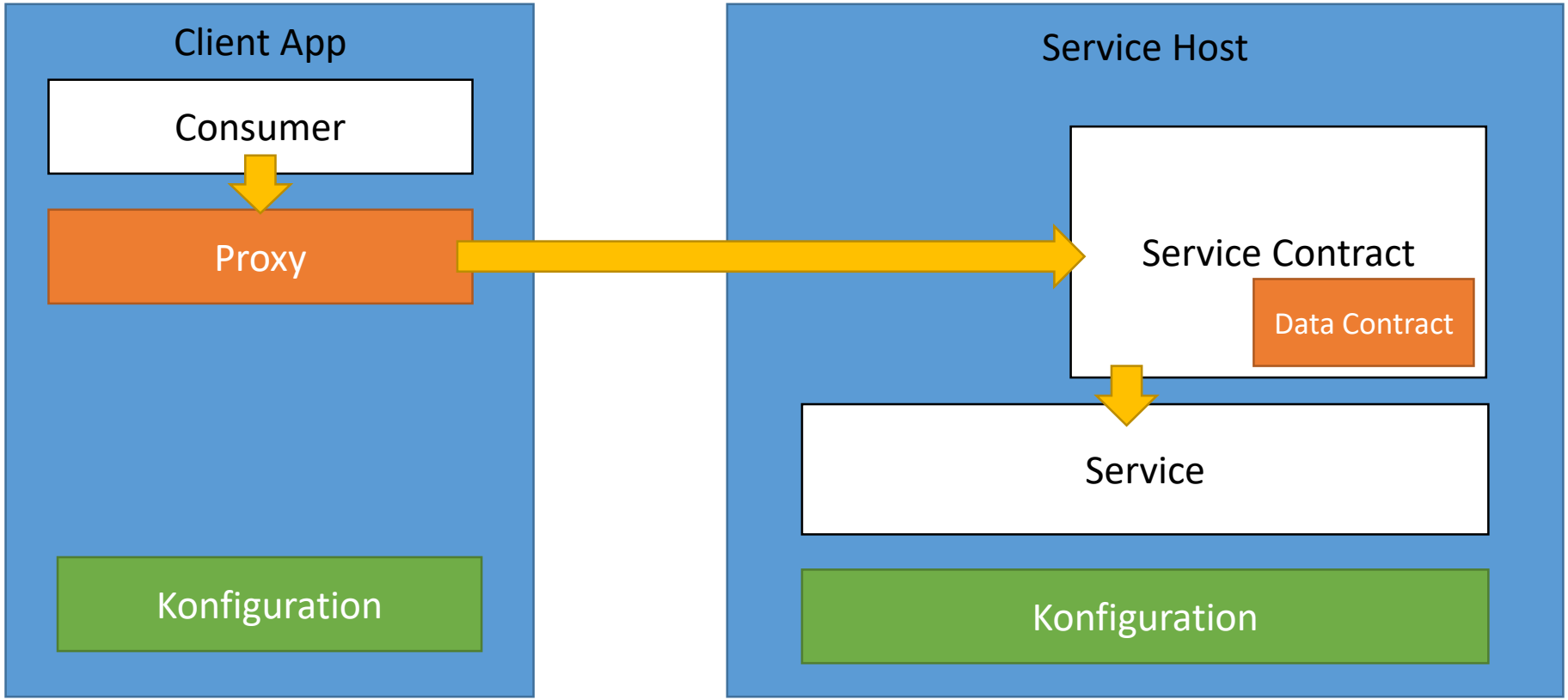
5 Eigenschaften von WCF

1. Die verschiedenen Aspekte der verteilten Kommunikation (Schnittstelle, Implementierung, Kommunikationsprotokoll, Infrastrukturdienste, Hosting) sind logisch trennbar
2. WCF ermöglicht die Erstellung eines Kommunikationskanals aus verschiedenen Formaten und Protokollen
3. Unterstützung für proprietäre (.NET) Protokolle und plattformübergreifende Protokolle und Formate (W3C/WS-*)
4. Änderung der Kommunikationsform des Dienstes ohne (Neu-)Kompilierung möglich
5. Ergänzung um weitere Kommunikationsformen ohne (Neu-)Kompilierung

Client und Server



WCF Bausteine



WCF Konfiguration

- Endpunkte
  - Adresse
  - Binding
  - Contract
- Bindings
- Behaviors

Ziel: Interoperabilität

- Services können auch mit nicht .NET Anwendungen kommunizieren
- Standard Web Service SOAP und REST
- SOAP ist ein Protokoll zur Formatierung von Informationen mittels XML
  - TCP oder HTTP
  - Struktur definiert den Transport der Nachricht
- WCF erledigt die “*Klempnerarbeit*”, um die Nachrichten zu übermitteln
  - Einfache Erstellung von Services und Clients
  - Fokussierung auf den *Business Aspekt* bei der Service Erstellung

Flexibel und Sicher

- Sehr flexible Konfiguration
- Relativ einfach, sichere Services erstellbar
- Verwendung von mehreren Clients vorgesehen
  - Hohe Verfügbarkeit
  - Zuverlässig
- Einfache Konfiguration (z.B. Tausch des Protokolls)

WCF - Architektur

Anwendung	
Contracts	*Data Contract, Message Contract, Service Contract, Policy & Binding*
Service Runtime	*Error Behavior, Metadata Behavior, Message Behavior, ...*
Messaging	*WS Security, HTTP Channel, TCP Channel, Named Pipes, MSMQ Channel, ...*
Aktivierung und Hosting	*Windows Activation Service, .EXE, Windows Services, COM+*

Zusammenfassung WCF

- WCF ist eine Technologie zur Entwicklung von SOA Anwendungen mit C#.
- WCF ermöglicht die schnelle Erstellung von Services und Clients, um sich auf die Business Logik zu konzentrieren zu können
- WCF ist sehr flexibel und konfigurierbar (Protokolle, Sicherheit, Timeout, Transaktionen, ...)

Demo: WCF Service Hello World

- Self-Hosting

Erstellung eines SOAP Service mit WCF

Übersicht

- ABC eines WCF Services
- Erstellung des Service Contracts und der Implementierung
- Konfiguration der Bindings
- Konfiguration der Behaviors
  - Fokus auf InstanceContextMode und ConcurrencyMode
  - Fokus auf IsOneWay
  - Fokus auf TransactionScopeRequired
- Behandeln von Exceptions am Service

ABC eines WCF Services

- Am Endpunkt:
  - Adresse
  - Binding
  - Contract

WCF - Architektur

Anwendung	
Contracts	*Data Contract, Message Contract, Service Contract, Policy & Binding*
Service Runtime	*Error Behavior, Metadata Behavior, Message Behavior, ...*
Messaging	*WS Security, HTTP Channel, TCP Channel, Named Pipes, MSMQ Channel, ...*
Aktivierung und Hosting	*Windows Activation Service, .EXE, Windows Services, COM+*

Adresse

Was genau bedeutet Adresse?

- Die Adresse beschreibt, *wo(-her)* der Service eingehende Nachrichten erhält
- Z.B. URL mit Protokollangabe + Pfad + Port
- Client kann diese URL verwenden, um sich mit dem Service zu verbinden
- Beispiel: http://localhost:8001/MeinService/HelloWorld.svc

Bindings

Was ist das Binding?

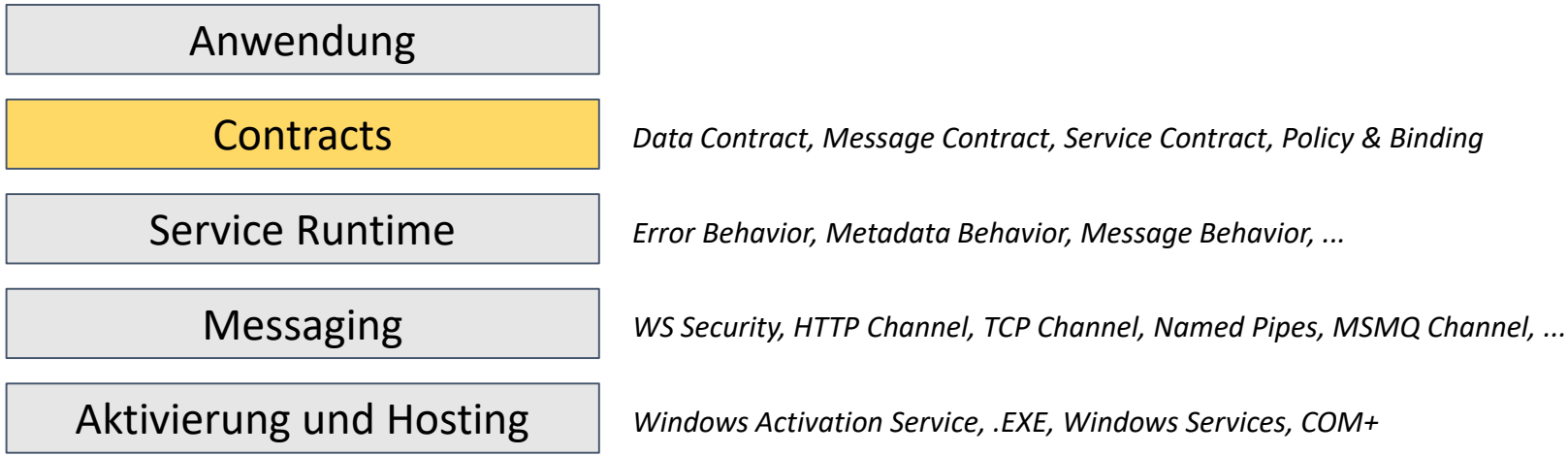
- Das Binding beschreibt, *wie* der Service eingehende Nachrichten erhält
- Z.B. die bei der Kommunikation verwendeten Protokolle und Format
- Das Binding definiert den Kommunikationskanal bzw. das Transportschema
  - HTTP/HTTPS: http://localhost:8001/MeinService
  - TCP: net.tcp://localhost:8002/MeinService
  - IPC: net.pipe://localhost/MeinService
  - Peer network
  - MSMQ: net.msmq/localhost/private/MeinService
  - ServiceBus: sb://MeinNamensraum.servicebus.windows.net/

Welche Protokolle gibt es?

- **basicHttpBinding** => einfachste Variante, erweiterbar, abgesicherter Nachrichtentransport möglich, Keine Transaktionen, Kompatibel SOAP 1.1
- **wsHttpBinding** => abgesicherter Transport und Messages möglich (verschlüsselte Messages), Transaktionen möglich
- **webHttpBinding** => REST
- **netTcpBinding** => TCP, ausschließlich .NET, Transaktionen und Sicherheit auf WCF optimiert, bessere Performance als HTTP
- **netNamedPipeBinding** => Client Server auf der gleichen Maschine, ausschließlich .NET, sehr schnell
- **netMsmqBinding** => Asynchrone Kommunikation mit MSMQ

Contract

WCF - Architektur



Was ist ein Contract?

- Der *Contract* (=Vertrag) beschreibt, *welche* Arten von Nachrichten der Service erhalten kann. *Was* kann der Service erhalten?
- Vereinbarung zwischen Client und Service
- Der Vertrag definiert die eigentlichen Operationen, die der Dienst anbietet
- Darin spezifiziert:
  - ServiceContract,
  - OperationContract,
  - DataContract,
  - MessageContract,
  - FaultContract

Behaviors

WCF - Architektur

Anwendung	
Contracts	*Data Contract, Message Contract, Service Contract, Policy & Binding*
Service Runtime	*Error Behavior, Metadata Behavior, Message Behavior, ...*
Messaging	*WS Security, HTTP Channel, TCP Channel, Named Pipes, MSMQ Channel, ...*
Aktivierung und Hosting	*Windows Activation Service, .EXE, Windows Services, COM+*

Was ist ein Behaviour?

- Behavior = Verhalten
- Elemente zur Konfiguration des Verhaltens von Service, Contract und Endpoint
- Beispiel: Session Management, Concurrence, Throttling, Transactions ...

Worauf lassen sich Behaviors anwenden?

- Behaviors werden in der Konfiguration von Client oder Service angewendet
- Behaviors sind kein Bestandteil der WSDL
- *Service* => ServiceBehavior
- *Endpoint* => EndpointBehavior
- *Operation* => OperationBehavior
- *Contract* => ContractBehavior

Hosting

WCF - Architektur

Anwendung	
Contracts	*Data Contract, Message Contract, Service Contract, Policy & Binding*
Service Runtime	*Error Behavior, Metadata Behavior, Message Behavior, ...*
Messaging	*WS Security, HTTP Channel, TCP Channel, Named Pipes, MSMQ Channel, ...*
Aktivierung und Hosting	*Windows Activation Service, .EXE, Windows Services, COM+*

Hosting Übersicht

- Services müssen in einem gehosteten Prozess laufen
- Der Host hört sich nach eingehenden Nachrichten um und verteilt die „Calls“ an die zuständigen Service Methoden
- Die Konfiguration bestimmt, wie der Host den Service verwenden darf

Wie kann ein WCF Service gehostet werden?

- WCF Service Library
  - Schnelle Tests und Debugging
- Self-Hosting
  - Jeder .NET Prozess
  - Windows Service
- Internet Information Service (IIS)
  - Host in Web Site auf IIS Server
  - Definiert als ASP.NET Web Application Projekt
  - Kann nur HTTP Traffic bearbeiten
  - Vorteil: IIS Skalierbarkeit, Robust und Konfigurierbar
  - Kann auch weitere Protokolle (TCP, MSMQ, …) durch “WAS” bearbeiten
    - Windows Process Activation Service

Zusammenfassung WCF Architektur

- WCF ermöglicht eine relativ einfache Client und Service Kommunikation
- Die Servicegrenzen sind im Service Contract, Data Contract und der Konfiguration definiert
- Endpunkte bestehen aus Adresse, Binding und Contract
- Behaviors legen fest, wie sich der Service verhält, wenn eine Nachricht ankommt
- Bindings spezifizieren die Kommunikationspipeline zwischen Service und Client

Demo: WCF Service Hello World

- Hosting IIS

- Nützliche Links:
  - Installation IIS auf Windows 10
  - Aktivierung NET.TCP auf IIS7

Implementierung eines Services mit WCF

- WCF Service Library Projekt erstellen
- Service Contract definieren
- Data Contracts als Anwendungsparameter definieren
- Service Klassen implementieren

Übung: WCF Service Taschenrechner

- Self-Hosting
- 4 Grundrechnungsarten als ServiceContract
- Client als eigenständige Consolenanwendug

- Proxy Erstellung mit svcutil.exe über die CMD
  - `C:\temp> svcutil /target:code net.tcp://localhost:XXXX/IRechnerService/Mex`

Übung: WCF Service Taschenrechner

- WCF Service Implementierung der 4 Grundrechnungsarten
- Self-Hosting
- WPF Client
- Bonus:
  - Hosting auf IIS

(Bekannte) WCF Errors

Good to know…

- Service Contracts sind .NET Interface Attribute [ServiceContract] und [OperationContract]
- Parameter und Return Types von Serviceoperationen sind Data Contracts
- Data Contracts können explizit, implizit oder einfach (=primitive) sein
- Service Implementierungen können [ServiceBehavior] Attribut verwenden
- Service Instanz / Lebenszyklus
  - PerCall bevorzugen wenn Transaktionen verwendet werden

Timeouts

- System.TimeoutException: The request channel timed out while waiting for a reply after 00:00:59.9320000. Increase the timeout value passed to the call to Request or increase the SendTimeout value on the Binding. The time allotted to this operation may have been a portion of a longer timeout' has exceeded the allotted timeout of 00:01.00. The time allotted to this operation may have been a portion of a longer timeout

Timeouts in WCF

Name	Konfiguration	Beschreibung	Default Value
OpenTimeout	Binding, client + service	Geöffneter Kommunikationskanal	1min
CloseTimeout	Binding, client + service	Geschlossener Kommunikationskanal	1min
SendTimeout	Binding, client + service	Zeit Maximum bei Anfrage vom Client an den Service	1min
OperationTimeout	Binding, client + service	Zeit Maximum bei Anfrage vom Client an den Service	(SendTimeout)
ReceiveTimeout	Binding, Service	Die Zeit bevor eine Session ohne neue Message geschlossen wird	10min
InactivityTimeout	ReliableSession client + service	Zeit zwischen 2 KeepAlive messages bevor die Verbindung geschlossen wird	10min
ChannelInitializationTimeout	ConnectionOrientedTransportBinding client + service	Wie lange ein Kanal im initialen Zustand sein kann	30 sec
RequestInitialiyationTimeout	HttpTransportBinding	Defaultmäßig deaktiviert	0

Beispiel: Konfiguration Timout Binding

```
<configuration>
  <system.serviceModel>
    <bindings>
      <wsHttpBinding>
        <binding openTimeout="00:10:00"
                 closeTimeout="00:10:00"
                 sendTimeout="00:10:00"
                 receiveTimeout="00:10:00">
        </binding>
      </wsHttpBinding>
    </bindings>
  </system.serviceModel>
</configuration>
```

Message size

- The maximum message size quota for incoming messages (65536) has been exceeded. To increase the quota, use the MaxReceivedMessageSize property on the appropriate binding element.

- An error occurred while receiving the HTTP response to http://localhost:9003/MyService.svc. This could be due to the service endpoint binding not using the HTTP protocol. This could also be due to an HTTP request context being aborted by the server (possibly due to the service shutting down). See server logs for more details.

- Maximum number of items that can be serialized or deserialized in an object graph or increase the MaxItemsInObjectGraph quota.

Eigenschaften mit Einfluss auf die maximalen Message Größen

- **MaxReceivedMessageSize**: Repräsentiert die maximale Größe der vom WCF empfangenen Nachricht. Default Wert 65536 Bytes.
- **MaxRequestLength**: Maximale Request Größe. Default Größe: 4069 KB (4 MB)
- **MaxBufferSize**: Maximale Größe des Buffers. Identisch buffered messages = **MaxReceivedMessageSize**. Default Value 65536 Bytes.
- **MaxBufferPoolSize**: Maximale Größe des für den BufferManager notwendigen allokierten Speichers zur Verwaltung der Endpunkte.

Beispiel: Konfiguration Message Size Binding

```
<bindings>
    <basicHttpBinding>
       <binding name="Binding1"
                hostNameComparisonMode="StrongWildcard"
                receiveTimeout="00:10:00"
                sendTimeout="00:10:00"
                openTimeout="00:10:00"
                closeTimeout="00:10:00"
                maxReceivedMessageSize="65536"
                maxBufferSize="65536"
                maxBufferPoolSize="524288"
                transferMode="Buffered"
                messageEncoding="Text"
                textEncoding="utf-8"
                bypassProxyOnLocal="false"
                useDefaultWebProxy="true" >
              <security mode="None" />
        </binding>
    </basicHttpBinding>
 </bindings>
```

- Alternative Idee, um diese Probleme zu lösen: Verwenden von TransferMode Stream anstelle von default Buffer (Daten als Block oder ASAP senden)

```
<bindings>
    <netTcpBinding>
        <binding name="MyService.netTcpBinding"
                 transferMode="Buffered" closeTimeout="0:01:00" openTimeout="0:01:00">
        </binding>
    </netTcpBinding>
</bindings>
```

Readerquotas

- Definiert die Einschränkungen der Komplexität von SOAP Messages an den Endpunkten.

Attribut	Beschreibung
maxArrayLength	A positive integer that specifies the maximum allowed array length of data being received by Windows Communication Foundation (WCF) from a client. The default is 16384.
maxBytesPerRead	A positive integer that specifies the maximum allowed bytes returned per read. The default is 4069.
maxDepth	A positive integer that specifies the maximum nested node depth per read. The default is 32.
maxNameTableCharCount	A positive integer that specifies the maximum characters allowed in a table name. The default is 16384.
maxStringContentLength	A positive integer that specifies the maximum characters allowed in XML element content. The default is 8192.

KnownType

- There was an error while trying to serialize parameter http://tempurl.com/:query. The InnerException message was 'Type Test.Dog' with data contract name 'Dog:http://schemas.datacontract.org/2004/07/**Test.Dog' is not expected. Add any types not known statically to the list of known types – for example, by using the KnownTypeAttribute attribute** or by adding them to the list of known types passed to DataContractSerializer.'. Please see InnerException for more details.

KnownType: Lösungsvorschlag ServiceContract

```
[ServiceContract]
public interface IMyService
{
    [OperationContract]
    Animal GetAnimal(string name);
}

[DataContract]
public abstract class Animal { }

[DataContract]
public class Dog : Animal { }

[DataContract]
public class Cat : Animal { }
```

```
[ServiceContract]
public interface IMyService
{
    [OperationContract]
    [ServiceKnownType(typeof(Dog))]
    [ServiceKnownType(typeof(Cat))]
    Animal GetAnimal(string name);

    [DataContract]
    [KnownType(typeof(Dog))]
    [KnownType(typeof(Cat))]
    public abstract class Animal { }
}
```

KnownType: Lösungsvorschlag Konfiguration

```
<configuration>
  <system.runtime.serialization>
    <dataContractSerializer>
      <declaredTypes>
        <add type="Test.Animal, MyAssembly">
          <knownType type="Test.Dog, MyAssembly"/>
        </add>
        <add type="Test.Animal, MyAssembly">
          <knownType type="Test.Dog, MyAssembly"/>
        </add>
      </declaredTypes>
    </dataContractSerializer>
  </system.runtime.serialization>
</configuration>
```

Tipps & Tricks Service Erstellung

- Service im Format Interface => Implementierung
- Contract im Interface angeben
- Service in Klassenbibliothek definieren und Klassenbibliothek in Host Projekt referenzieren
- Service Klasse nicht im Host Projekt verwenden
- Exceptions immer mit Try/Catch und FaultException<T> behandeln
- Logging und Include Exception sollte enabled sein, während das Projekt im Debug Mode kompiliert wird.
- Im Production Mode, sollte das Logging und die Exception Details disabled sein.

(Kritik) WCF im Vergleich...

Unterschied WCF und ASP.NET Web Services

- **Web Service:**
  - SOAP und XML
  - HTTP Protokoll
  - Nicht Open Source, Client muss XML verstehen
  - IIS Hosting
- **WCF:**
  - SOAP-Prinzip und XML
  - Weiterentwicklung der Web Services (ASMX)
  - verschiedene Protokolle wie TCP, HTTP, HTTPS, Named Pipes, MSMQ. WCF
  - sehr konfigurierbar, Client muss XML verstehen
  - IIS, als Windows Service oder in der Anwendung gehostet werden.

Vorteile und Nachteile von WCF

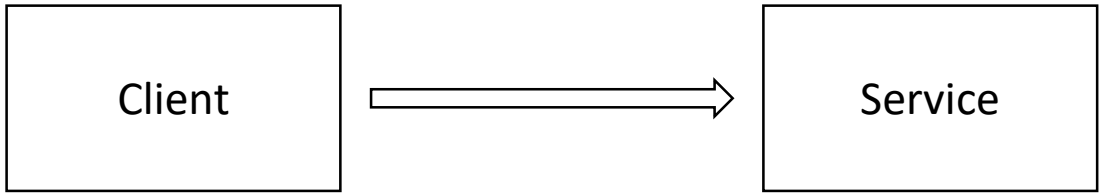
- **Vorteile**
  - WCF ist zuverlässiger und sicherer als die älteren ASMX Web Services
  - Nur geringe Code Anpassungen notwendig, um Änderungen vorzunehmen. Ein Anpassen des Bindings in der Konfiguration ist ausreichend.
  - WCF bietet Interoperabilität
  - Eingebauter Logging Mechanismus
  - Eingebauter JSON Support
  - Skalierbarkeit und definierte Security Einstellungen
  - Mehrere Transportmöglichkeiten (mehr als nur HTTP), Nachrichtenbasierte Security
- **Nachteile**
  - Keine Unterstützung des Binär Formates und nur begrenzte Unterstützung von Vorgänger Technologien wie .NET Remoting oder COM+
  - Austausch des verwendeten Protokolls nicht immer unproblematisch
  - WCF muss mehr konfiguriert werden als die WEB API

Unterschied WCF und ASP.NET Web API

- Die ASP.NET Web API ist eine Fortführung und Erweiterung der bestehenden WCF Web API, wobei sich verschiedene Konzepte geändert haben.
- Das WCF Framework wurde ursprünglich entwickelt, um SOAP basierte Services einfach erstellen zu können.
- Wenn einfachere REST Services erstellt werden sollen, ist die ASP.NET Web API eine gute Wahl!

Webservices mit WCF konsumieren

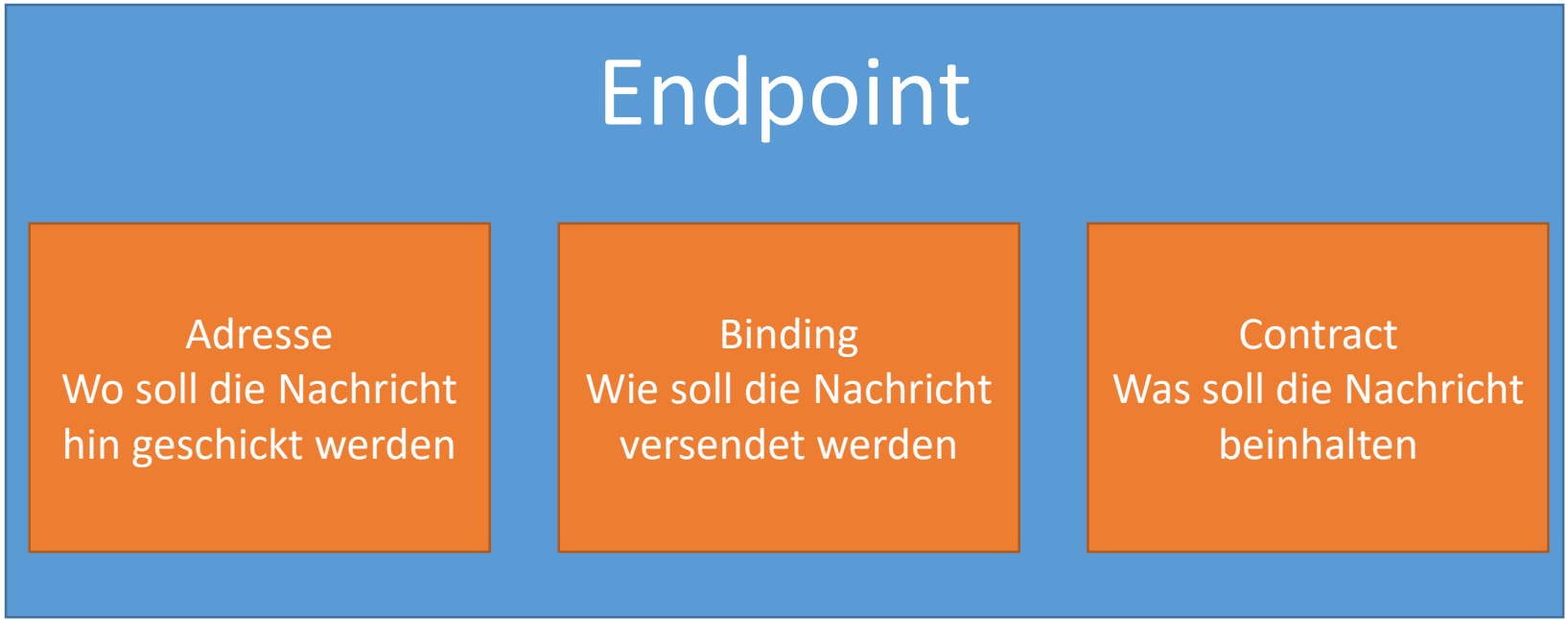
- Der Client benötigt die folgenden Infos vom Service
  - Wo sende ich die Nachricht hin (Adresse)
  - Wie sende ich die Nachricht, mit welchen Protokollen (Binding)
  - Was soll die Nachricht beinhalten (Contract)



- WCF antwortet auf diese Fragen mit Endpoints

Wiederholung: Endpoint

- Endpunkte definieren wie WCF die Kommunikationskanäle aufbaut



Endpunkte Konfiguration

- Mehrere verschiedene Endpunkte pro Service können definiert werden

REST over HTTP

SOAP + WS-* over MSMQ

SOAP + WS-* over TCP

```
<system.serviceModel>
  <services>
    <service name="InvoiceService">
      <endpoint
        address="http://server/invoiceservice"
        binding="webHttpBinding"
        contract="IInvoiceService"/>
      <endpoint
        address="net.msmq://server/invoicequeue"
        binding="netMsmqBinding"
        contract="IInvoiceService"/>
      <endpoint
        address="net.tcp://server:8081/invoiceservice"
        binding="netTcpBinding"
        contract="IInvoiceService"/>
    </service>
  </services>
</system.serviceModel>
```

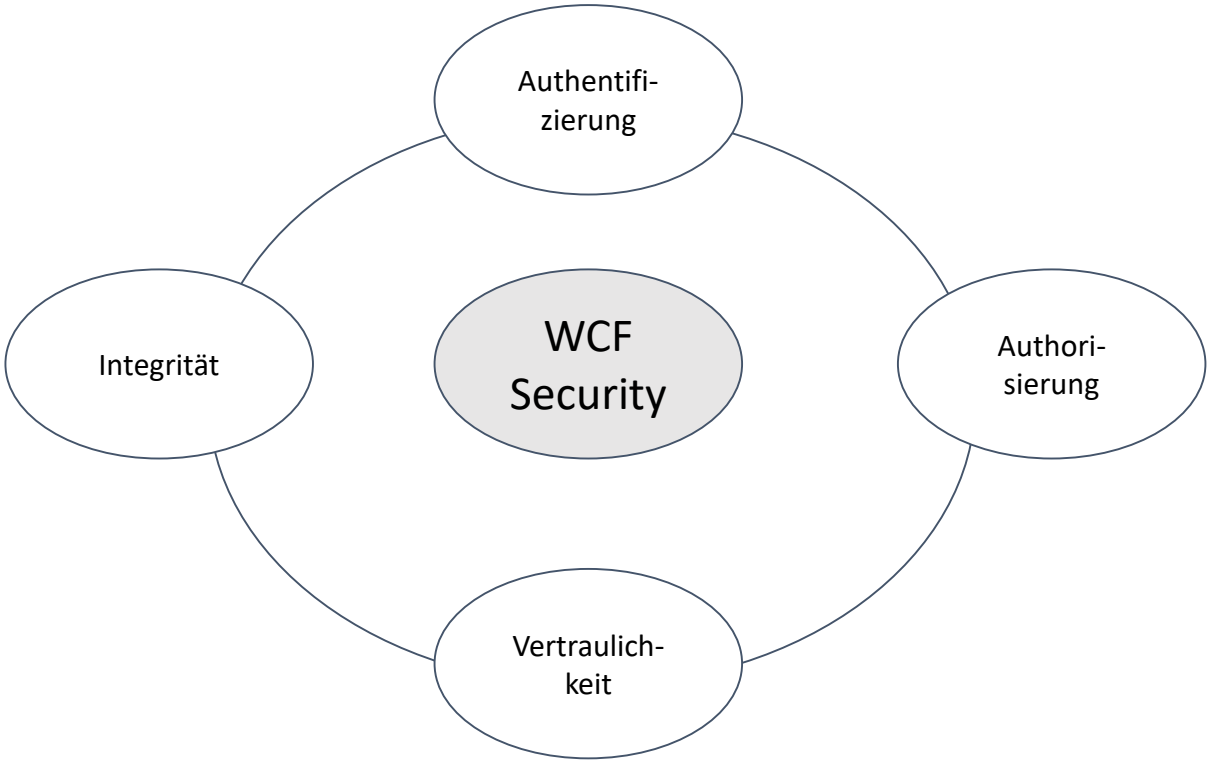
WCF‘s eingebaute Bindings

- NetXYZ Bindings sind speziell für .NET-to-.NET Kommunikation

Binding Name	Szenario der Kommunikation
WebHttpBinding	Interoperable RESTful Kommunikation mit HTTP
BasicHttpBinding	Interoperable SOAP Kommunikation mit HTTP, bietet „basic“ Protokolle nach WS-I Basic Profile Standard
WSHttpBinding	Interoperable SOAP Kommunikation mit HTTP, bietet die volle SOAP + WS-* Protokoll Palette
NetTcpBinding	Cross-Maschine WCF Kommunikation mit TCP
NetPeerTcpBinding	Cross-Maschine WCF Kommunikation mit P2P
NetNamedPipesBinding	Same-machine WCF Kommunikation mit IPC
NetMsmqBinding	Disconnected/Asynchronous WCF Kommunikation mit MSMQ

Security

Security Features



WCF Security Overview

- Service Calls sollten gesichert sein
  - Authentifizierung des Aufrufers
  - Autorisierung, was dürfen die Aufrufer auf der Serverseite
- Die meisten WCF Bindings sind defaultmäßig gesichert, durch die Windows Authentication und Authorization
  - BasicHttpBinding is not secure by default

WCF Authentifizierung

- Verschiedene Arten von Client-Credentials
  - Windows, Username, Certificate, Tokens
- Möglichkeiten zur Credential-Validierung:
  - Windows, Membership Provider, Security Token Service, Custom Hooks

WCF Autorisierung

- Nach der Validierung erstellt WCF den Thread nach IPrincipal Client Identity
  - Thread.CurrentPrincipal static Property
- Kann programmatisch die Iprincipal.IsInRole() aufrufen
- [PrincipalPermission] Attribut kann in Service Methode gesetzt werden
  - [PrincipalPersmission(SecurityAction.Demand, Role=„BUILTIN\\Administrator“)]
- ClaimsPrincipal zum feststellen, ob Aufrufer über bestimmte Claims verfügt
- Custom ServiceAuthorizationManager als Alternative

WCF Windows Authentifizierung

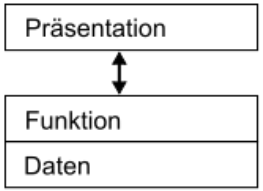
- Client schickt Windows Identity zum Service
  - Client verarbeitet Identity
  - Alternative Credentials werden durch Client Proxy gesetzt
- WCF validiert automatisch die Client Identity am Server mithilfe des Windows Network Security Protokolls
  - NTLM
  - Kerberos
- Principal kann für Autorisierung verwendet werden
  - Var principal = Thread.CurrentPrincipal;
    if (!principal.IsInRole(„BUILTIN\\Administrators“)) trow new SecurityException(„No“)

Demo: WCF Security

- Alternative Windows Credentials:
- Service:
  - Proxy.ClientCredentials.Windows.ClientCredential.UserName = „XPS9100\\test“;
  - Proxy.ClientCredentials.Windows.ClientCredential.Passwort = „test“;
- Client:
  - [PrincipalPermission(SecurityAction.Demand, Role = („BUILTIN\\Users“))]

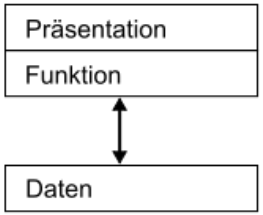
Schichtenarchitektur

- unterschiedliche Aufteilung



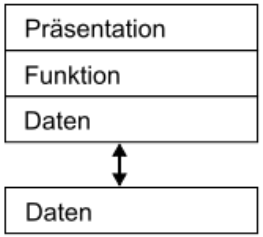
THIN-CLIENT, AKTIVER SERVER
+ zentrale Administration & Wartung
+ Kostenersparnis
+ Sicherheit (nur 1 Server muss geschützt werden)
+ Flexibilität

- hohe Belastung für Server



DATEN-SERVER
➔ Server liefert nur benötigte Daten
+ jeder Client kann selber rechnen ➔ höhere Performance
+ zentrale Datenhaltung bleibt bestehen ➔ Sicherheit

- hoher Installationsaufwand, da jeder Client alle Applikationen braucht

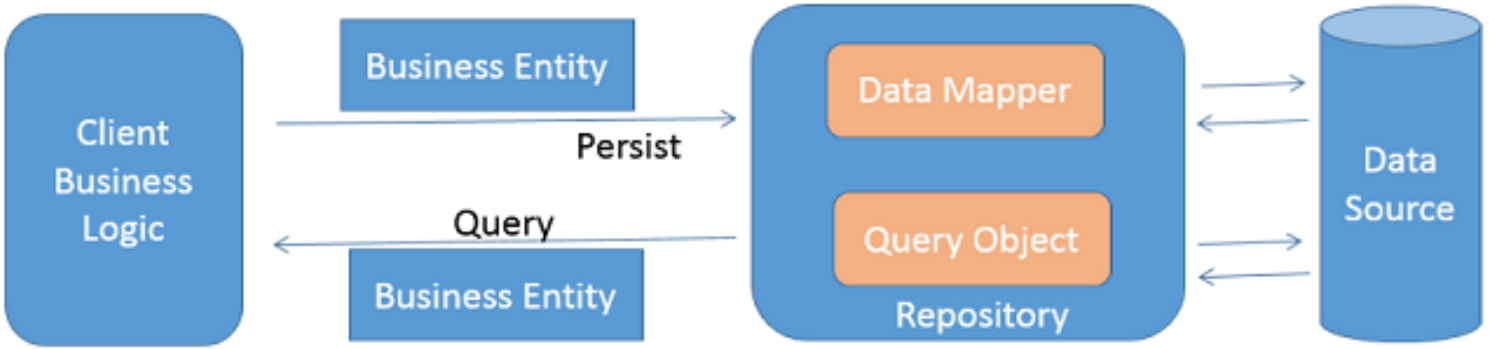


FAT-CLIENT
+ Entlastung Server & Datenleitung durch Plug-Ins
+ Weiterarbeiten möglich, wenn Kommunikation zum Server gestört

- lange Dauer, bis alle Clients Updates haben
- hohe Wartungskosten

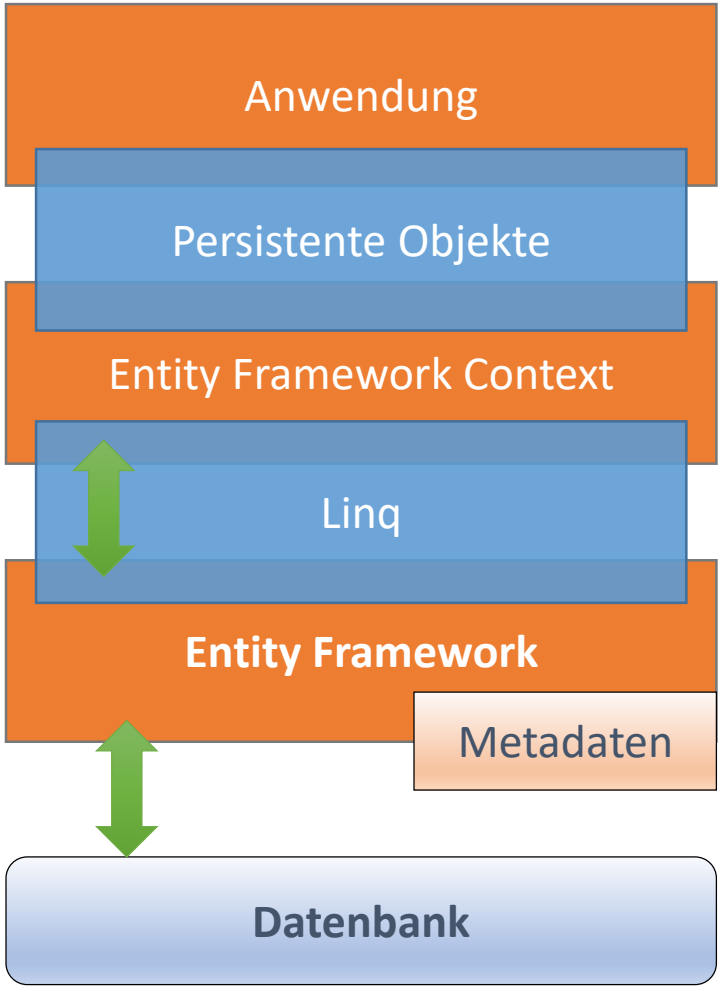
Quelle: https://de.wikipedia.org/wiki/Schichtenarchitektur

Repository Pattern



Exkurs: Entity Framework

- Entity Framework
  - O/R Mapping Framework
  - Übersetzt LINQ in SQL
  - Abstrahiert ADO.NET
  - Ermöglicht es, Daten nur bei Bedarf zu laden (lazy loading)
  - Der Einsatz von O/R Mappern spart viel Zeit!



Übung Entity Framework

Übung: Schichtenarchitektur

- Entities Layer
- Data Layer
- Services Layer
- Self-Hosting

WCF Client

- Zur WCF Konsumierung benötigt der .NET ein Client Proxy
- Proxies können von VS generierter Code sein, der auf den Service Metadaten basiert
  - Service muss Metadaten durch WSDL oder WS-MetadataExchange am Endpoint bereitstellen
- Proxies können auch händisch programmiert werden
- Aufruf des Services über Proxy Instanzmethoden
  - Öffnet Verbindung zum Service
  - Request/Response mittels SOAP Nachrichten

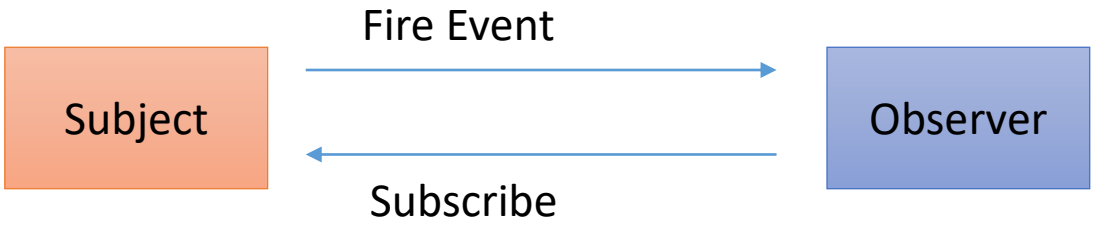
Demo: Windows Service

- install: installutil MyService.exe
- uninstall: installutil /u MyService.exe
- view services: services.msc

- Auf der Console Tooling:
  - Sc /?

- https://docs.microsoft.com/en-us/dotnet/framework/wcf/feature-details/how-to-host-a-wcf-service-in-a-managed-windows-service

Projekt: Publisher-Subscriber Design Pattern



- https://docs.microsoft.com/en-us/dotnet/framework/wcf/samples/design-patterns-list-based-publish-subscribe

REST

SOAP != REST

- REST unterscheidet sich von SOAP in der Nachrichtenkodierung
  - SOAP = XML, Nachricht im Body
  - REST = Message in der URL
  - SOAP => POST da die Message im Body der Nachricht ist
- SOAP ist flexibel in der Wahl des Protokolls
- REST verwendet ausschließlich HTTP als Protokoll
- REST ist Stateless

REST Zugriff auf Ressourcen

- GET holt die Ressource vom Server ab: GET http://ppedv.de/brezn/123
- POST erstellt eine Ressource: POST http://ppedv.de/brezn/123
- PUT verändert (=*update*) ein Objekt in einer Ressource: PUT http://ppedv.de/brezn/123?date=20180209
- DELETE löscht eine Ressource vom Server
- Das Datenformat im GET darf sein: XML, JSON, RSS

- Performance Vorteil durch schnellen Zugriff ohne WSDL!

REST Operationen (mittels HTTP)

HTTP Methode	CRUD Methode	Beschreibung
GET	Read	Gibt einen einzelnen oder eine Sammlung von Einträgen zurück.
POST	Create	Erzeugen eines neuen Eintrags.
PUT	Update	Update (Aktualisierung) eines bestehenden Eintrags.
DELETE	Delete	Löschen eines Eintrags.
PATCH	Update	Teile eines Eintrags werden geändert / aktualisiert
Head	N/A	Liest die Message Header anhand der URI aus

SOAP oder REST?

- REST
  - Leicht Erweiterbar und auf das Web optimiert
  - Interoperabilität – jeder kann HTTP verwenden
  - Performant
  - XML, RSS, JSON
  - Kaum Koppelung zwischen Client und Server
- SOAP
  - Verschiedene Protokolle zur Verfügung
  - Vorteil oder Nachteil: Inkompatibilität der Ressource z.B. net.tcp performanter als http, aber nur .NET Welt
  - Nicht Zustandslos mittels [InstanceContextMode]

Demo: REST Endpunkte

- IIS Self Hosting

Nützliche Links

- WCF Website:
  https://docs.microsoft.com/en-us/iis/web-hosting/configuring-servers-in-the-windows-web-platform/windows-communication-framework-wcf
- WCF Windows Service Tutorial
  https://docs.microsoft.com/en-us/dotnet/framework/wcf/feature-details/how-to-host-a-wcf-service-in-a-managed-windows-service
- WCF API Resource:
  https://docs.microsoft.com/en-us/dotnet/framework/wcf/wcf-and-aspnet-web-api