TECHNISCHE UNIVERSITÄT DRESDEN

Vorlesung „Service and Cloud Computing“

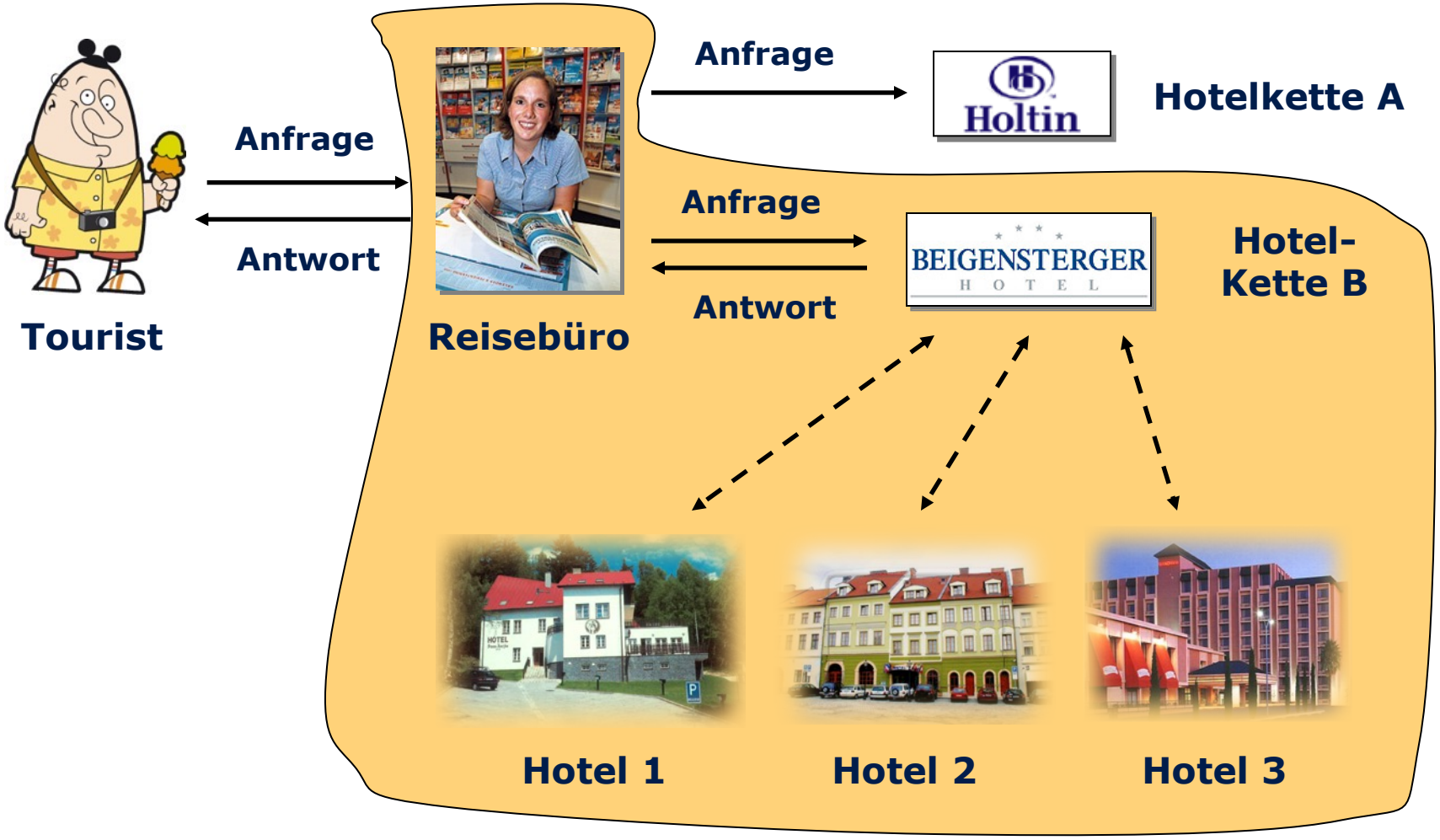
2. Implementierung von Services und Clients

Dr.-Ing. Iris Braun

Gliederung

- Anwendungsszenarien, Wiederholung SOA-Aspekte
- geeignete Technologien für SOA
- Client-Server-Paradigma
- RESTful Services
- Web Services Basic Protocol Stack
- Implementierung von Web Services und Clients

Szenario


Anfrage
Antwort
Tourist
Reisebüro
Anfrage
Holtin
Hotelkette A
Anfrage
Antwort
BEIGENSTERGER
HOTEL
Hotel-
Kette B
Hotel 1
Hotel 2
Hotel 3

Szenario: Aspekte von SOA

Reisebüro
(Dienstnutzer)

Interaktion

Anfrage

Antwort

BEIGENSTERGER HOTEL

Hotelkette (Dienst-Anbieter)

Transparente Verarbeitung

Hotel 1 Hotel 2 Hotel 3

Szenario: Aspekte von SOA

Reisebüro
(Dienstnutzer)

Anfrage ?

Antwort ?

BEIGENSTERGER HOTEL

Hotelkette (Dienst-Anbieter)

Sichtbarkeit

!

XML Dienstbeschreibung

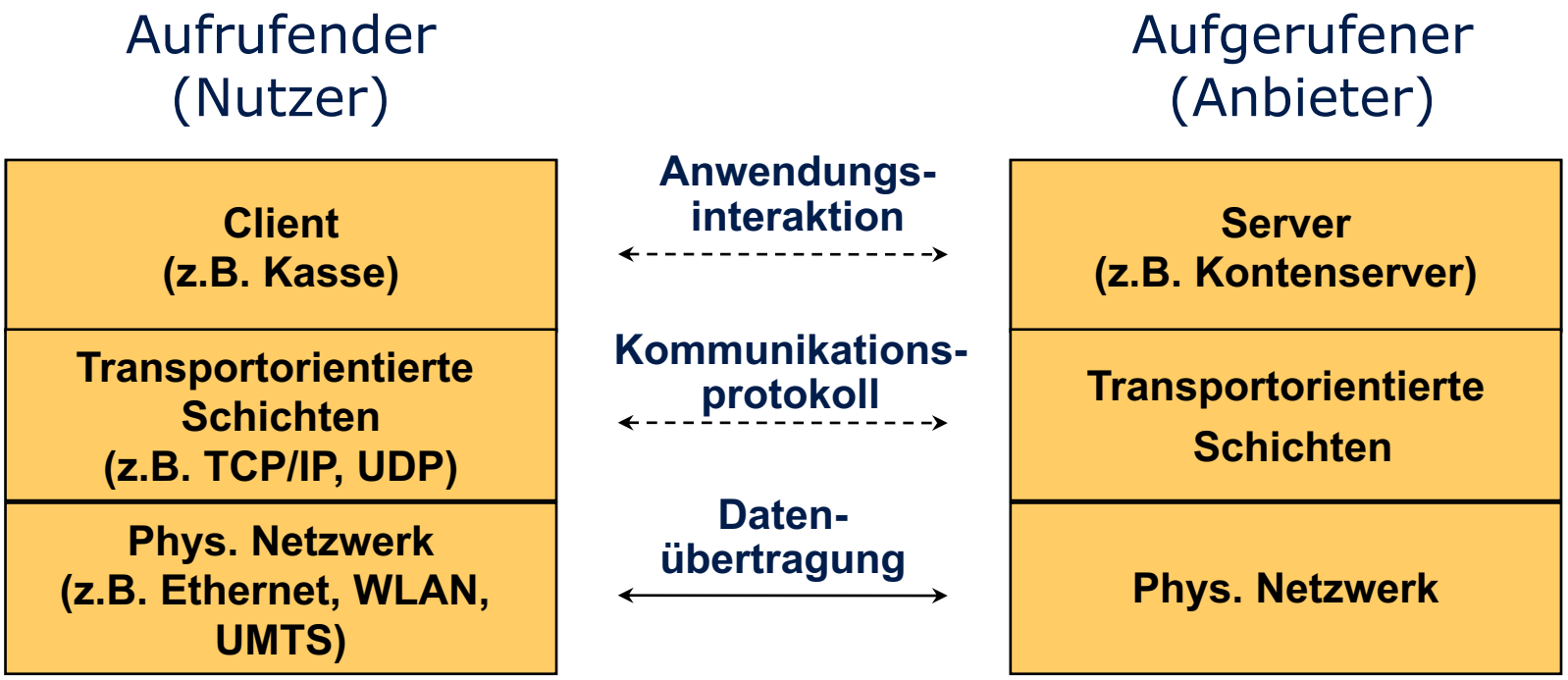
Operationen:
- **Hotels suchen**
- **Gefundenes Hotel buchen**
- **Buchung stornieren**

Bewertung der Technologien zur Umsetzung von SOA

- **Anforderungen:**
  - ***Effect***: Funktionalität kapseln
  - ***Visibility***: Über wohldefinierte Schnittstelle anbieten
  - ***Interaction***: Lose Kopplung über Nachrichtenaustausch
  - Interoperabilität
  - Transparenz der zugrundeliegenden Implementierung

- **Geeignete Technologien?**
  - XML RPC
  - Web Services
  - RESTful Services

Client-Server-Paradigma



TECHNISCHE UNIVERSITÄT DRESDEN

2. Implementierung von Services und Clients

RESTful Services

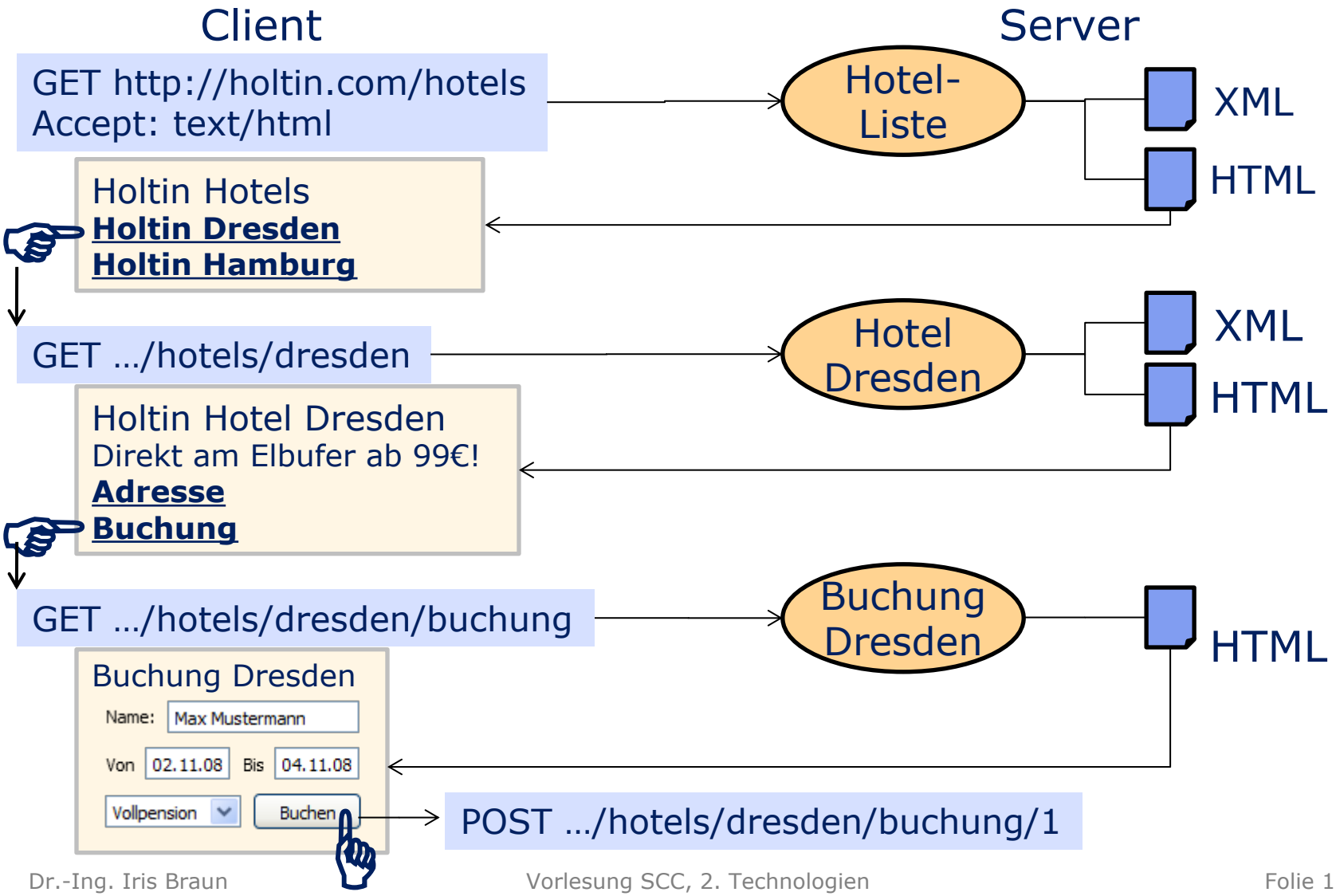
REST: Representational State Transfer

- Architekturstil, beschrieben in der Dissertation von Roy T. Fielding [Fie00]
- Client verändert seinen Anwendungszustand, indem er Repräsentationen von Ressourcen von einem Server bezieht

Grundlegende Konzepte:

- Eindeutige Identifikation von Ressourcen (z.B. über URIs)
- Repräsentationen von Ressourcen (z.B. in XML, JSON)
- Selbstbeschreibende Nachrichten (z.B. mittels HTTP)
- Hypermedia zur Veränderung des Anwendungszustands

RESTful Services: Hypermedia



REST: Uniform Interface

- Bedingung: alle Komponenten nutzen einheitliche Schnittstelle, d.h. begrenzte Anzahl an Methoden (CRUD)

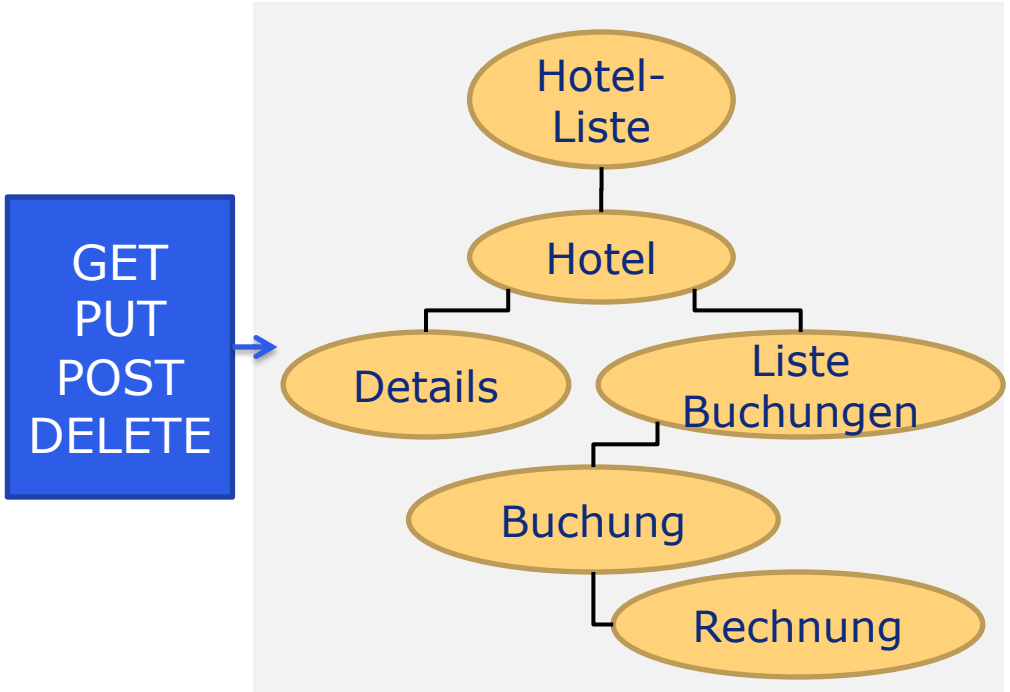
Beispiel SOAP Web Service

http://holtin.com/HotelService

- getHotelList
- checkRoomAvailability
- bookHotelRoom
- cancelBooking
- obtainInvoice

- Z.B. cancelBooking = DELETE Buchung

REST Web Service

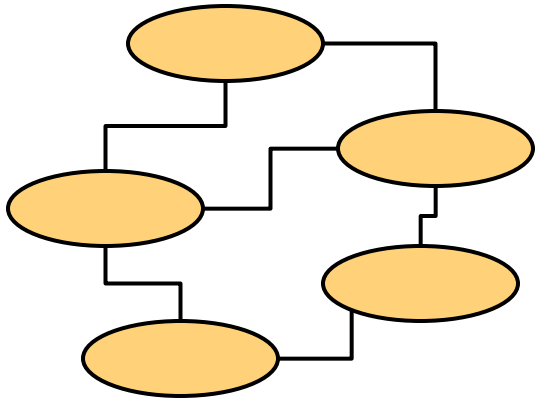


Das REST-Missverständnis

- Häufig: „REST ist XML über HTTP“ (HTTP+POX: „HTTP and plain old XML“)
  - Query-Parameter kaschieren spezielle Methodenaufrufe, oft gibt es nur eine Ressource, die die gesamte Funktionalität bereitstellt
- Kein REST im eigentlichen Sinn!
- Besser: Ressourcen-orientierte Architektur (ROA) [RR07]

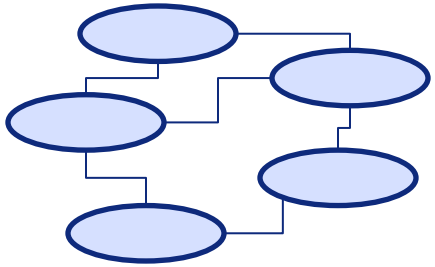
```
GET …/rest?action=cancelbooking&bid=1
Accept: application/xml
…
```

```
<?xml version="1.0"?>
<booking id="1">
  <state>cancelled</state>
</booking>
```

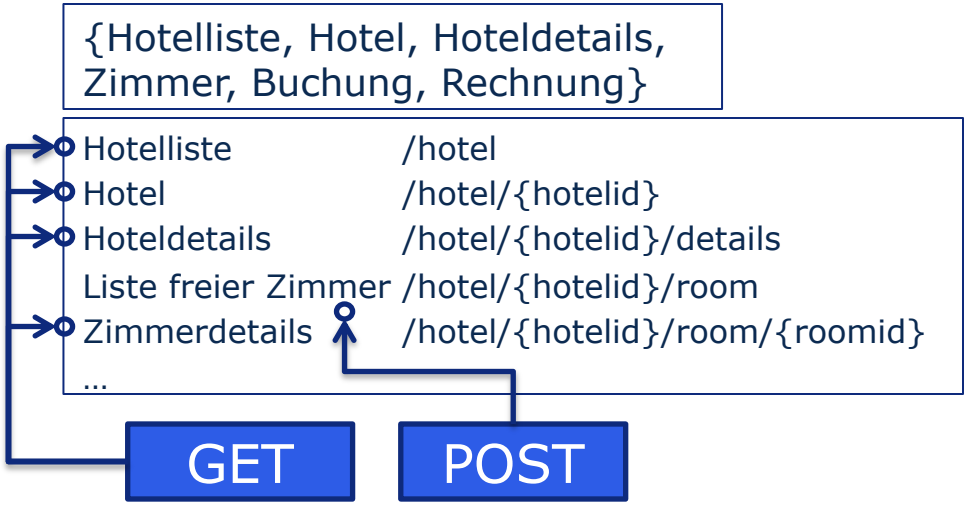


REST: Entwurf von Ressourcen

1. Datenbestand feststellen
2. Datenbestand in Ressourcen unterteilen
3. Ressourcen mit URIs benennen
4. Untermenge der Schnittstelle für jede Ressource festlegen
5. Repräsentationen entwerfen
6. Integration der Ressourcen durch Hyperlinks



Nach [RR07]



Arten von Ressourcen:

- Top-Level-Ressourcen (dürfen nicht entfernt werden)
- Objekt-Ressourcen
- Algorithmusergebnis-Ressourcen (z.B. Abbildungen von Such-Algorithmen)

REST: Implementierung

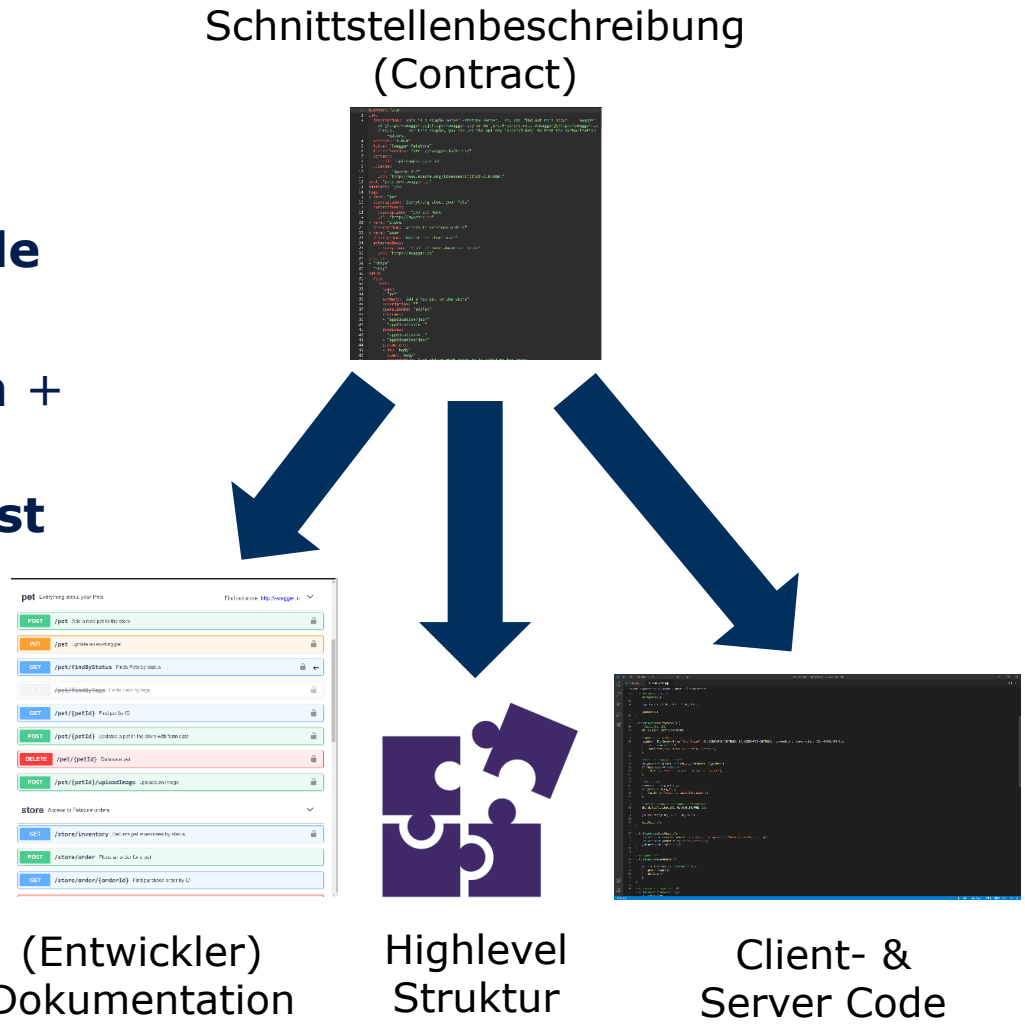
- Grundsätzlich möglich mit jeder HTTP-Bibliothek
  - z.B. Jakarta Commons HttpClient oder Ruby Net::HTTP
- Frameworks abstrahieren von Low-Level-Details
  - Java: Restlet-Framework
  - Ruby on Rails: Active Record enthält REST-Unterstützung
- Standard-Entwicklung
  - Java API for REST Web Services (JAX-RS): JSR-311 (10.10.2008) [JSR311]
- Tool-Unterstützung
  - Mittlerweile breite Werkzeugpalette: Jersey, Restlet, RESTEasy (JBoss), RubyOnRails, PHP Restler, ...
  - NetBeans: REST Web Service automatisch aus JPA-Entitätsklassen generieren

REST: Beispiel (JAX-RS Hello World)

```
Path("/helloWorld")
public class HelloWorldResource {
    Context
 private UriInfo context;
 private static String myName = "unknown";
    GET
    ProduceMime("text/html")
 public String getXml() {
     return "<html><body><h1>Hello World, this is "
     + HelloWorldResource.myName + "</body></h1></html>";
 }
    PUT
    ConsumeMime("text/plain")
 public Response putXml(String content) {
     HelloWorldResource.myName = content;
     return Response.ok().build();
 }
```

REST Schnittstellenbeschreibungen

- Engl. **I**nterface **D**escription **L**anguages (**IDL**s)
- Definieren die **Schnittstelle**
- **Referenz** für Entwickler
- Generiert **Dokumentation** + mehr
- Motiviert zur **Contract-First** Entwicklungsmethode

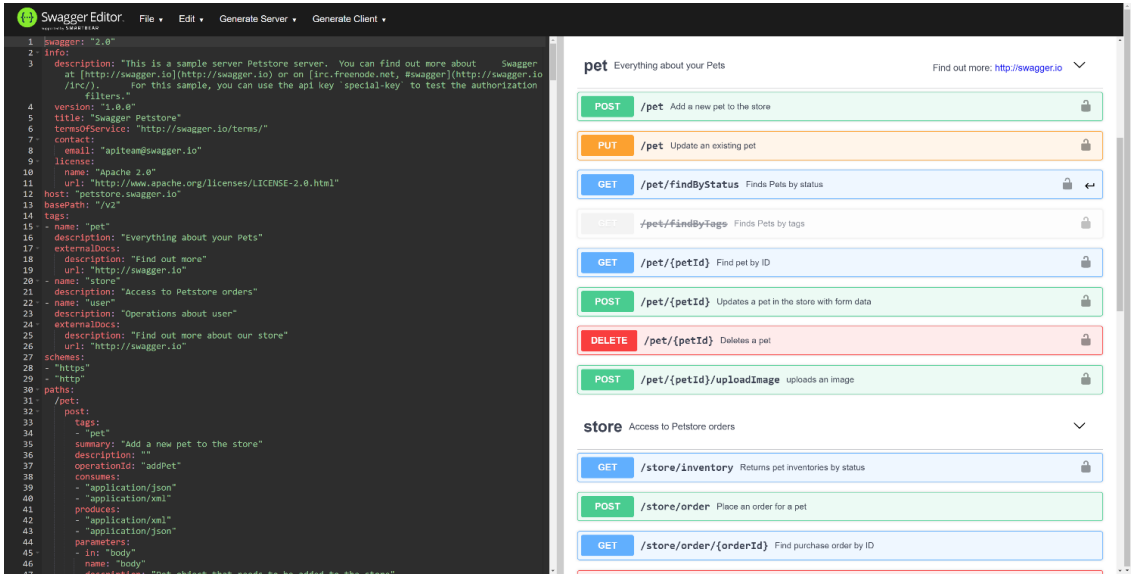


REST Schnittstellenbeschreibungen
OpenAPI + Swagger

- Open-source
- Gehört zur **OpenAPI-Initiative**
- browserbasierter Editor
- Kann Code in vielen Sprachen generieren

Swagger™
Supported by SMARTBEAR

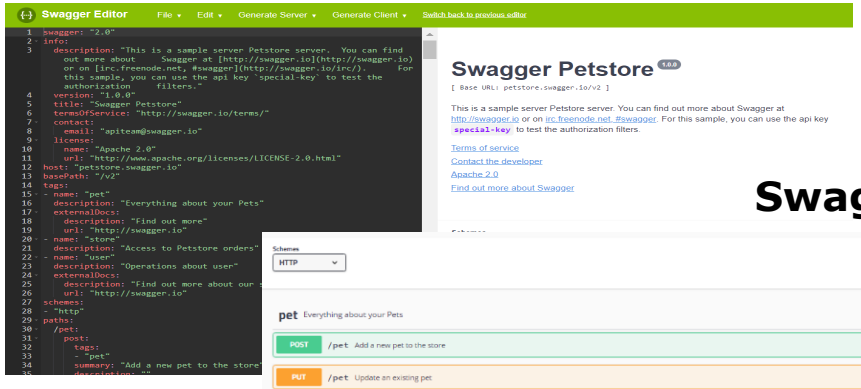
Quellen: [https://www.openapis.org/]



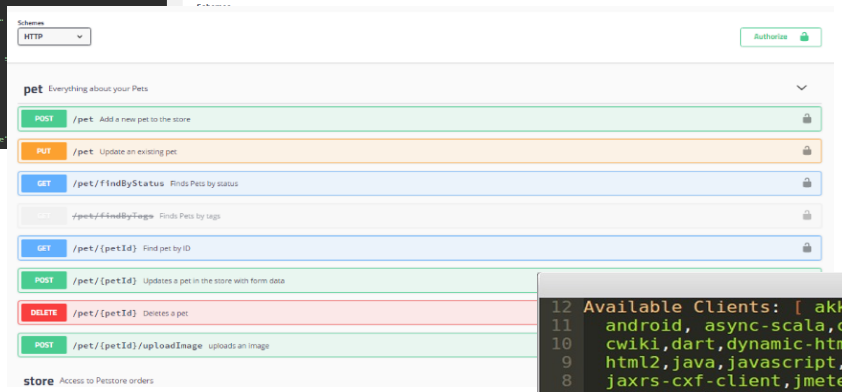
Screenshot von [https://swagger.io/]

REST OpenAPI + Swagger

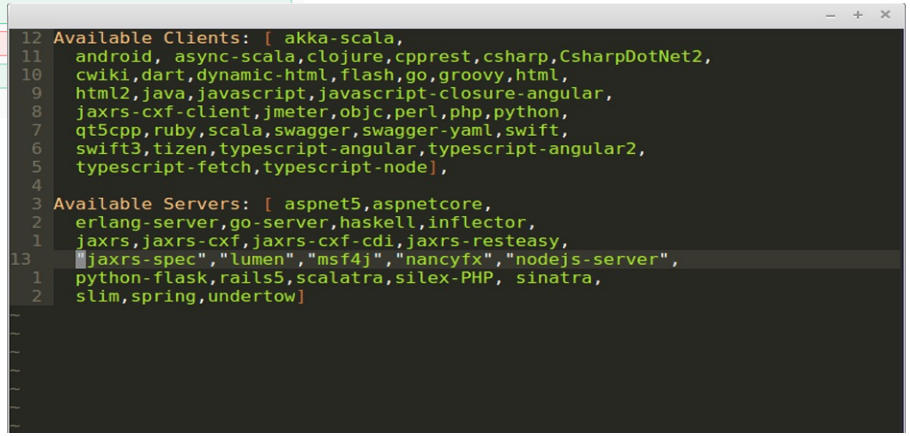
Swagger Editor



Swagger UI



Swagger Codegen



Screenshots von [https://swagger.io/]

REST OpenAPI + Swagger

Swagger™
Supported by SMARTBEAR

Format	Code gen. (Client)	Code gen. (Server)	Unterstützt durch
YAML oder **JSON**	**JS/TS/NodeJS**, **Java**, **C#**, C++, Go, PHP, Swift, Lua, …	**NodeJS**, **Java** (Spring Boot, Undertow, JAX-RS, …), **C#**, C++, Go, PHP, Rust, …	SmartBear Software, **OpenAPI Initiative** (Linux Foundation)

TECHNISCHE UNIVERSITÄT DRESDEN

2. Implementierung von Services und Clients

Web Services

Web Services: Begriffsdefinition

- World Wide Web Consortium (W3C):

  „A Web service is a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using the XML-based messages exchanged via Internet-based protocols."

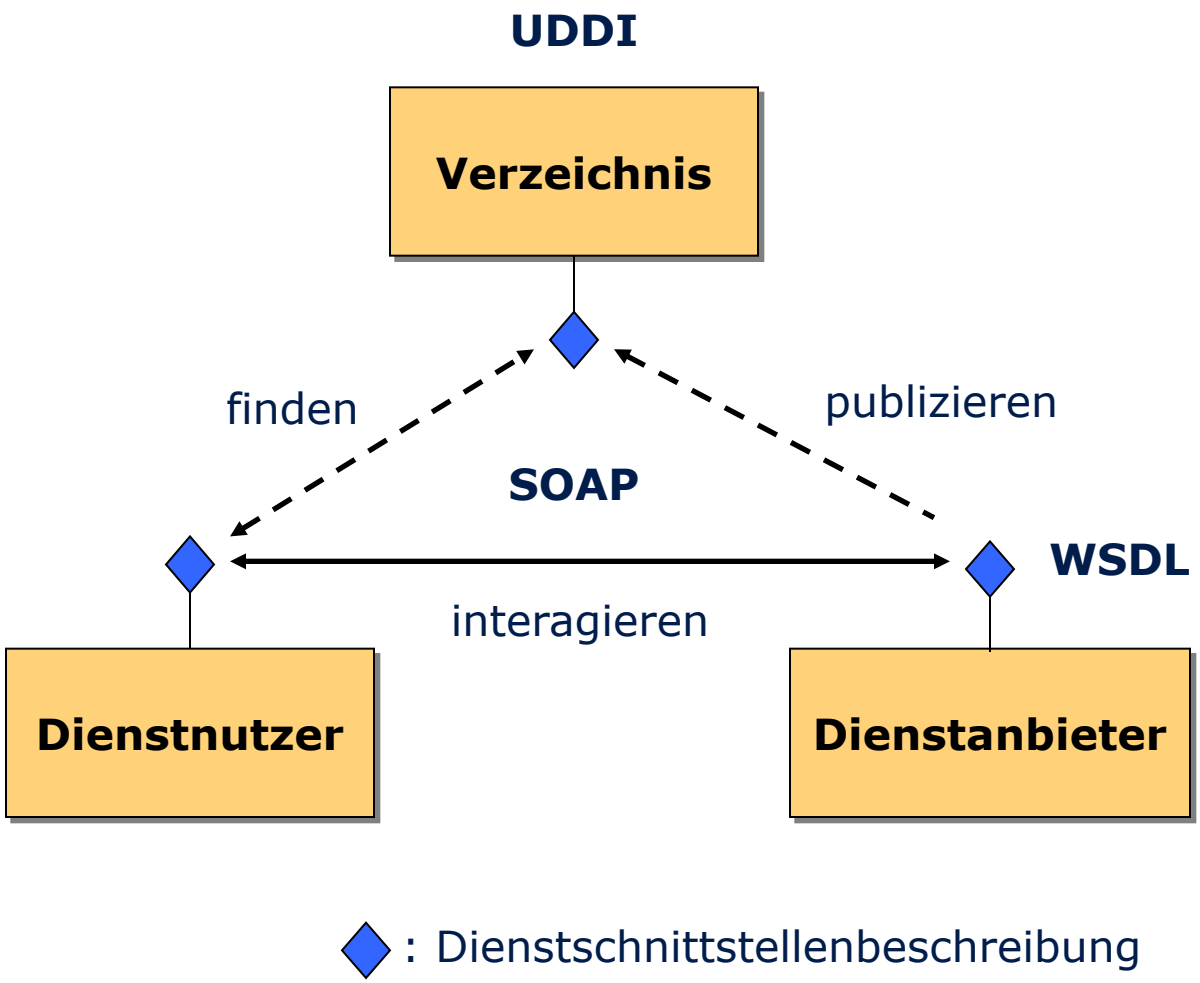
- IBM:

  „Web Services are self-contained, modular applications that can be described, published, located and invoked over a network, generally, the World Wide Web."

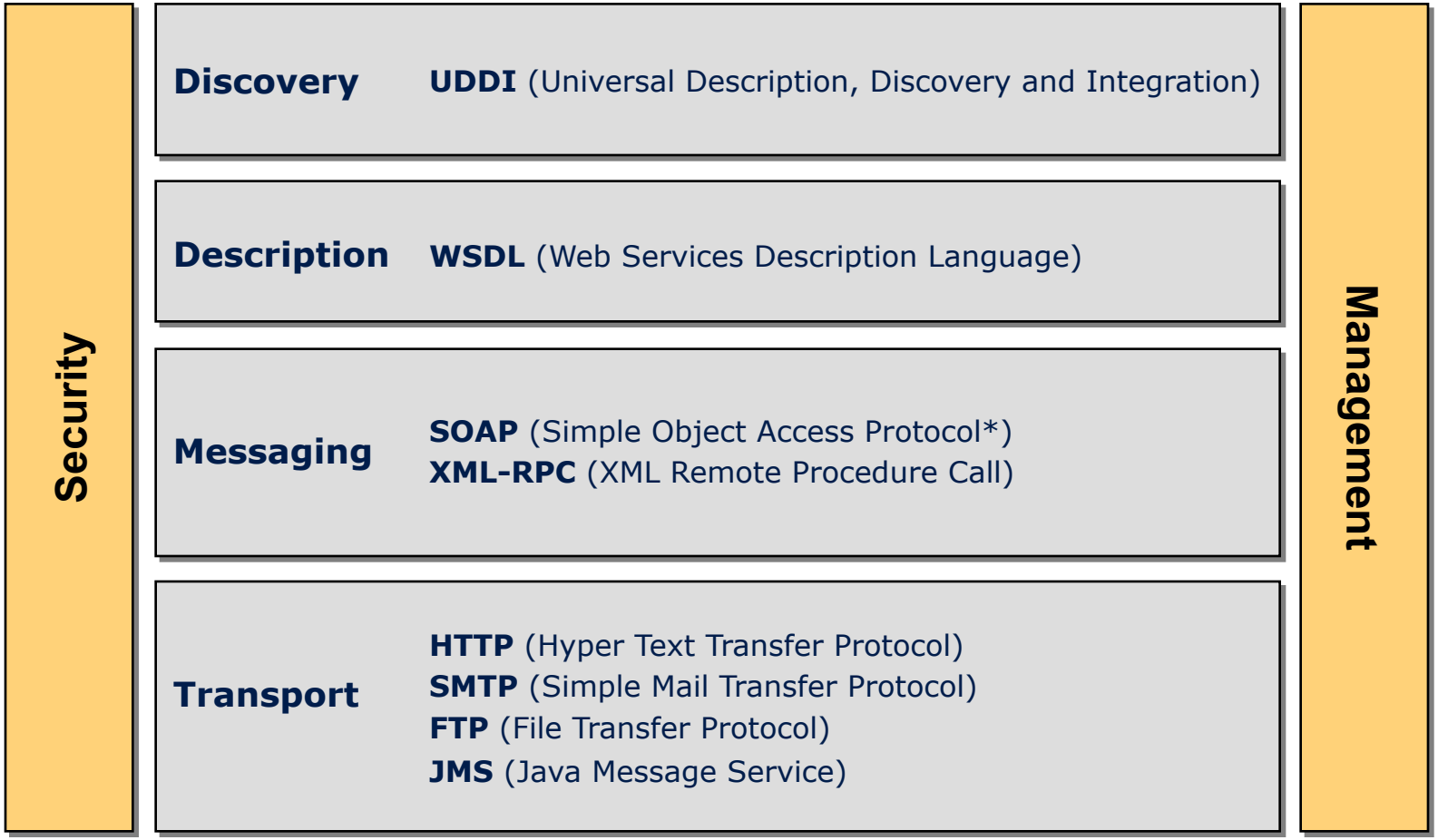
- Cerami:

  „A Web Service is any piece of software that makes itself available over the Internet and uses a standardized XML messaging system."

Web Services: Architektur



Web Services: Basic Protocol Stack

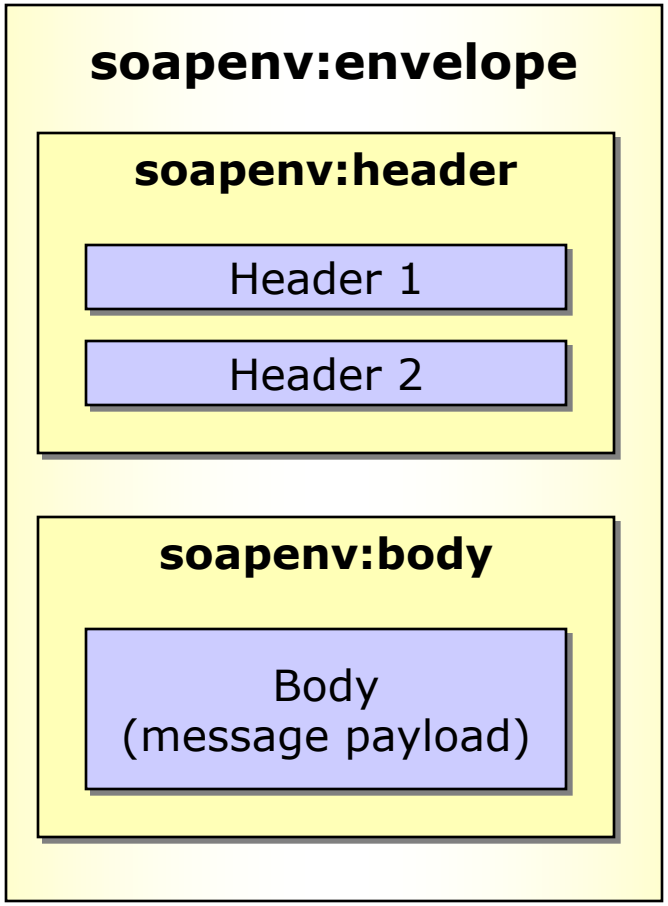


*: ursprüngliche Abkürzung, die nicht mehr verwendet wird

SOAP

- W3C-Standard, aktuelle Version: 1.2
- Setzt auf vorhandenen Protokollen auf: HTTP, HTTPs, SMTP, ...

- SOAP-Envelope: besteht aus
  - Header: optional, enthält Meta-Informationen wie Adressierung, Transaktions- und Verschlüsselungsattribute
  - Body: enthält „Nutzdaten“ und Bereich für Fehlermeldungen (fault messages)

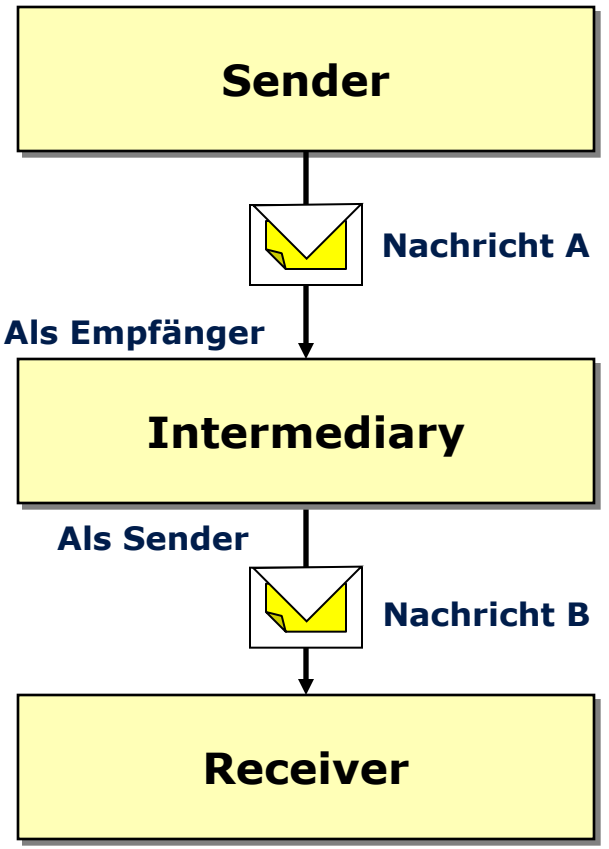


SOAP-Nachrichten: Beispielanfrage

```
<soap:Envelope xmlns:soap=...>
   <soap:Header>
     <m:land xmlns:m="..." soap:mustUnderstand="0"
     soap:actor="next">Deutschland</m:land>
   </soap:Header>
   <soap:Body soap:encodingStyle=...>
     <tns:doHotelSearch xmlns:tns=...>
       <m:stadt xsi:type="xsd:string">Dresden</stadt>
     </tns:doHotelSearch>
   </soap:Body>
</soap:Envelope>
```

SOAP

- Teilnehmer eines SOAP-Austauschs:
  - Sender
  - Intermediary (Vermittler)
  - Receiver (Empfänger)
- Vermittler erscheint je nach Sicht als Empfänger oder als Sender
- Wenn A≠B: Active Intermediary
  - Vermittler verändert Header oder Body der SOAP-Nachricht (z.B. Verschlüsselung, Re-Routing zum Lastausgleich)

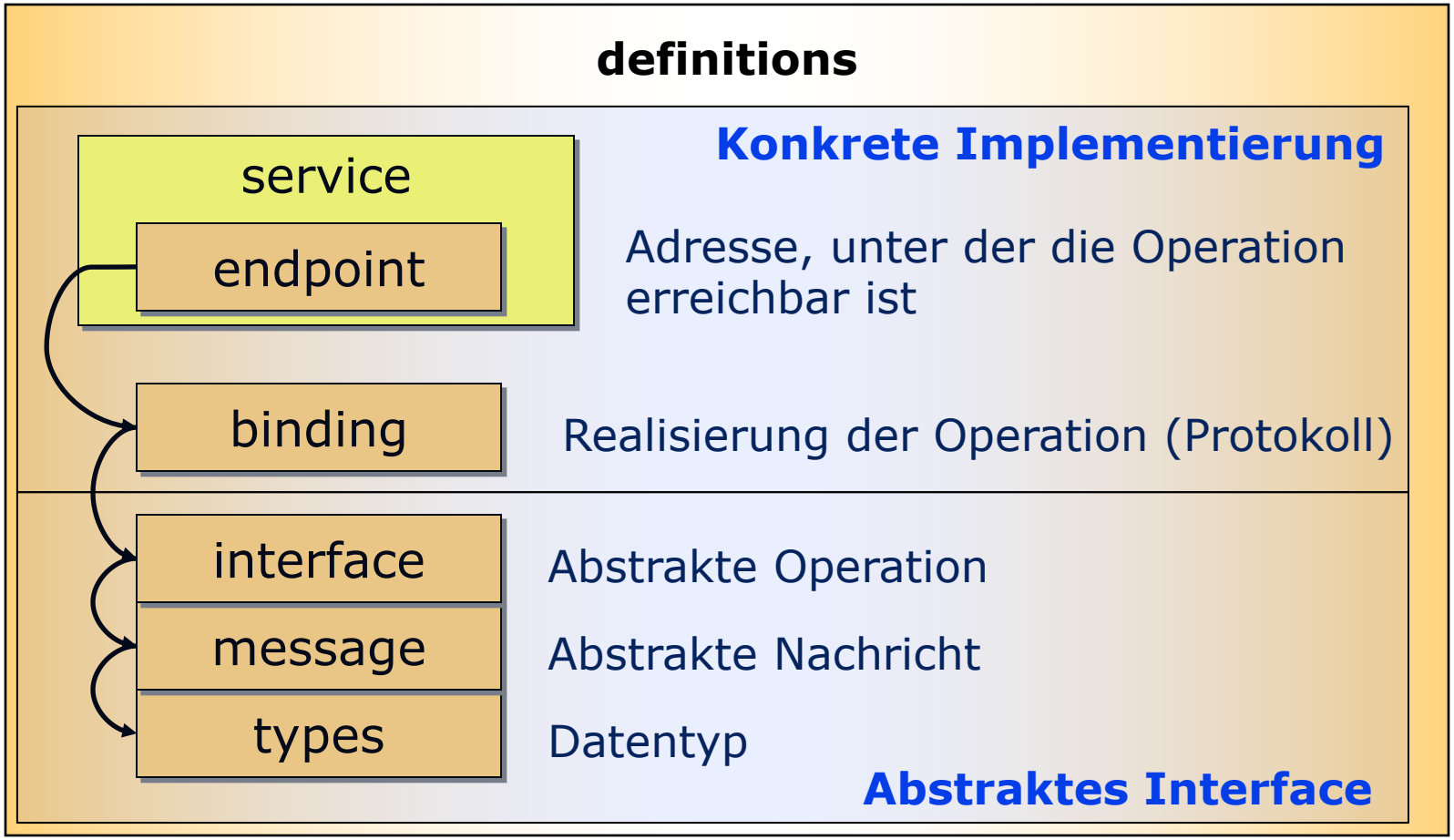


WSDL

- ***Web Services Description Language***: allgemeingültige, programmiersprachenunabhängige Schnittstellenbeschreibung
- W3C-Standard, aktuelle Version: Version 2.0 (2007)

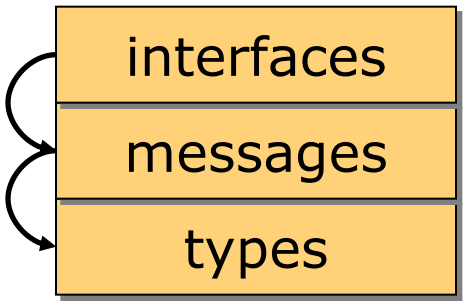
- **6 Hauptelemente (WSDL 2.0):**
  - `Types`: alle Datentypen, die zwischen Client und Server ausgetauscht werden
  - `Message`: abstrakte Beschreibung der Nachrichten
  - `Interface`: Menge von abstrakten Operationen
  - `Binding`: legt konkretes Protokoll und Datenformate für Operationen und Nachrichten eines Port Types fest
  - `Endpoint`: Adresse für ein Binding (Kommunikations-Endpunkt)
  - `Service`: Menge von Endpoints

WSDL: Zusammenfassung



Dienstbeschreibung mit WSDL

- Dienst Hotelbuchung: beschrieben in **`hotelbooking.wsdl`**
- Vorgehensweise:
  - Identifikation und Definition der benötigten Datenstrukturen (**`Types`**)
  - Definition entsprechender Nachrichten (**`Messages`**), jeweils unterteilt in **`Parts`** (Parameter eines bestimmten Typs)
  - Definition von Operationen (**`Operations`**) nach einem bestimmten MEP (Message Exchange Pattern, Nachrichtenaustausch-Muster)
  - Gruppieren der Operationen in **`Interfaces`**



Abstrakte Operationen

Abstrakte Nachrichten

Datentypen

Dienstbeschreibung: WSDL-Struktur

```
<?xml version="1.0">
<wsdl:definitions xmlns:"http://schemas.xmlsoap.org/wsdl/"
   name="HotelBooking" targetNamespace="urn:HotelBooking">
```

Abstrakter Teil

```
  <wsdl:documentation ... />
  <wsdl:types> Schema Imports </wsdl:types>
  <wsdl:message> Nachrichten </wsdl:message>
  <wsdl:interface> Operationen </wsdl:Interface>
```

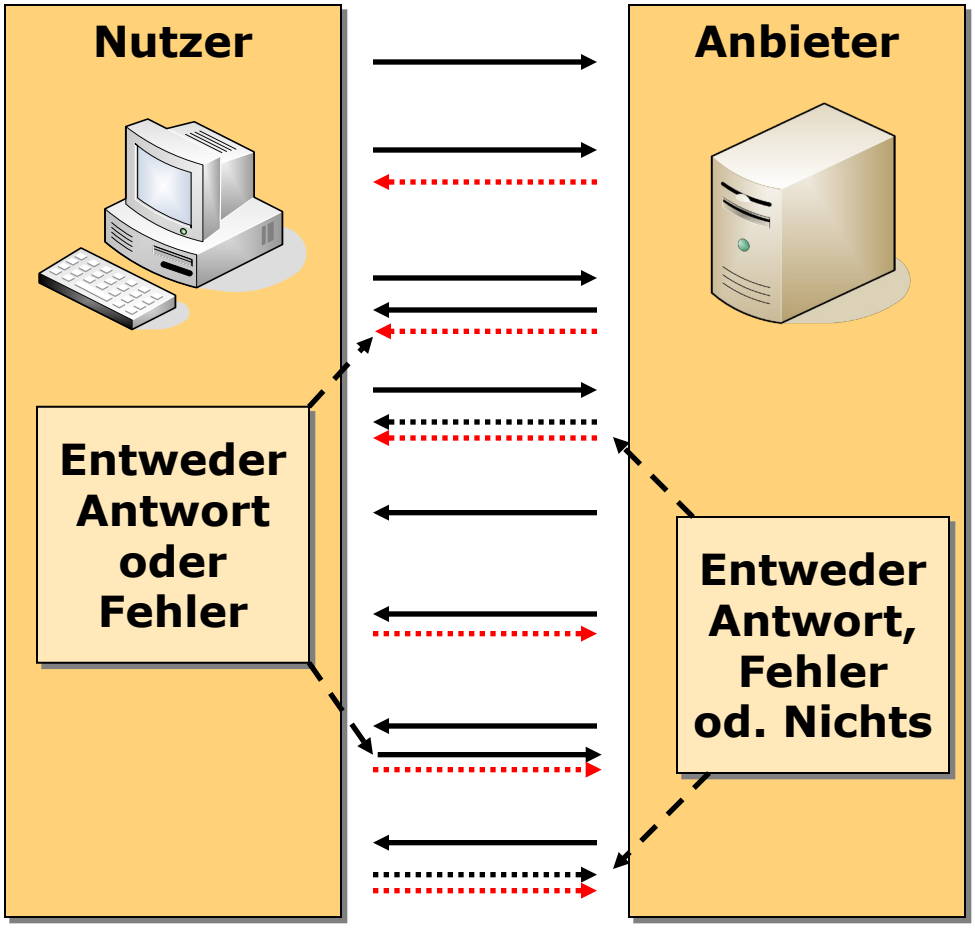
```
  <wsdl:binding> Protokolle und Formate </wsdl:binding>
  <wsdl:service> Service-Definition </wsdl:service>
```

Konkreter Teil

```
</wsdl:definitions>
```

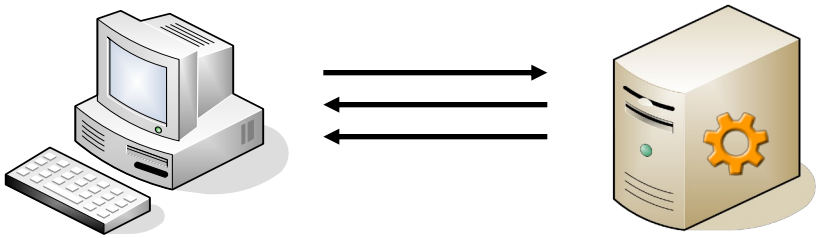
WSDL 2.0 Nachrichtenaustauschmuster (MEP)

- In-Only
  - `~/wsdl/in-only`
- Robust In-Only
  - `~/wsdl/robust-in-only`
- In-Out
  - `~/wsdl/in-out`
- In-Optional-Out
  - `~/wsdl/in-opt-out`
- Out-Only
  - `~/wsdl/out-only`
- Robust Out-Only
  - `~/wsdl/robust-out-only`
- Out-In
  - `~/wsdl/out-in`
- Out-Optional-In
  - `~/wsdl/out-opt-in`



`~: http://www.w3.org/2004/03 ...` für pattern-Attribut der Operation

WSDL: Komplexe Interaktionsmuster



⇒Anfrage an Hotelkette

⇐Bestätigung der Anfrage

⇐Rückgabe von Hotelzimmern (Benachrichtigung)

```
<wsdl:operation name="..."/>
    <wsdl:input message="..."/>
    <wsdl:output message="..."/>
    <wsdl:output message="..."/>
</wsdl:operation>
```

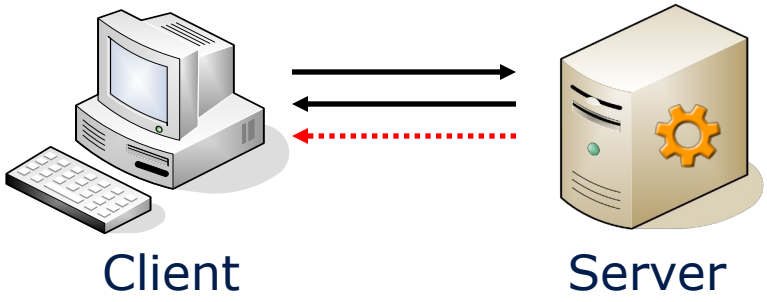
In WSDL
nicht erlaubt!

WSDL: Fehlermeldungen

- Statt weiterer Antwort kann auch Fehler gemeldet werden:

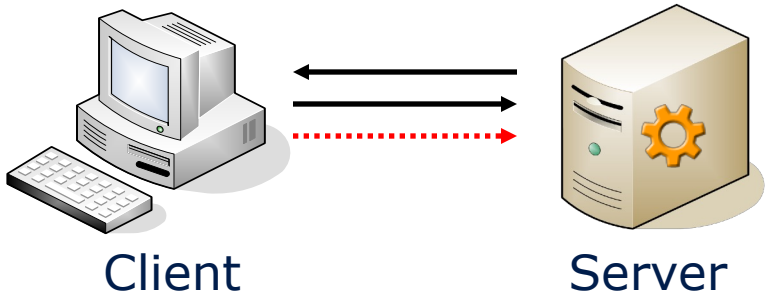
Anfrage-Antwort

```
<wsdl:operation name="..."/>
   <wsdl:input message="..."/>
   <wsdl:output message="..."/>
   <wsdl:fault message="..."/>
</wsdl:operation>
```



Benachrichtigung-Antwort

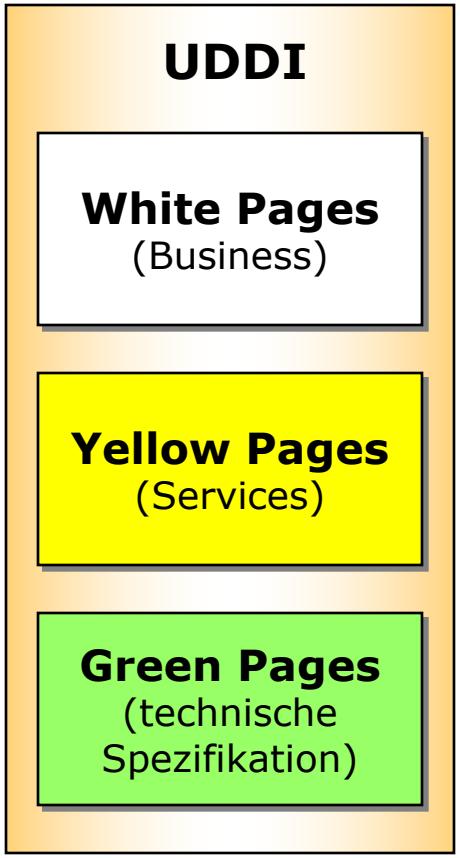
```
<wsdl:operation name="..."/>
   <wsdl:output message="..."/>
   <wsdl:input message="..."/>
   <wsdl:fault message="..."/>
</wsdl:operation>
```



UDDI

- Universal Description, Discovery and Integration
- OASIS-Standard, aktuell Version 3.0
- Ziel: Standardisierte Methode zum Veröffentlichen und Auffinden von Web Services
- 3 Hauptkategorien:
  - **White Pages**: Namensregister, Kontakt zum Anbieter, mehrsprachige Unternehmenvorstellung, Klassifizierungscode (z.B. „Bank")
  - **Yellow Pages**: „Branchenbuch", Einordnung der Web Services in standardisierte Geschäftskategorien (z.B. UNSPSC)
  - **Green Pages**: technische Informationen zu den angebotenen Web Services (tModel), Geschäftsmodell und -prozesse

UNSPSC: United Nations Standard Products and Services Code



UDDI

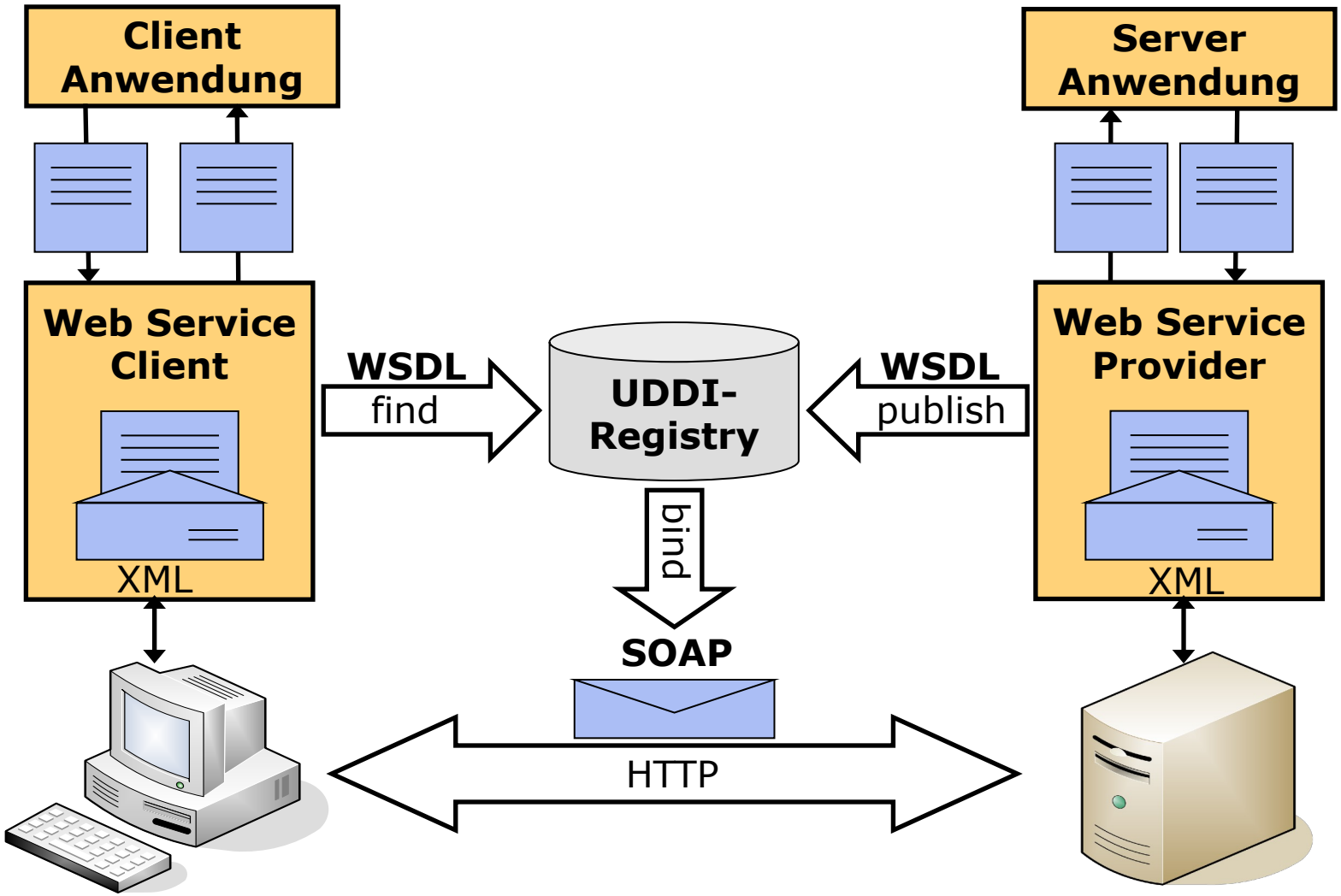
- Mögliche Anfragen an UDDI-Registry (via SOAP-Nachrichten)
  - Authentifikation (bei Nutzeranmeldung)
  - Suchen (Nachrichten zum Auffinden von Diensten, SOAP-Body beginnt mit **`find`**)
  - Detailinformationen abfragen (von Entities)
  - Hinzufügen und Ändern (gleich aufgebaut, automatische Erkennung durch die Registry ob Änderung oder Erstellung)
  - Löschen (erfordert wie Hinzufügen/Ändern Authentifikation)

- Unabhängiges Konsortium verwaltet frei zugängliche UDDI-Register, automatischer Abgleich der öffentlichen Informationen

\+ Globales „öffentliches" Register UBR (UDDI Business Registry)

– Hat aber in den letzten Jahren an Bedeutung verloren (UBR Anfang 2006 eingestellt)

Zusammenfassung Web Services



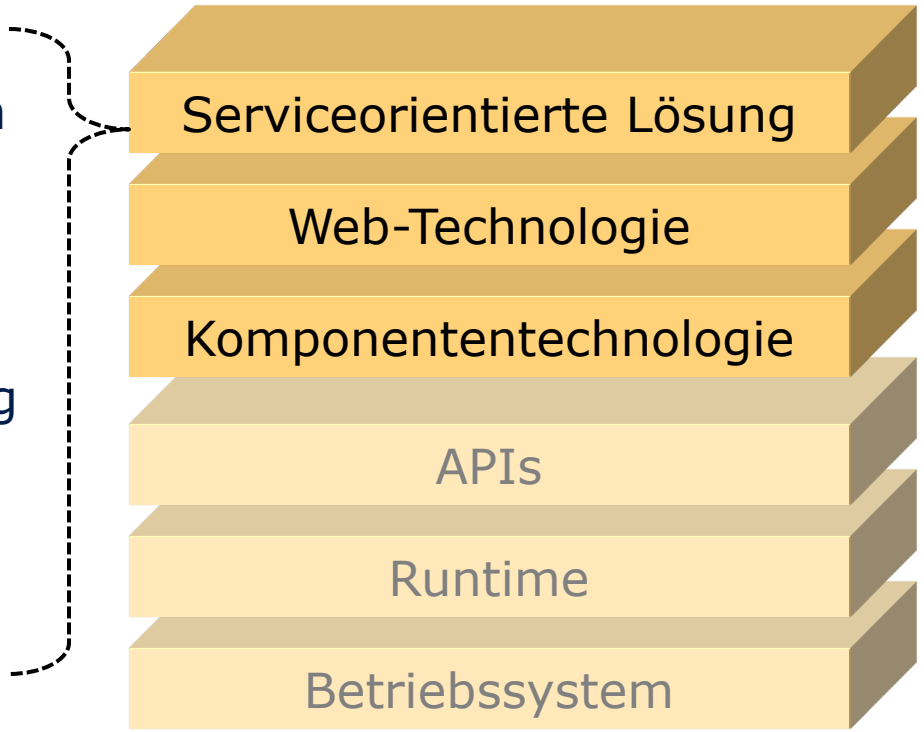
TECHNISCHE UNIVERSITÄT DRESDEN

2. Implementierung von Services und Clients

Web Service und Client Implementierung

Mögliche Implementierungsarchitekturen

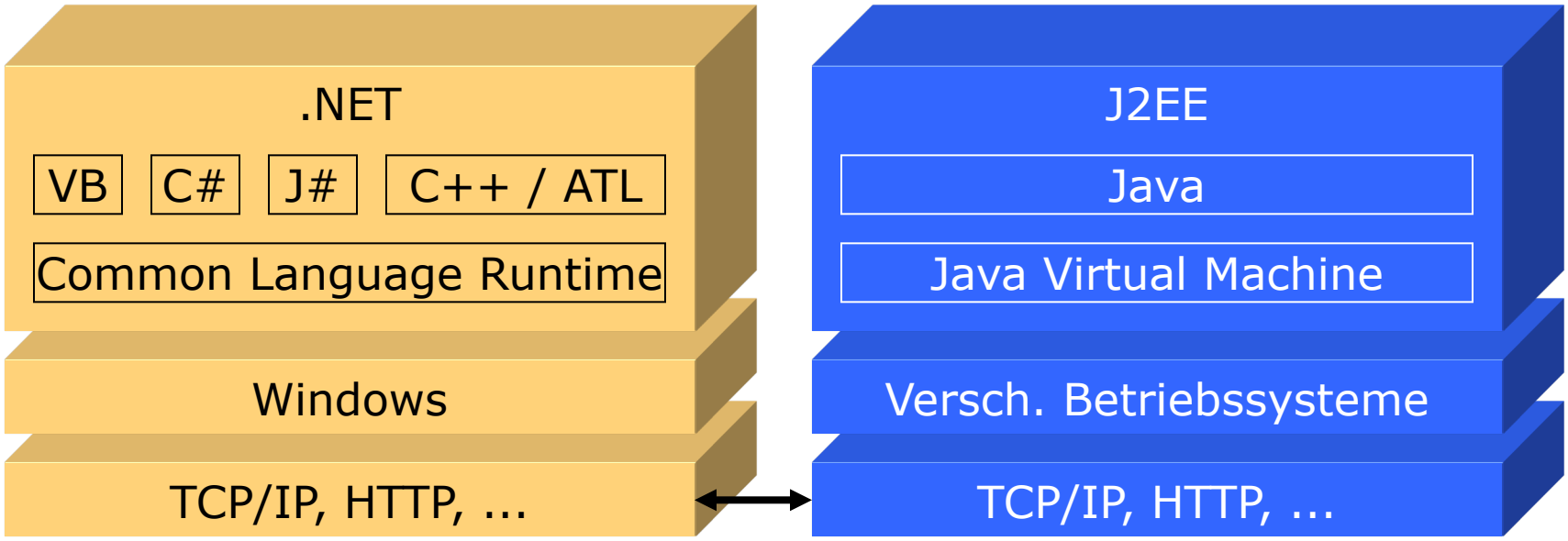
- Notwendige Logik:
  - WSDL-Interpretation*/ WSDL-Erstellung+Publikation
  - SOAP-Nachrichtenempfang und -Versand
  - SOAP-Header-Verarbeitung
  - SOAP-Body (Payload) Parsing und Validierung
  - SOAP-Body-Transformation
  - Anwendungsspezifische Geschäftslogik



*: nur auf Dienstnutzer-Seite

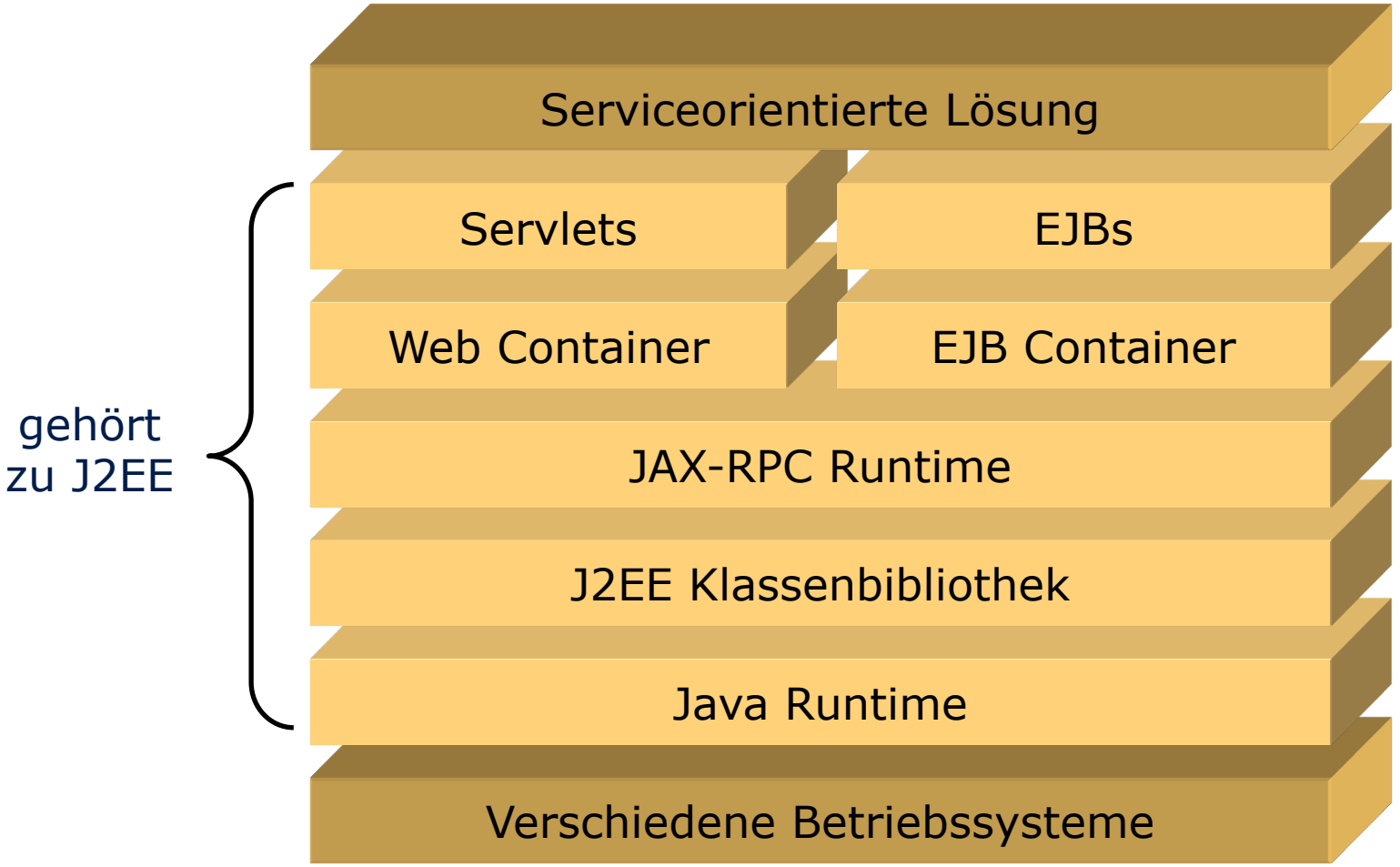
Mögliche Implementierungsarchitekturen

- Zwei häufig verwendete Architekturen:
  - .NET (Microsoft)
  - J2EE (Sun)



- auch Skriptsprachen wie PHP, Perl, Ruby, Python, etc.

J2EE - Java 2 Platform Enterprise Edition



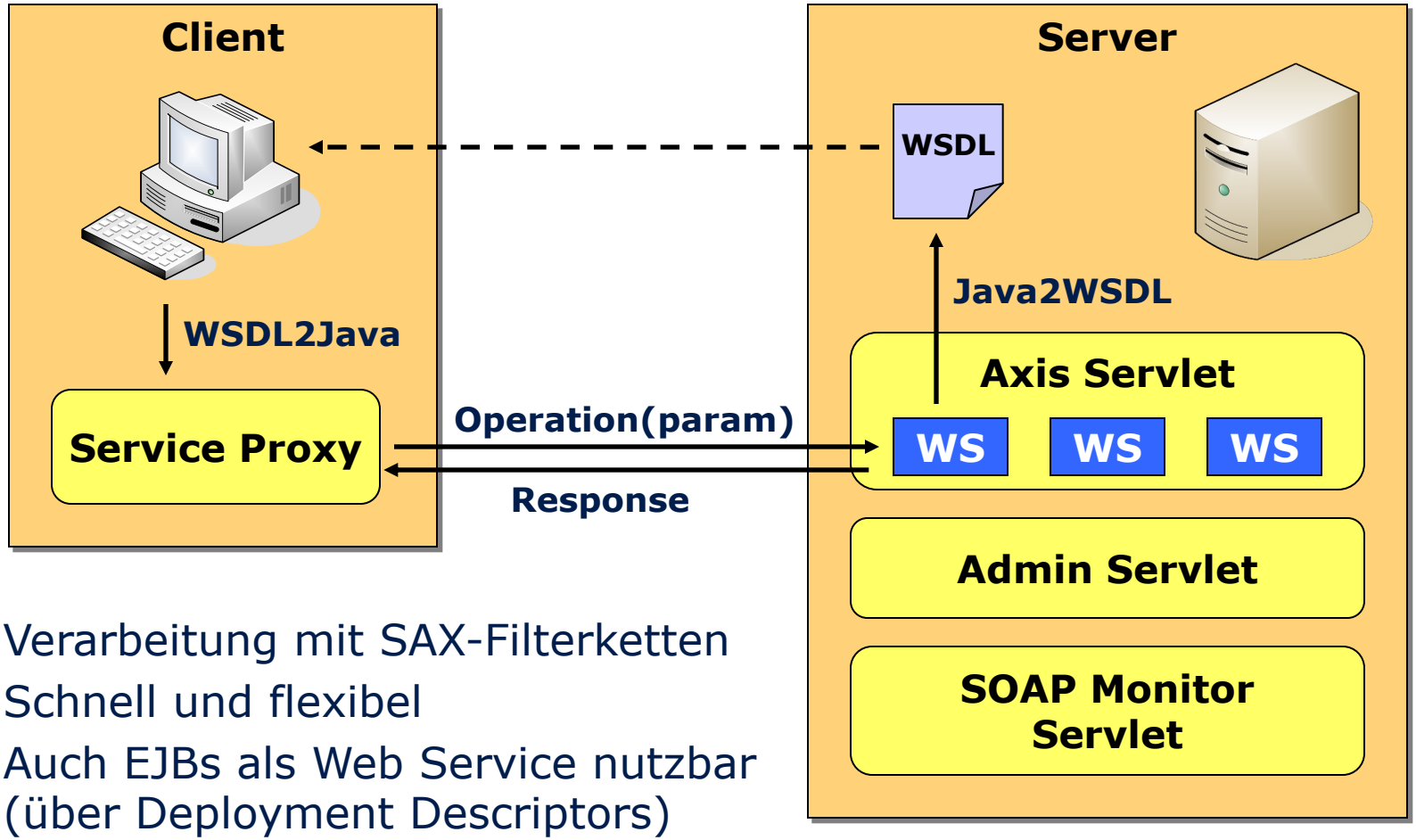
J2EE für SOA

- 2 grundlegende Arten von Dienstanbietern:
  - JAX-RPC Service Endpoints
  - EJB Service Endpoints
- 3 Arten von Dienstnutzer-Proxys:
  - Generated Stub
  - Dynamic Proxy
  - Dynamic Invocation Interface
- J2EE unterstützt wie .NET die Entwicklung von WS-I Basic-Profile-kompatiblen Diensten
  - fördert wichtige SOA-Charakteristiken wie Interoperabilität, offene Standards und Anbietervielfalt
- Bietet eine Vielzahl von APIs

Laufzeitumgebung für Web Services mit Java

- Unterstützung der SOAP-Kommunikation durch Frameworks
- Beispiel: Apache Axis2 (Extensible Interaction System)
  - Open-Source-Implementierung
  - Entstanden aus Apache SOAP, früher SOAP4J (IBM)
  - Entwicklung von Clients, Servern und Gateways möglich
  - Unterstützte Standards: SOAP 1.1/1.2, WSDL 1.1/2.0, SAAJ (SOAP with Attachments API) 1.1
  - Als Java-Servlet in Servlet-Containern einsetzbar
  - Werkzeuge: Java2WSDL und WSDL2Java
- Weitere Frameworks:
  - Sun Java Web Services Developer Pack (WSDP), jetzt Teil von Project Glassfish (Open-Source-Implementierung)
  - webMethods GLUE (kommerziell)
  - Java Spring Framework

Apache Axis: Überblick

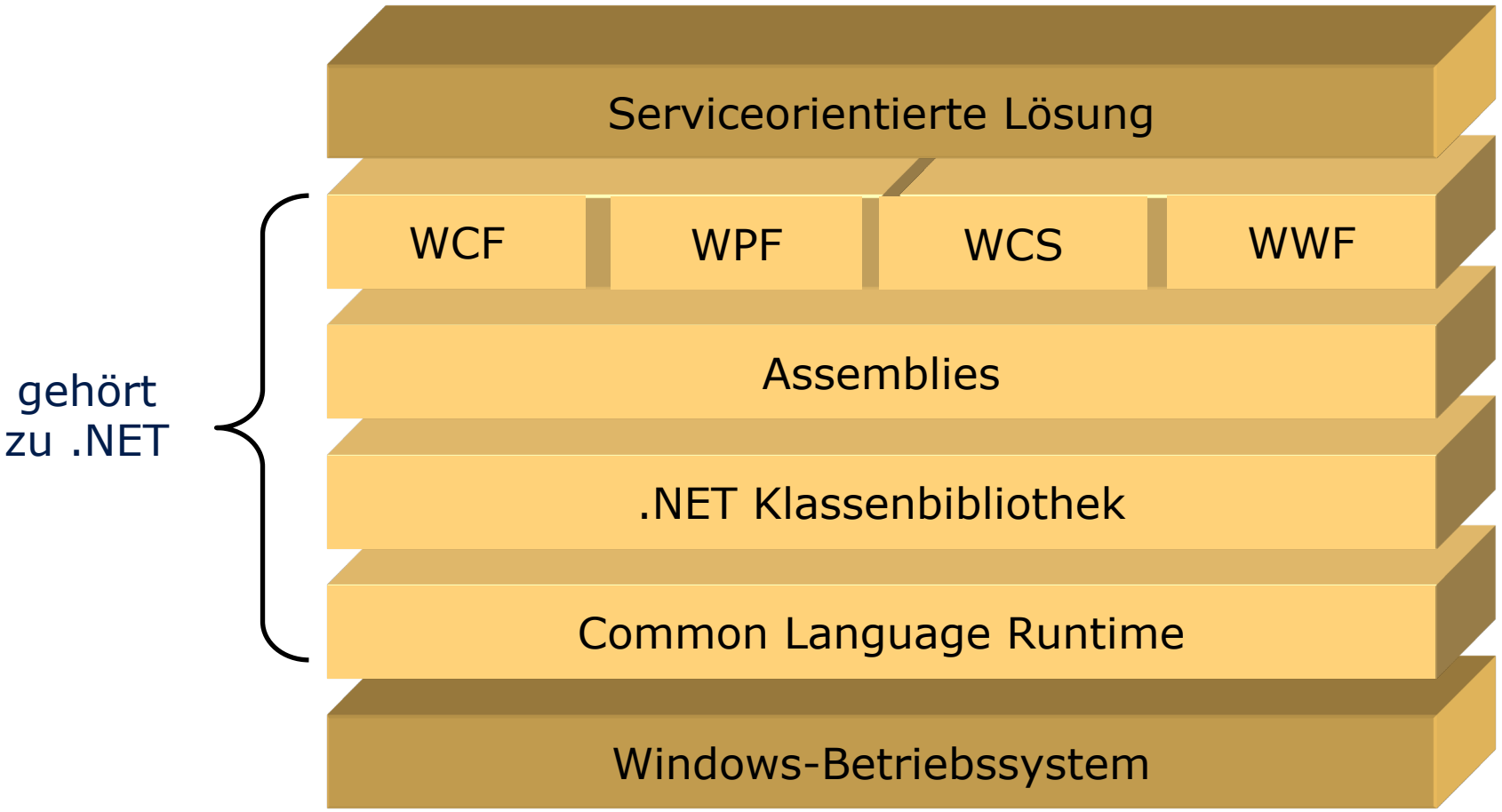


- Verarbeitung mit SAX-Filterketten

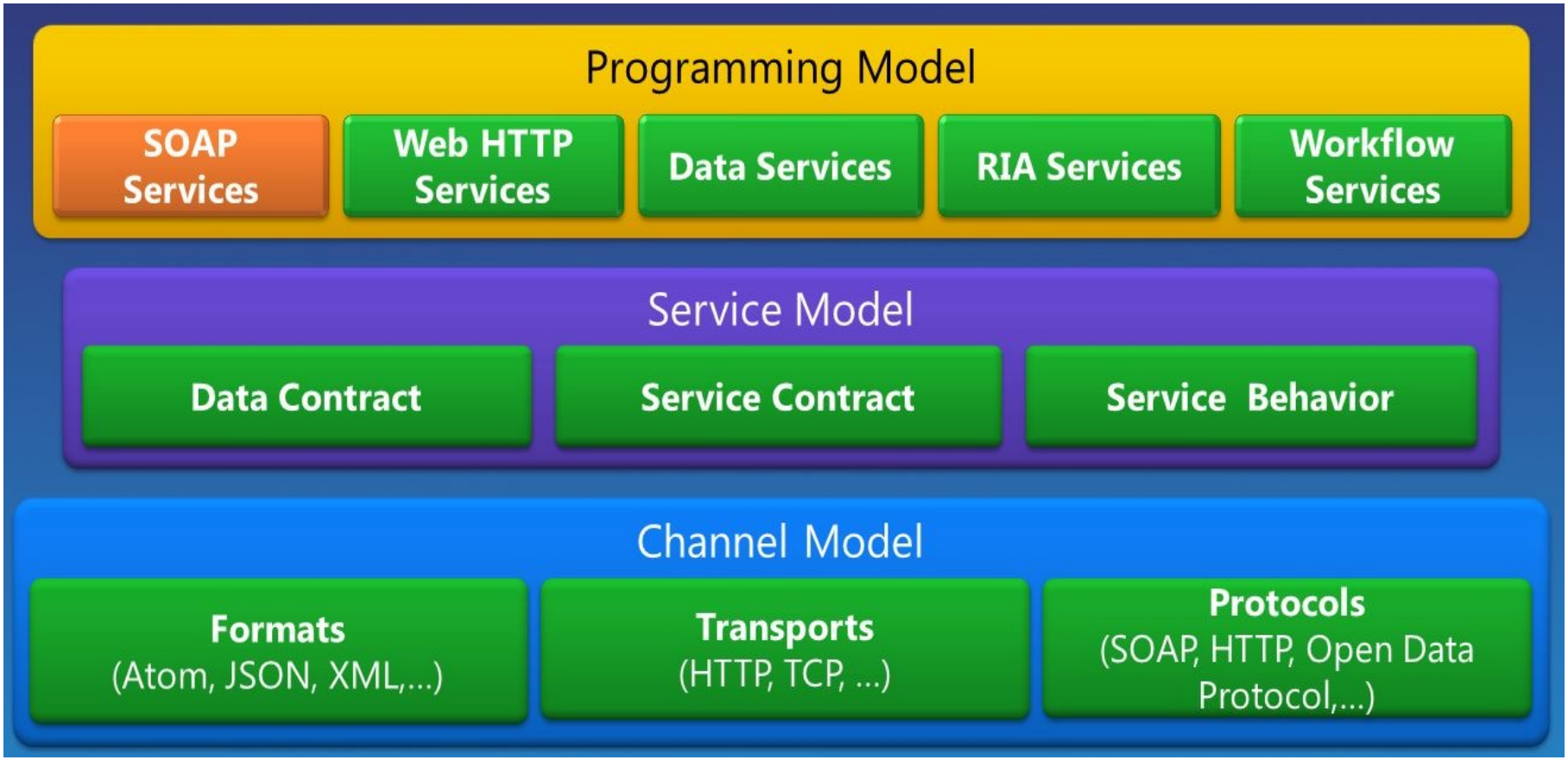
⇒ Schnell und flexibel

- Auch EJBs als Web Service nutzbar (über Deployment Descriptors)

.NET Framework



.NET WCF Architektur



Quelle: MSDN.microsoft.com

.NET WCF Architekturkomponenten

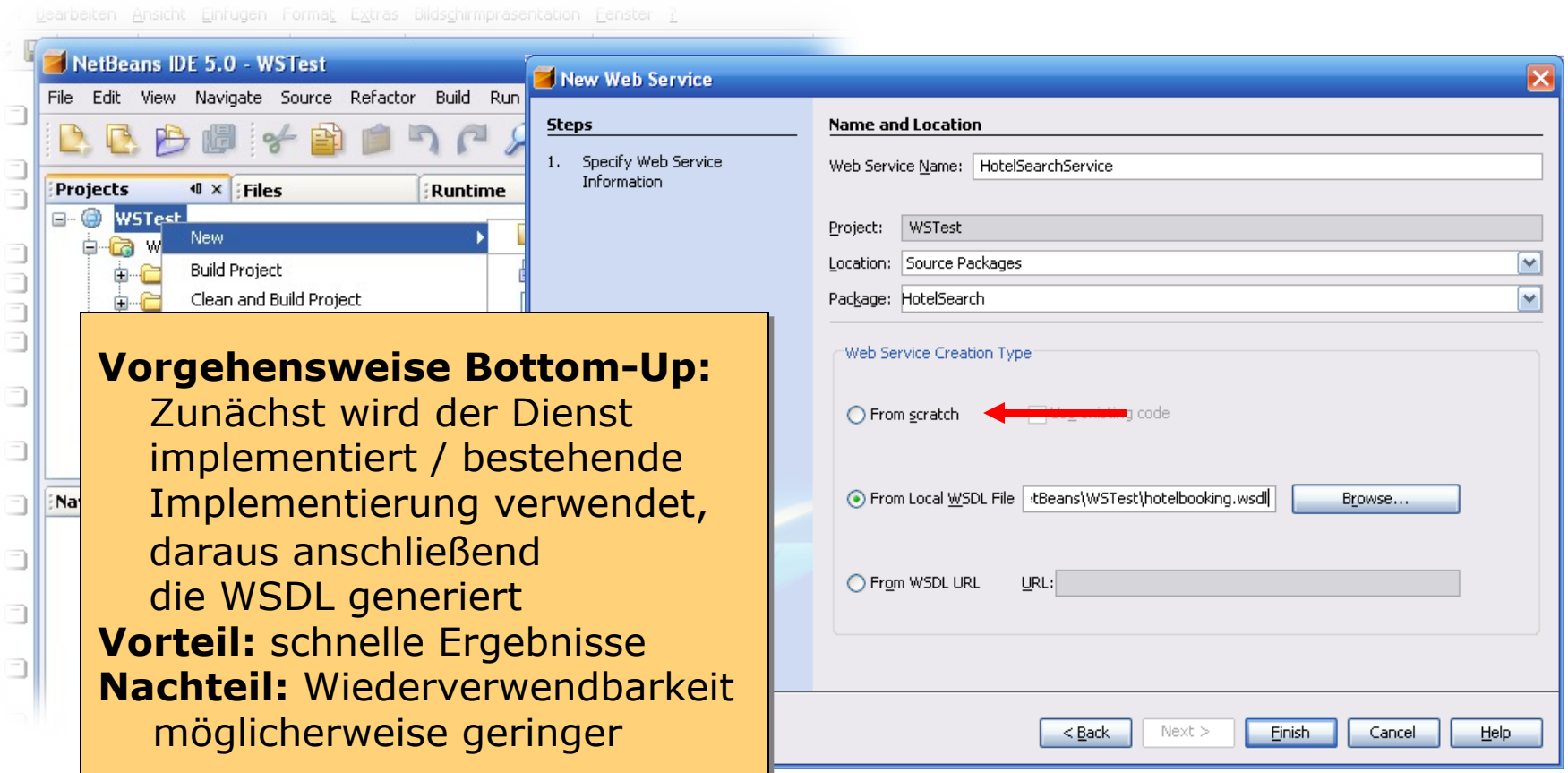
Service Model:

- ***Datenvertrag*** beschreibt alle Parameter, aus denen die einzelnen Nachrichten bestehen (XSD-Dokumente)
- ***Dienstvertrag*** gibt die tatsächlichen Methodensignaturen des Dienstes an und wird als Schnittstelle in einer der unterstützten Programmiersprachen verteilt (z. B. Visual Basic oder Visual C#)
- ***Richtlinien und Bindungen*** legen die zur Kommunikation mit einem Dienst erforderlichen Bedingungen fest (Transportprotokoll, Kodierung, Sicherheitsanforderungen,…)

Channel Model:

- ***Transportkanäle*** lesen und schreiben Nachrichten aus dem Netzwerk (HTTP, Pipes, TCP und MSMQ)
- ***Formats/Codierungen*** (XML oder optimierte Binärdateien)
- ***Protokollkanäle*** implementieren Nachrichtenverarbeitungs-protokolle. (bspw. WS-Security, WS-Reliability)

- Beispiel: Java-Entwicklungsumgebung NetBeans IDE 5.0



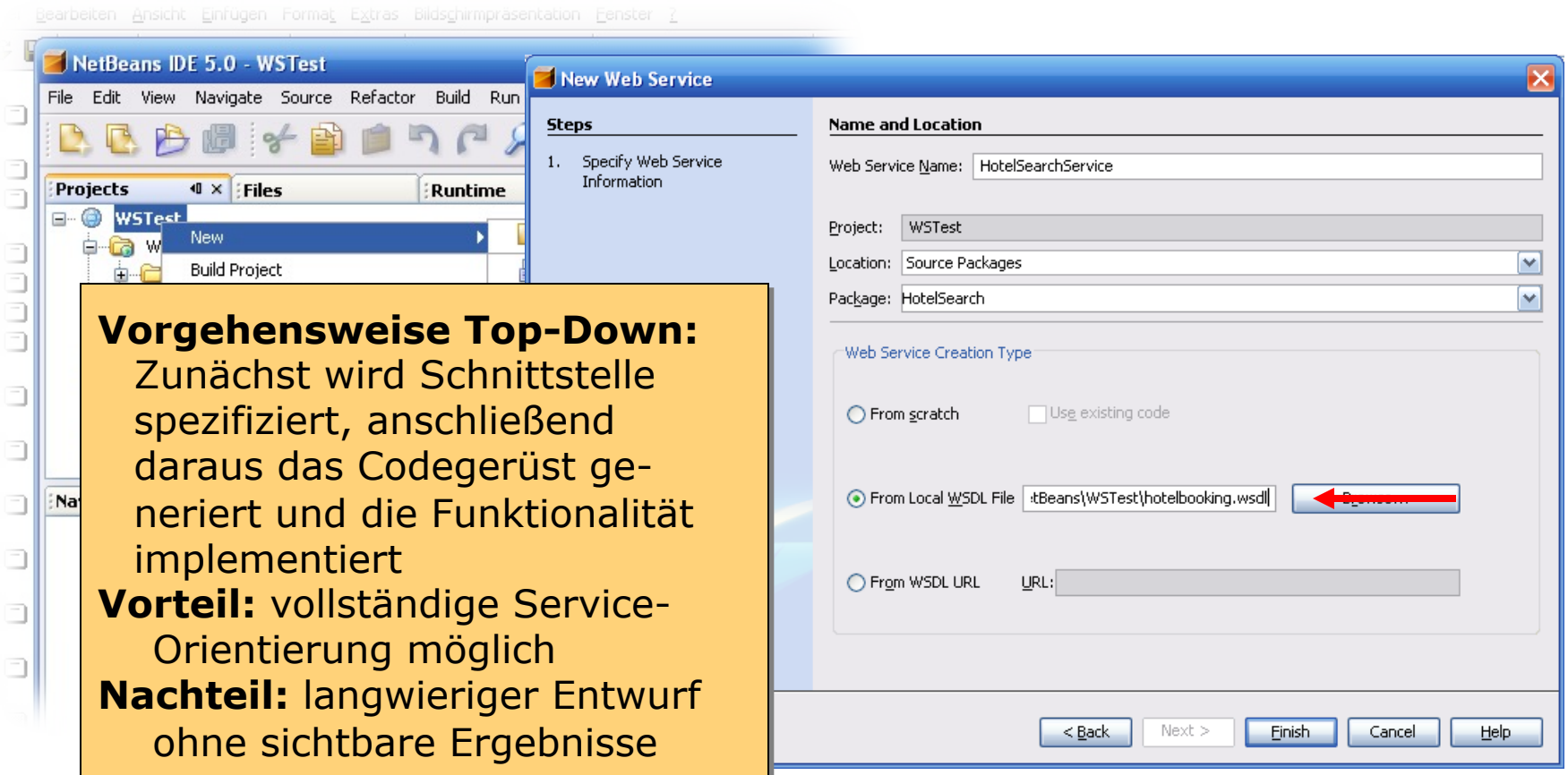
Vorgehensweise Bottom-Up:
Zunächst wird der Dienst implementiert / bestehende Implementierung verwendet, daraus anschließend die WSDL generiert

Vorteil: schnelle Ergebnisse

Nachteil: Wiederverwendbarkeit möglicherweise geringer

Implementierung von Web Services

- Beispiel: Java-Entwicklungsumgebung NetBeans IDE 5.0



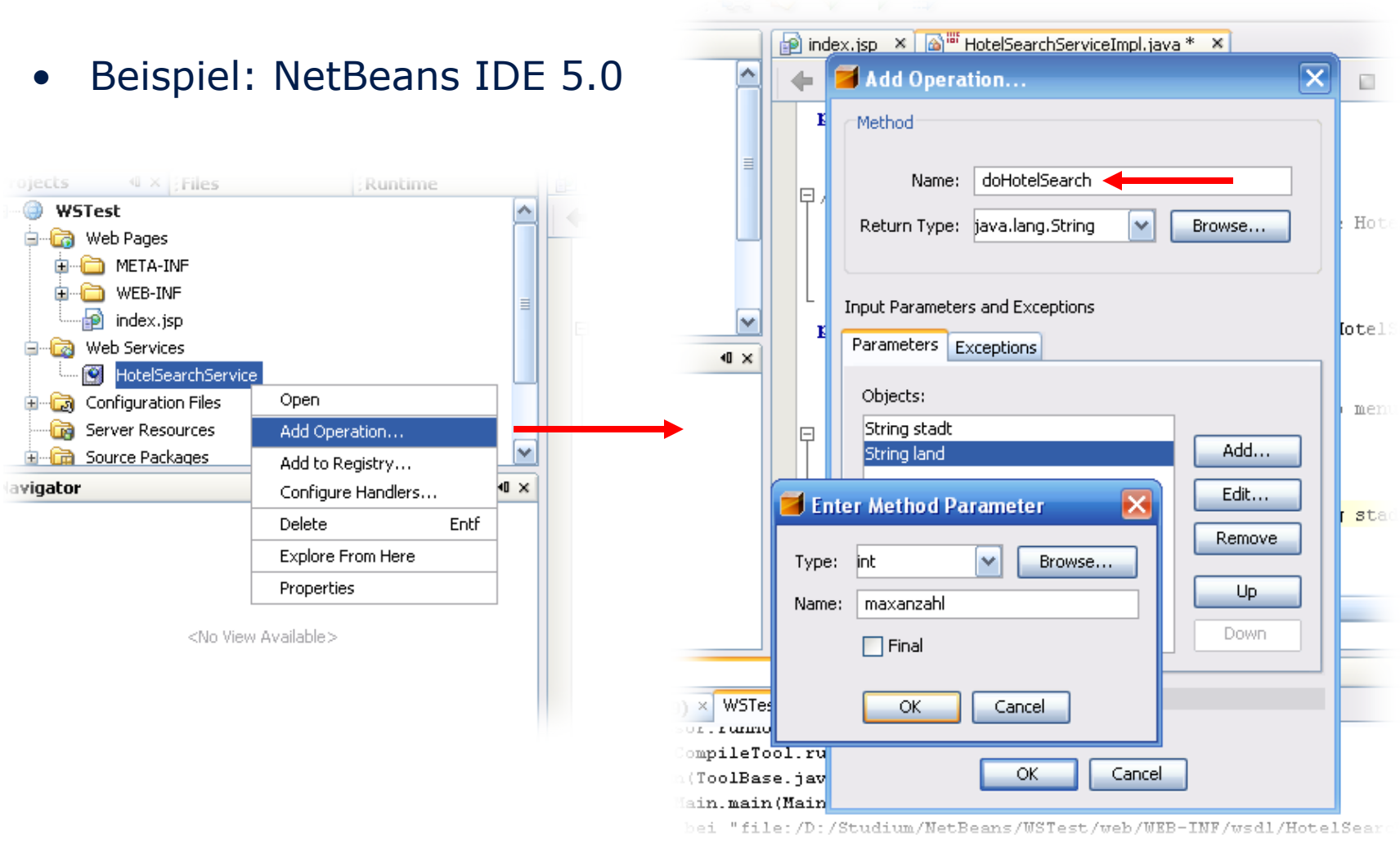
Vorgehensweise Top-Down: Zunächst wird Schnittstelle spezifiziert, anschließend daraus das Codegerüst generiert und die Funktionalität implementiert

Vorteil: vollständige Service-Orientierung möglich

Nachteil: langwieriger Entwurf ohne sichtbare Ergebnisse

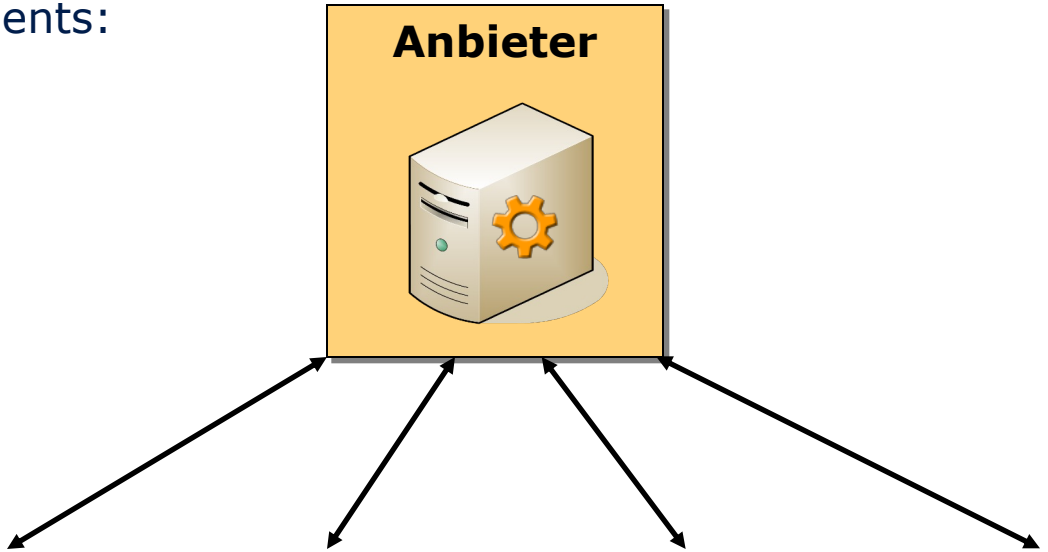
Implementierung von Web Services

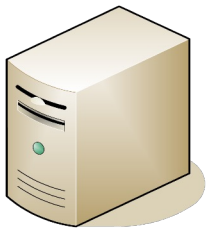
- Beispiel: NetBeans IDE 5.0



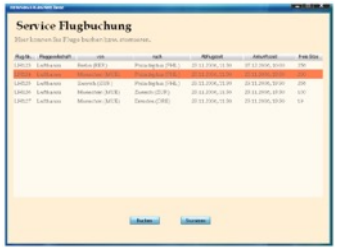
Implementierung eines WS-Clients

Mögliche WS-Clients:



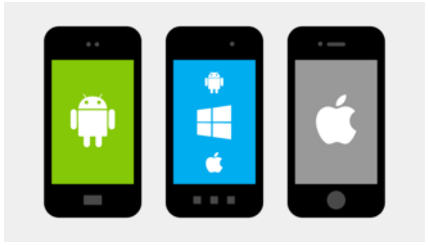


Andere WS oder Komposition



Rich-Client-Anwendung

Web-Client-Anwendung



Mobile Anwendung

Implementierung eines WS-Clients in .NET

- Beispiel: .NET Framework-SDK
- Automatische Generierung eines Platzhalters (*Proxy*) für den entfernten Web Service: Werkzeug wsdl.exe

```
wsdl.exe /namespace:HotelClient /out:HSSProxy.cs
   http://api.holtin.com/hss?WSDL
```

- Verwendung des Proxy in einer C#-Client-Anwendung:

```
using System;
using HotelClient; // Namespace des Proxy
public class NetClient {
  public static void Main(string[] args) {
    HotelSearchService hss = new HotelSearchService();
    Console.WriteLine("Gefundene Hotels: \n"
          + hss.doHotelSearch("Dresden", "", 1);
  } }
```

Clienterstellung mittels Scriptsprachen

- Vorteil: Dynamische Stub-Generierung zur Laufzeit
- Beispiel: Ruby on Rails
  - Web Service Unterstützung mit SOAP4R Package (in Ruby enthalten)

```
require 'soap/wsdlDriver'
wsdl = 'http://141.76.40.55/services/FlugBuchung?wsdl'
driver = SOAP::WSDLDriverFactory.new(wsdl).create_rpc_driver
driver. ({})["Fluege"]["Flug"].each do |flug|
puts flug.flugNR end
```

- Beispiel: Python

```
imendpoint PySOAP, Webservice
ws = WebService.ServiceProxy(
"http://141.76.40.55/services/FlugBuchung.wsdl" )
Fluege = ws.alleFluegeAnzeigen()
```

Zusammenfassung

- WSDL-Dienstbeschreibung spezifiziert Schnittstelle und Ort der Dienstimplementierung der Hotelkette
- Reisebüro verwendet WSDL zur Generierung einer Proxy-Klasse, die wie ein lokales Objekt verwendet werden kann
- Datenaustausch erfolgt je nach Binding-Definition in der WSDL z.B. über SOAP-Nachrichten

Reisebüro

z.B. .NET-Webanwendung

SOAP-Anfrage

SOAP-Antwort

BEIGENSTERGER HOTEL

Hotelkette

z.B. J2EE-Anwendung auf Application Server

WSDL

Dienstbeschreibung

Quellen + Tutorials

[JSR311] Java Specification Request 311: JAX-RS, https://jsr311.dev.java.net/

[NBR08] NetBeans REST Tutorial, http://www.netbeans.org/kb/60/websvc/rest.html

[RR07] L. Richardson and S. Ruby. RESTful Web Services. O'Reilly Media, Inc., May 2007

Weitere Tutorials:

Web Service mit Microsoft .NET

https://www.tutorialspoint.com/webservices

Web Service Tutorial - Erstellung eines Gästebuchs mit WS

http://www.se.uni-hannover.de/pages/de:tutorials_webservice_guestbook_jax_ws

Java2Blog - Create RESTful web services in Java

https://www.java2blog.com/create-restful-web-servicesjax-rs-using/

Video Tutorials

Englisch:

Java EE Tutorial #8 - SOAP Web Services with Jax-WS
https://www.youtube.com/watch?v=_B3rAz8Ge58

Java EE Tutorial #18 - RESTful Web Services with Jax-RS
https://www.youtube.com/watch?v=dmMdLW92hBA

Simple Ruby on Rails 5 REST API From Scratch
https://www.youtube.com/watch?v=QojnRc7SS9o

Deutsch:

Web Services mit Eclipse Java (Hochschule für Technik Stuttgart)
https://www.youtube.com/watch?v=oDyF7VpFxyY

Web Services - Rest Bottom-Up Beispiel (Hochschule für Technik Stuttgart)
https://www.youtube.com/watch?v=Dfo9N_3-6yw

REST WCF Service in 120 Sekunden
https://www.youtube.com/watch?v=1LBLyYUvRsI

Wissensüberprüfung: AMCS

Web-Zugang:

https://amcs.website

PIN:

SCC2023

Vorlesung 2 - Implementierung von Services und Clients