

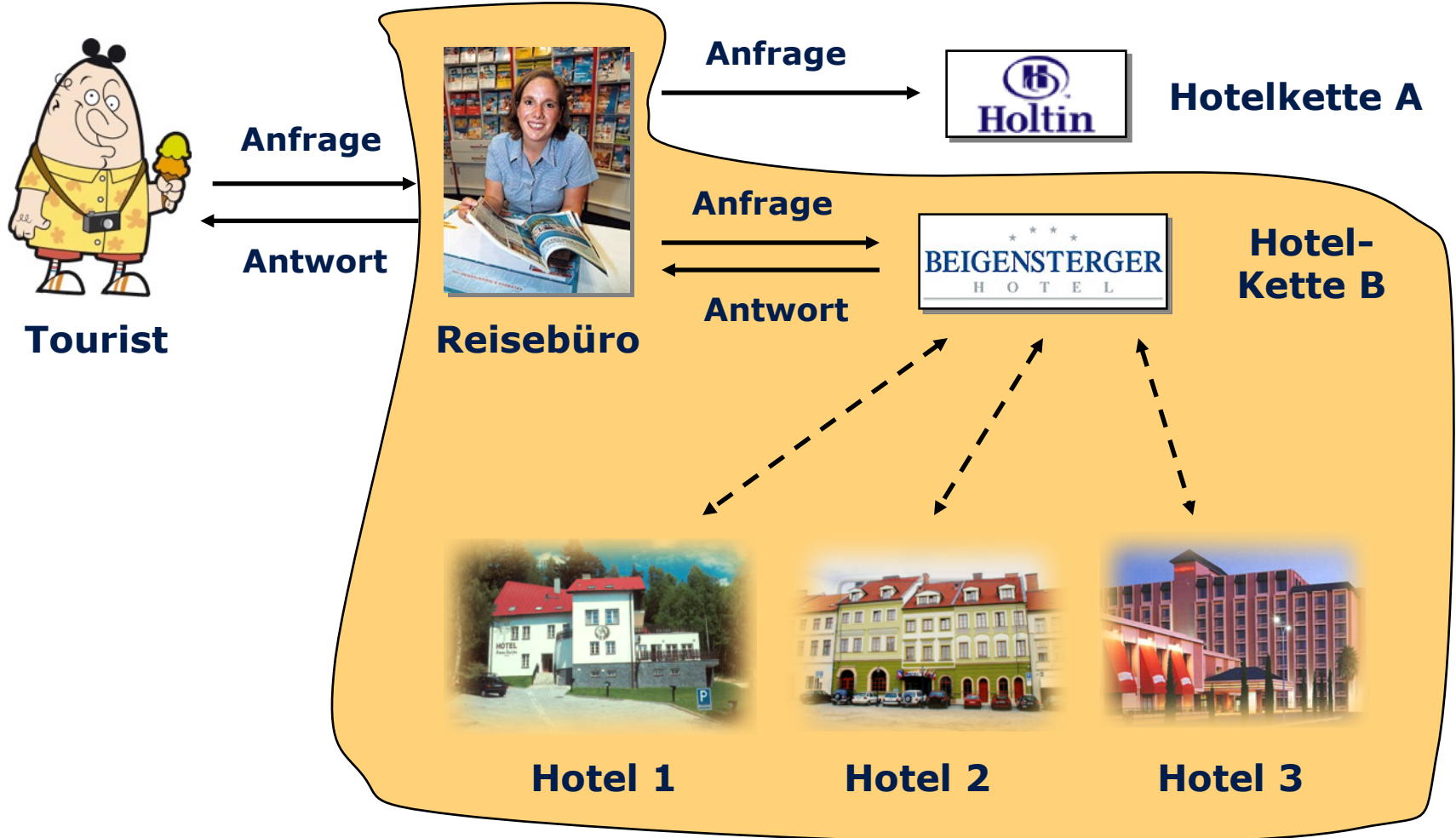


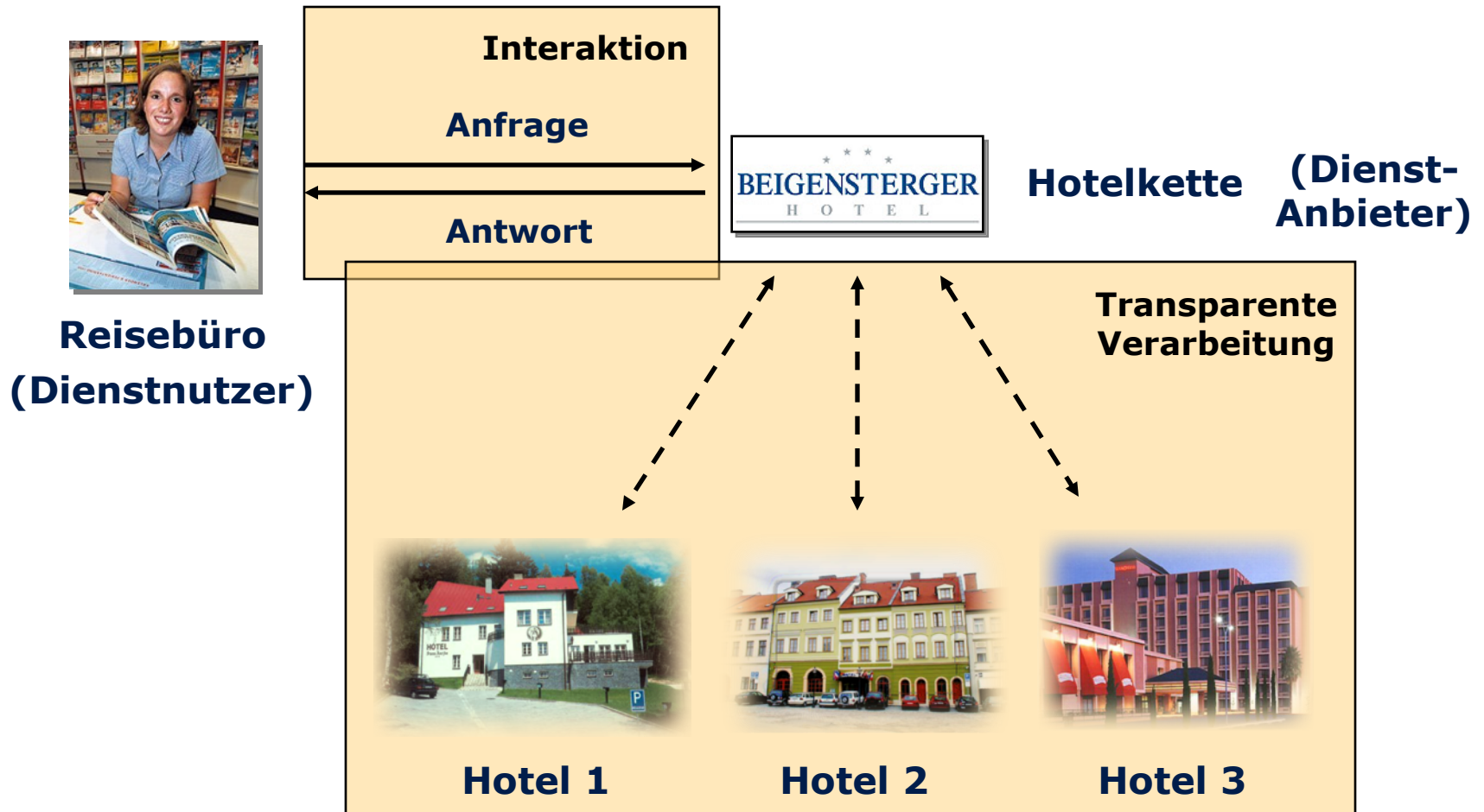
Vorlesung
„Service and Cloud Computing“

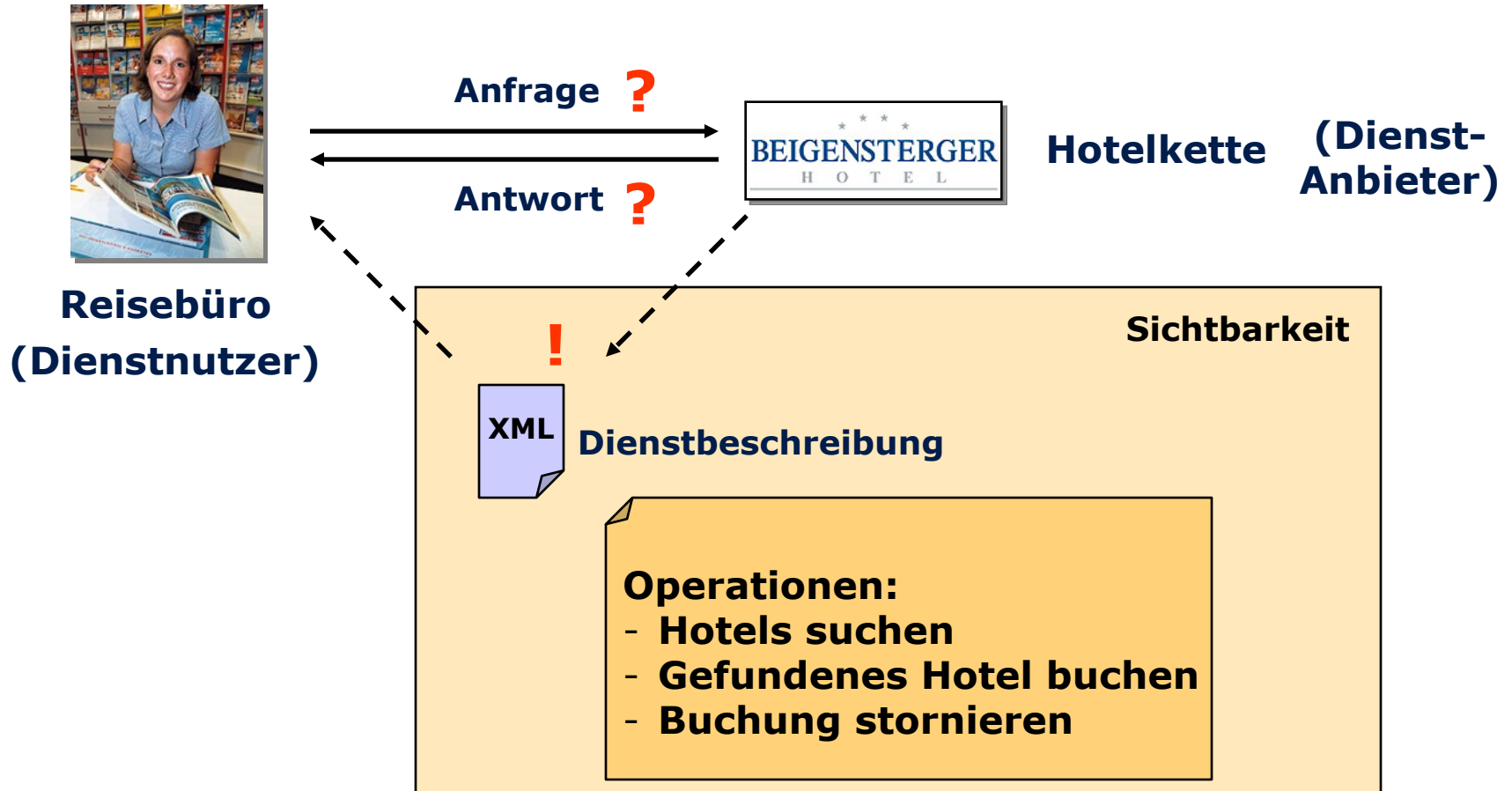
2. Implementierung von Services
und Clients

Dr.-Ing. Iris Braun

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







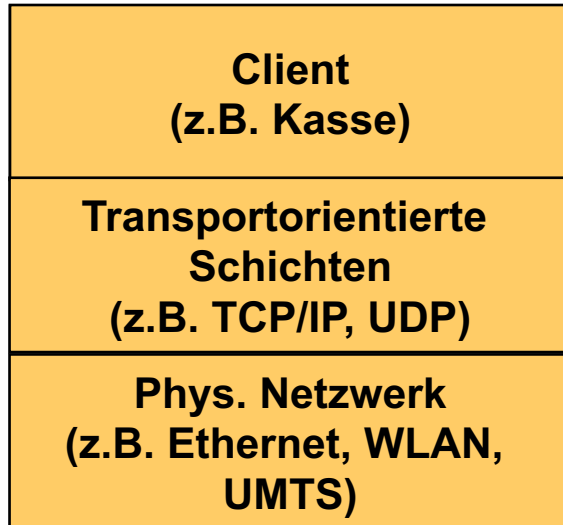
- **Anforderungen:**

- **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

Aufrufender (Nutzer)



Anwendungs-
interaktion



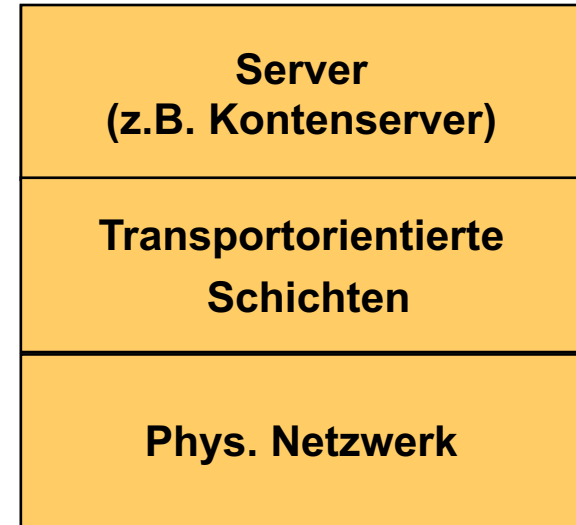
Kommunikations-
protokoll



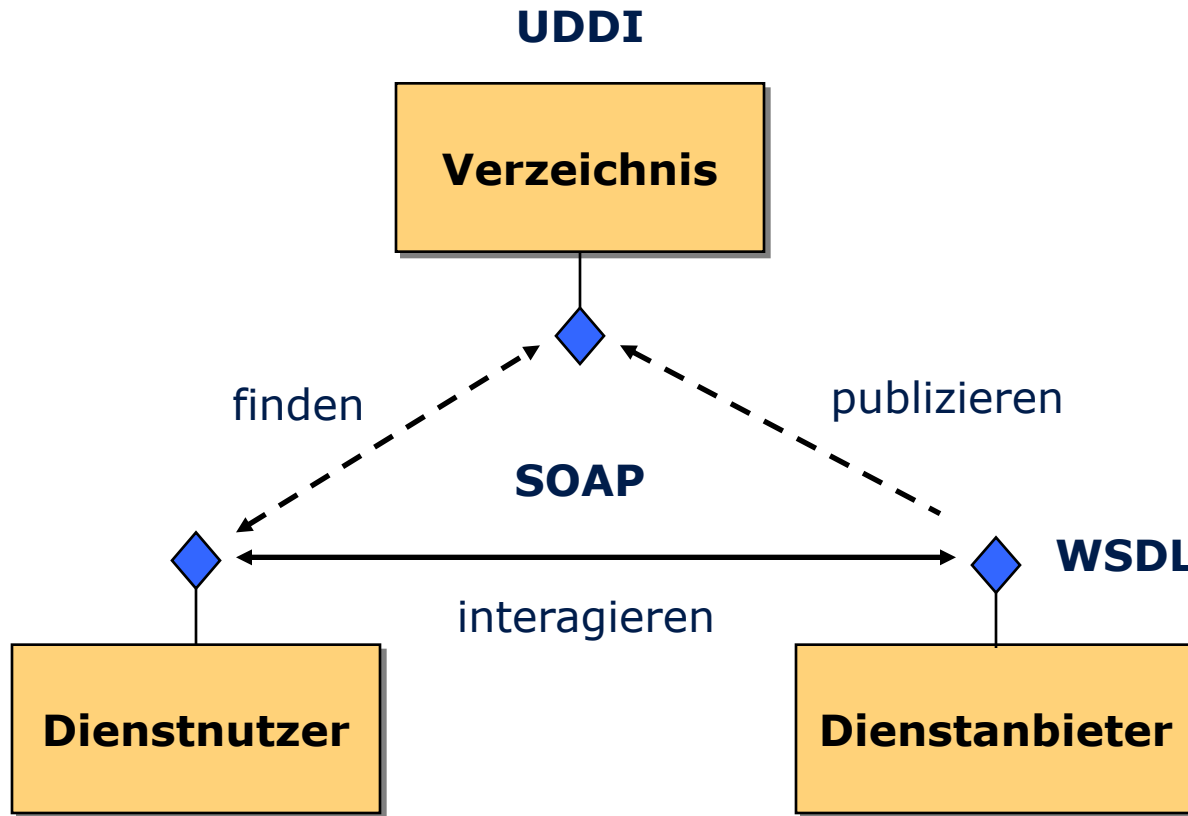
Daten-
übertragung



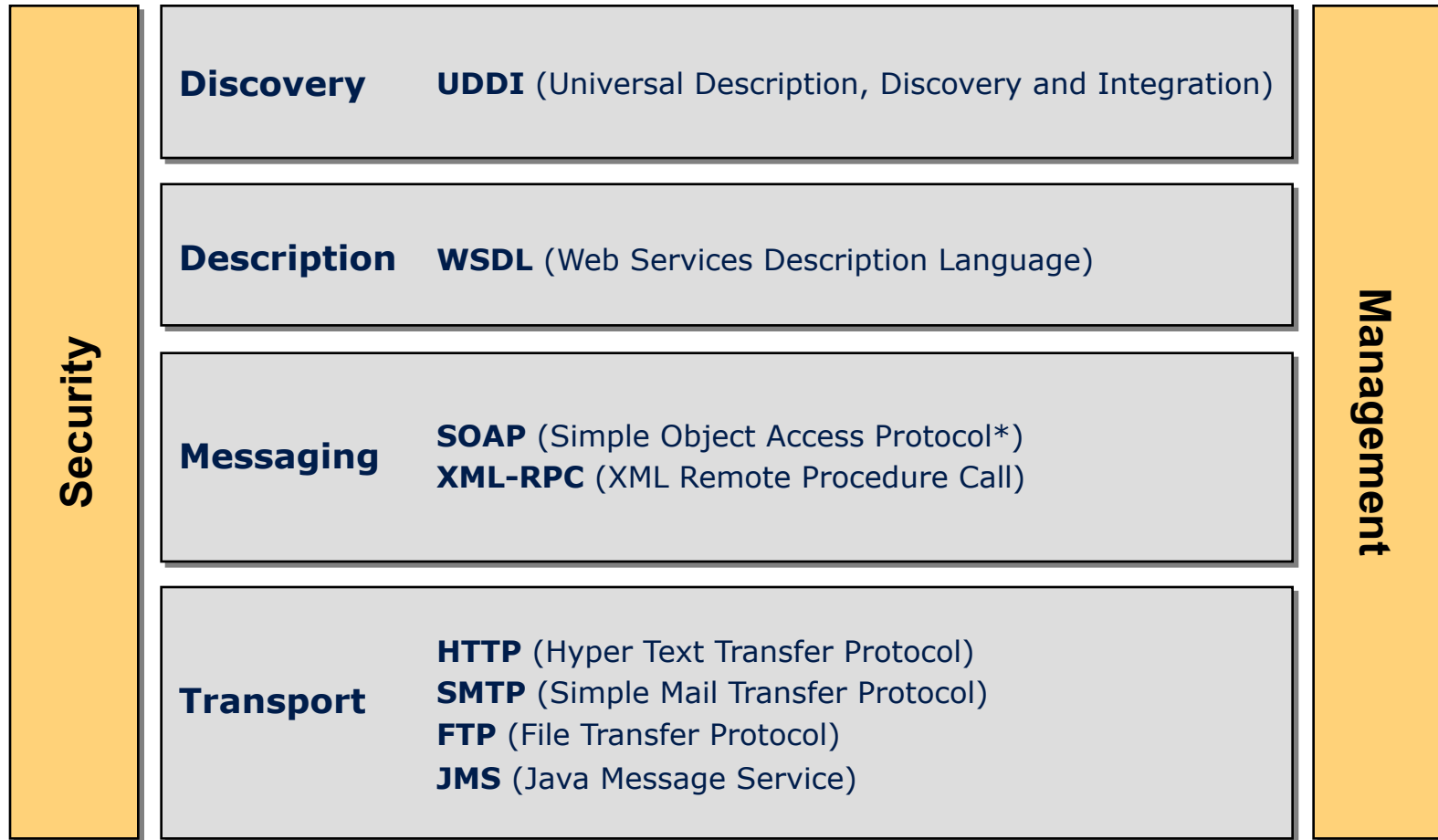
Aufgerufener (Anbieter)



- 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.”

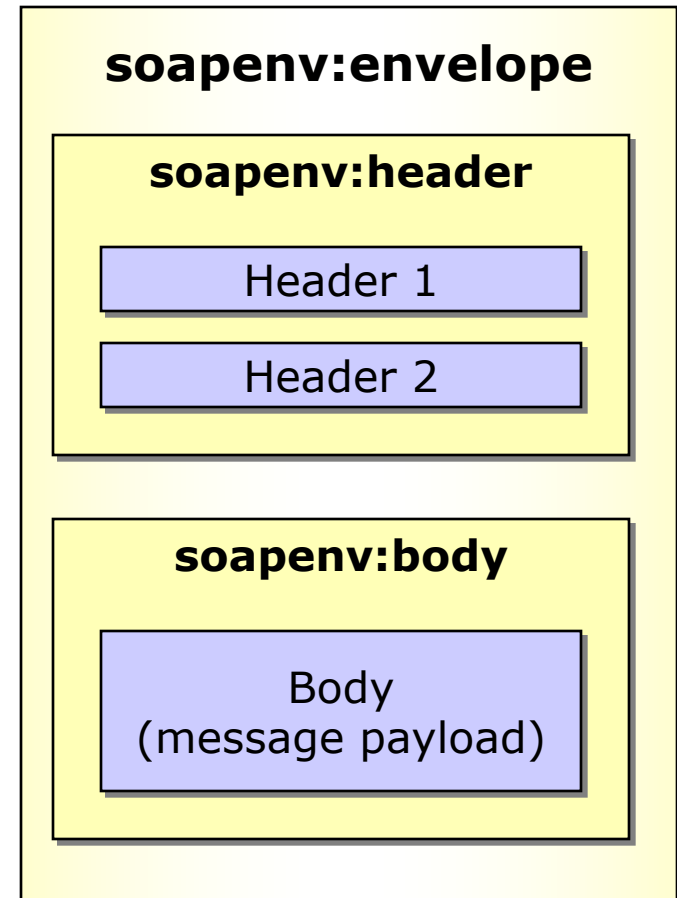


◆ : Dienstschnittstellenbeschreibung



*: ursprüngliche Abkürzung, die nicht mehr verwendet wird

- 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: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:Envelope xmlns:soap=...>
  <soap:Header>
    <m:land xmlns:m="..." soap:mustUnderstand="0"
      soap:actor="next">Deutschland</m:land>
  </soap:Header>
  <soap:Body soap:mustUnderstand="1">
    <tns:doHotel xmlns:tns="http://www.dresden.de/schemas/soap"
      <m:stadt xmlns:m="http://www.dresden.de/schemas/soap"
        </tns:doHotel>
  </soap:Body>
</soap:Envelope>
```

Attribute für Header-Blocks:

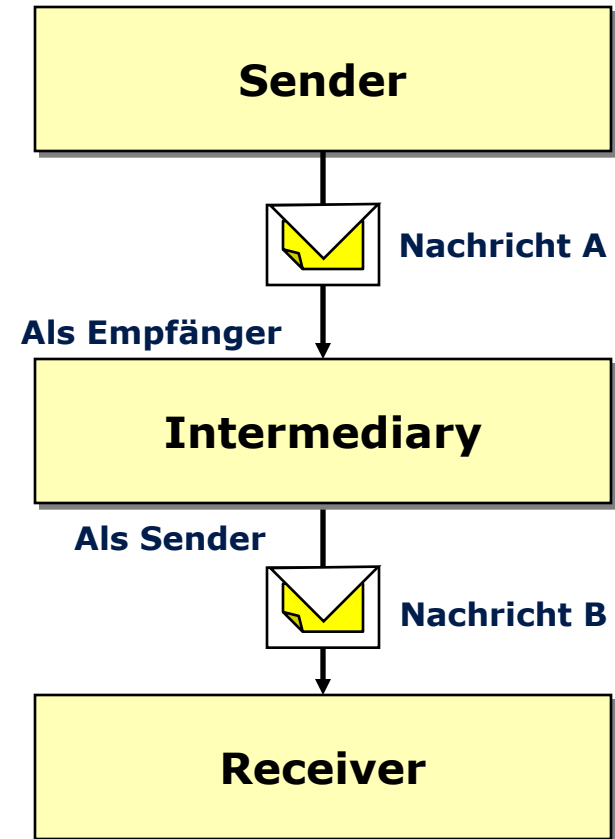
- **mustUnderstand="1"**: jeder Empfänger muss Element verstehen, sonst Fehler senden und Bearbeitung abbrechen
- **actor**: URI, für welchen Empfänger oder eine Zwischenstation (Firewall, Load Balancer) das Header-Element bestimmt ist, vordef.: <http://schema.xmlsoap.org/soap/actor/next> für erste Station, die diese Nachricht verarb.

```
<soap:Envelope xmlns:soap=...>
  <soap:Body soap:encodingStyle=...>
    <soap:Fault>
      <faultcode>soap:Server</faultcode>
      <faultstring>Deutschland ist als Land nicht
        vorhanden.</faultstring>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

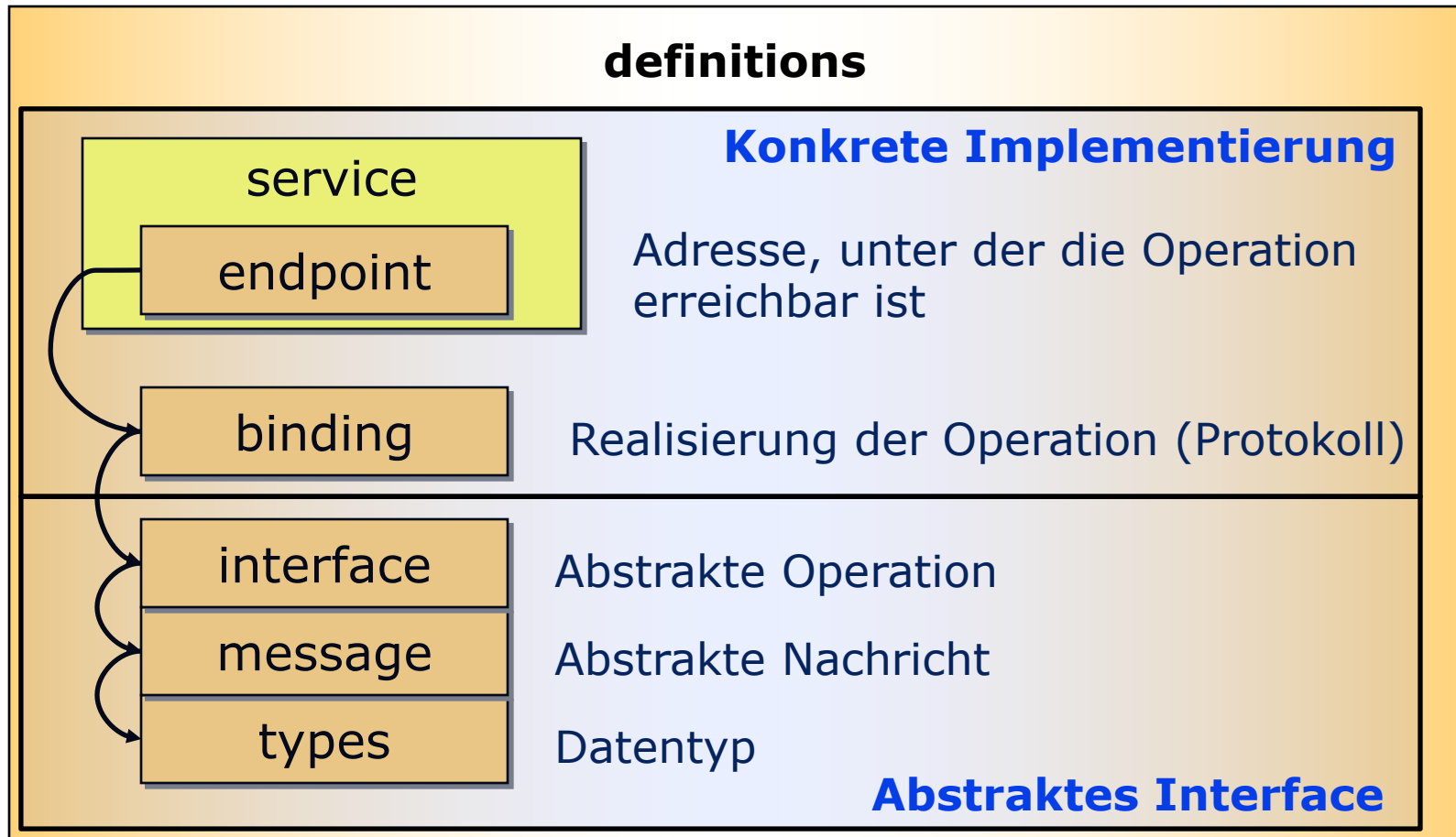
Bei Verarbeitungsfehler:

- Statt Body-Inhalt wird Fault-Element gesendet
- **faultcode**: entweder **soap:Client** oder **soap:Server**
- Weitere Unterelemente:
 - **faultString** (Beschreibung)
 - **faultActor** (Verantwortliche Instanz)
 - **details** (Programmspezifische Fehlermeld., z.B. Zeilennummer)

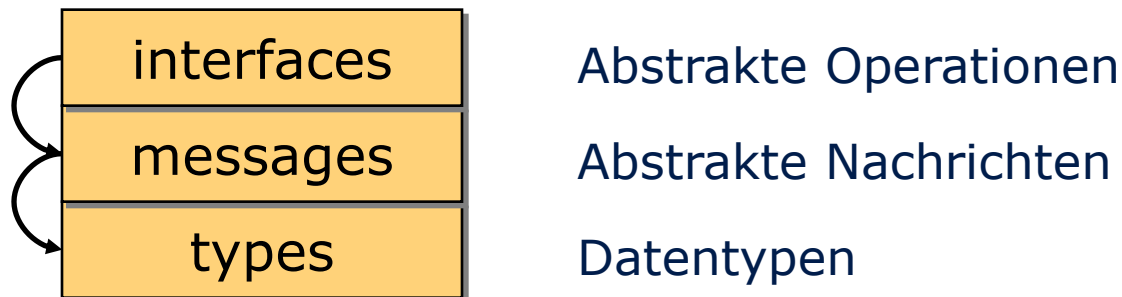
- Teilnehmer eines SOAP-Austauschs:
 - Sender
 - Intermediary (Vermittler)
 - Receiver (Empfänger)
- Vermittler erscheint je nach Sicht als Empfänger oder als Sender
- Wenn $A \neq B$: Active Intermediary
 - Vermittler verändert Header oder Body der SOAP-Nachricht (z.B. Verschlüsselung, Re-Routing zum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



- **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



- 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**



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

Abstrakter Teil

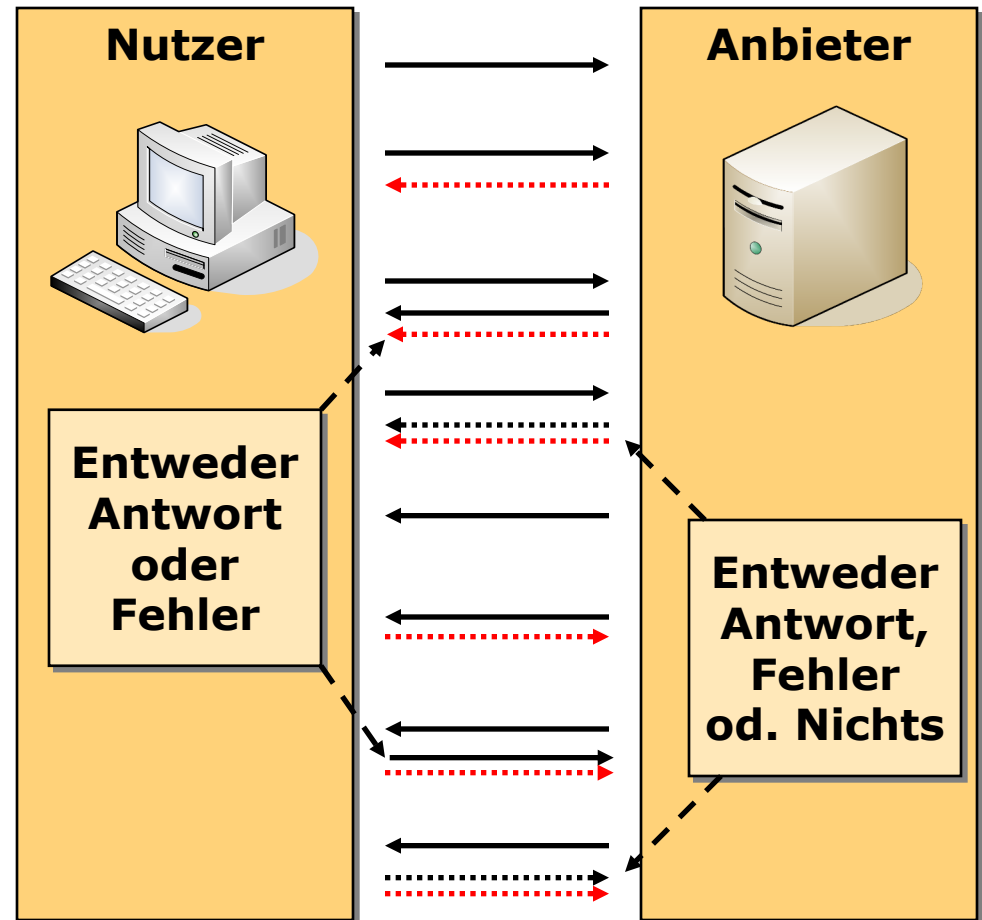
```
<wsdl:documentation ... />  
<wsdl:types> Schema Imports </wsdl:types>  
<wsdl:message> Nachrichten </wsdl:message>  
<wsdl:interface> Operationen </wsdl:Interface>
```

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

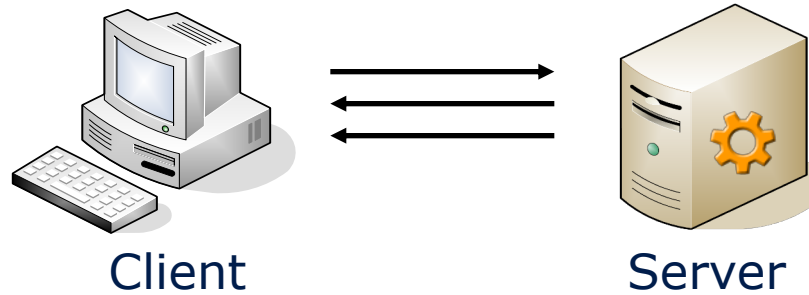
Konkreter Teil

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

- 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



⇒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!

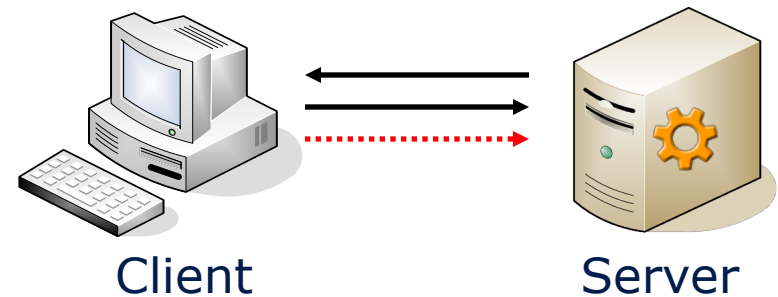
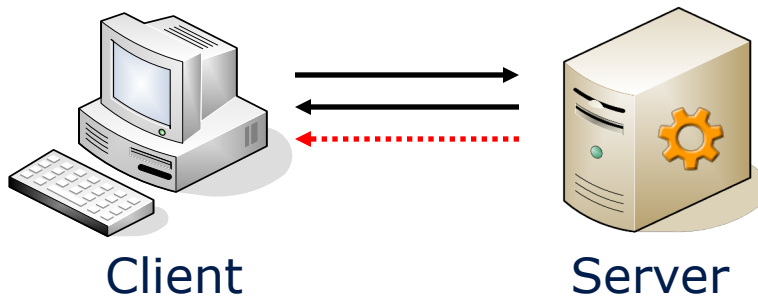
- Statt weiterer Antwort kann auch Fehler gemeldet werden:

Anfrage-Antwort

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

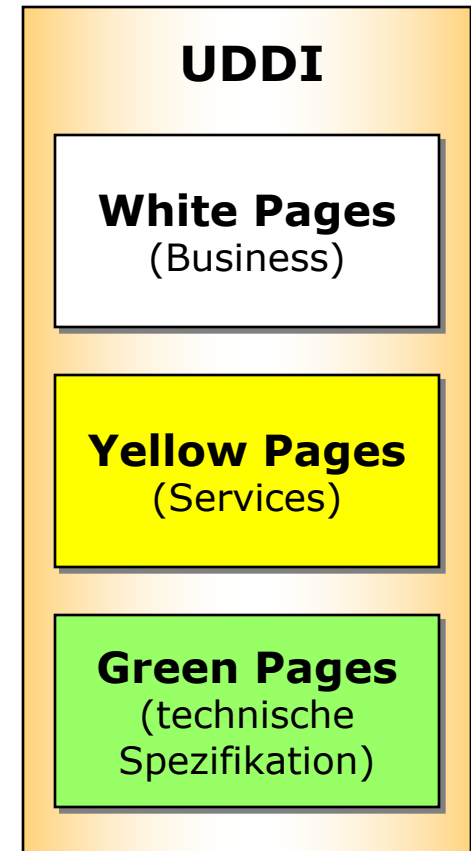
Benachrichtigung-Antwort

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

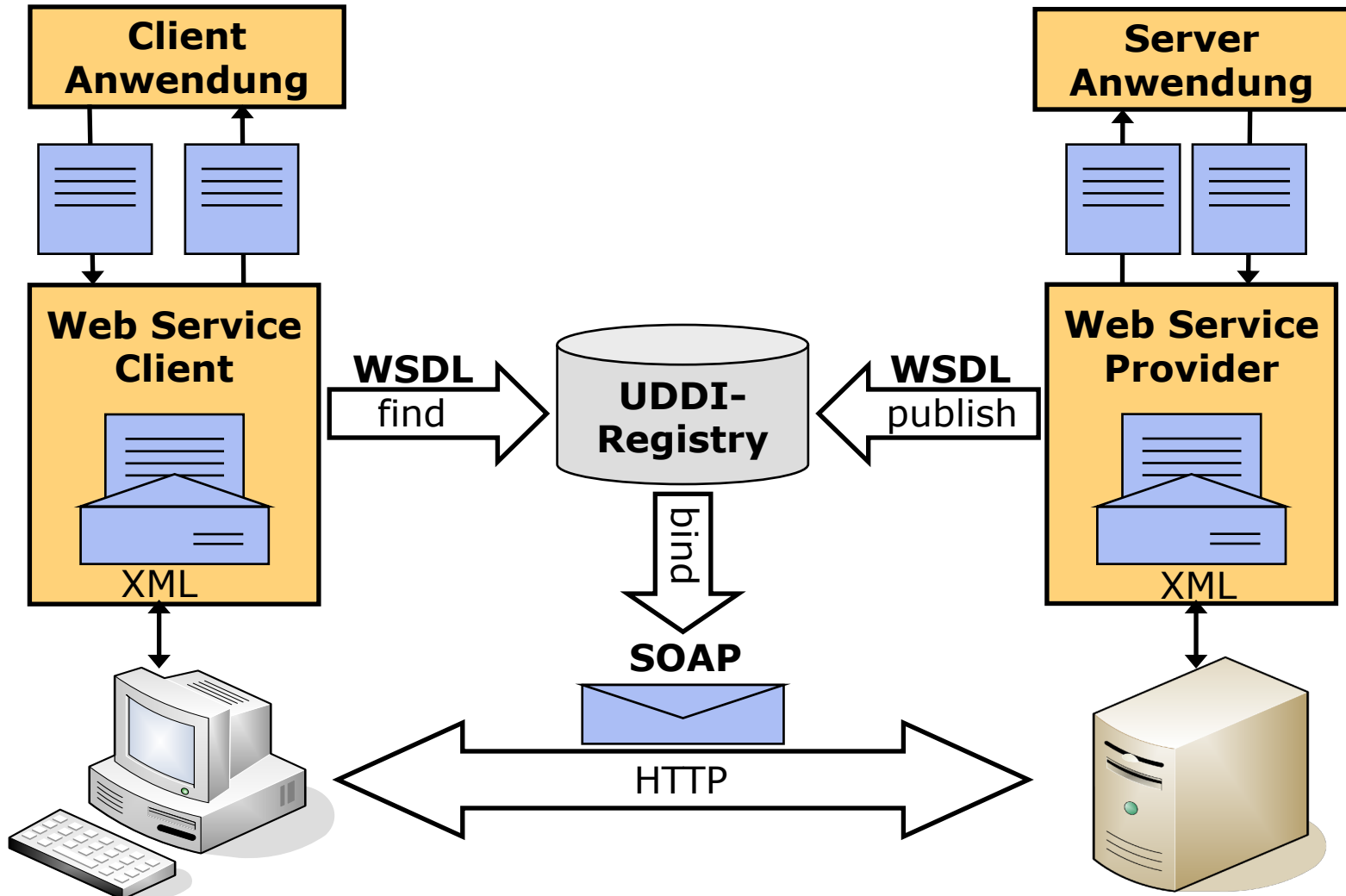


- Universal Description, Discovery and Integration
- OASIS-Standard, aktuell Version 3.0
- Ziel: Standardisierte Methode zum Veröffentlichen und Auffinden von Web Services
- 3 Hauptkategorien:
 - **White Pages:** Namensregister, Kontakt zum Anbieter, mehrsprachige Unternehmensvorstellung, Klassifizierungscode (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

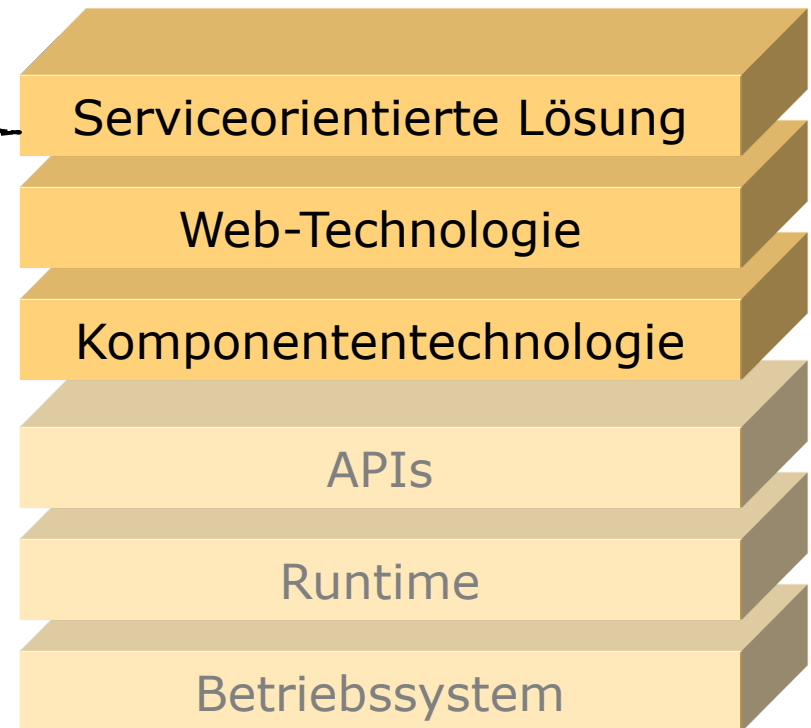
UNSPSC: United Nations Standard Products and Services Code



- 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)

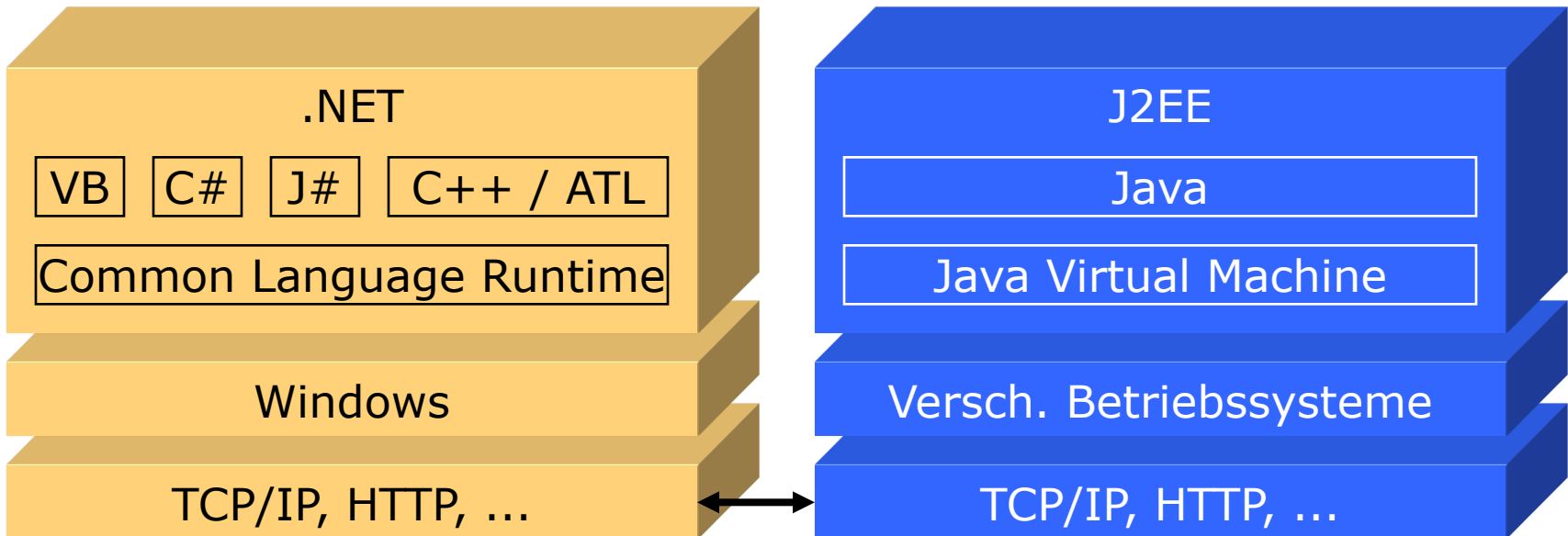


- Notwendige Logik:
 - WSDL-Interpretation*/
WSDL-Erstellung+Publikation
 - SOAP-Nachrichteneingang
und -Versand
 - SOAP-Header-Verarbeitung
 - SOAP-Body (Payload) Parsing
und Validierung
 - SOAP-Body-Transformation
 - Anwendungsspezifische
Geschäftslogik

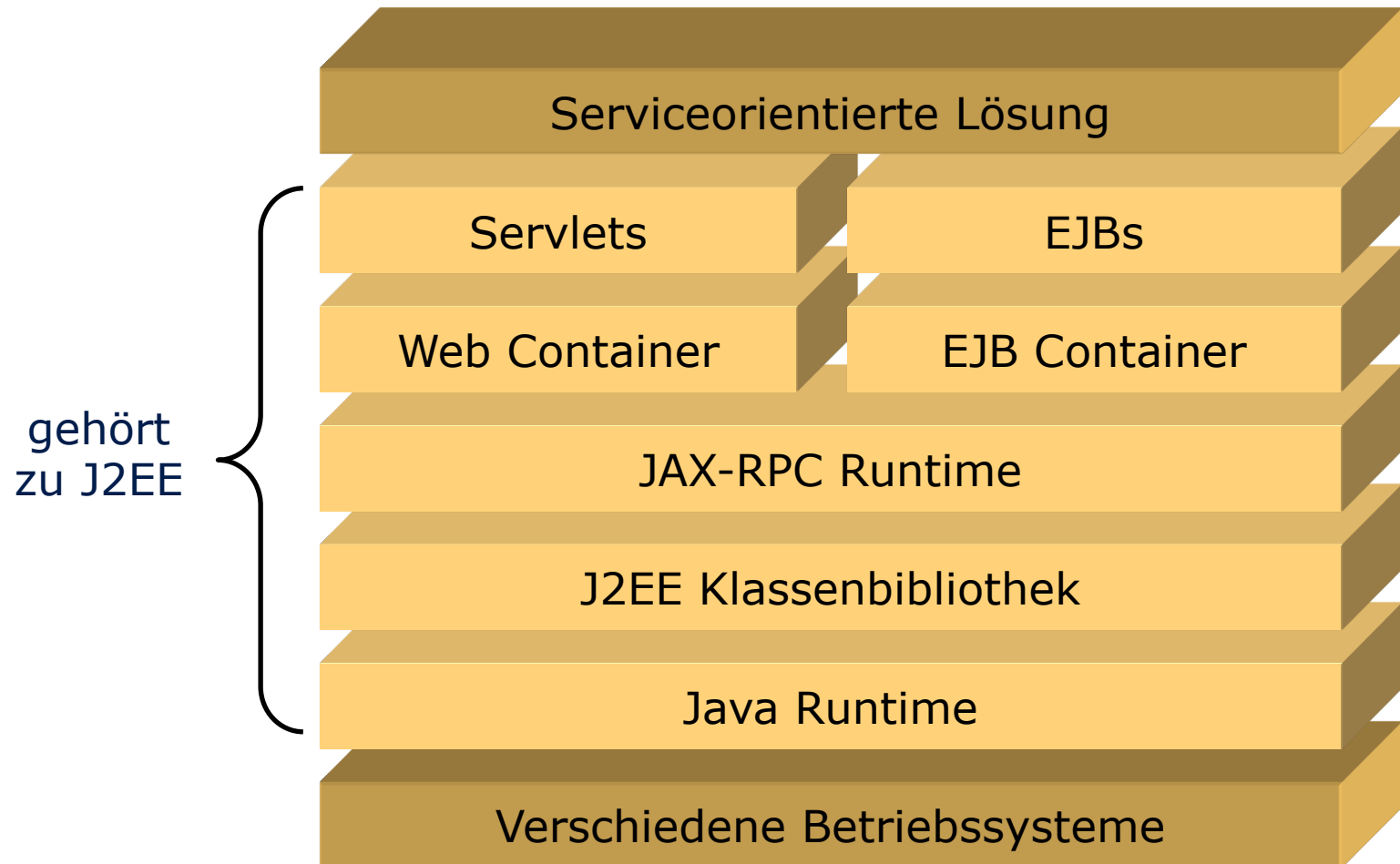


*: nur auf Dienstnutzer-Seite

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



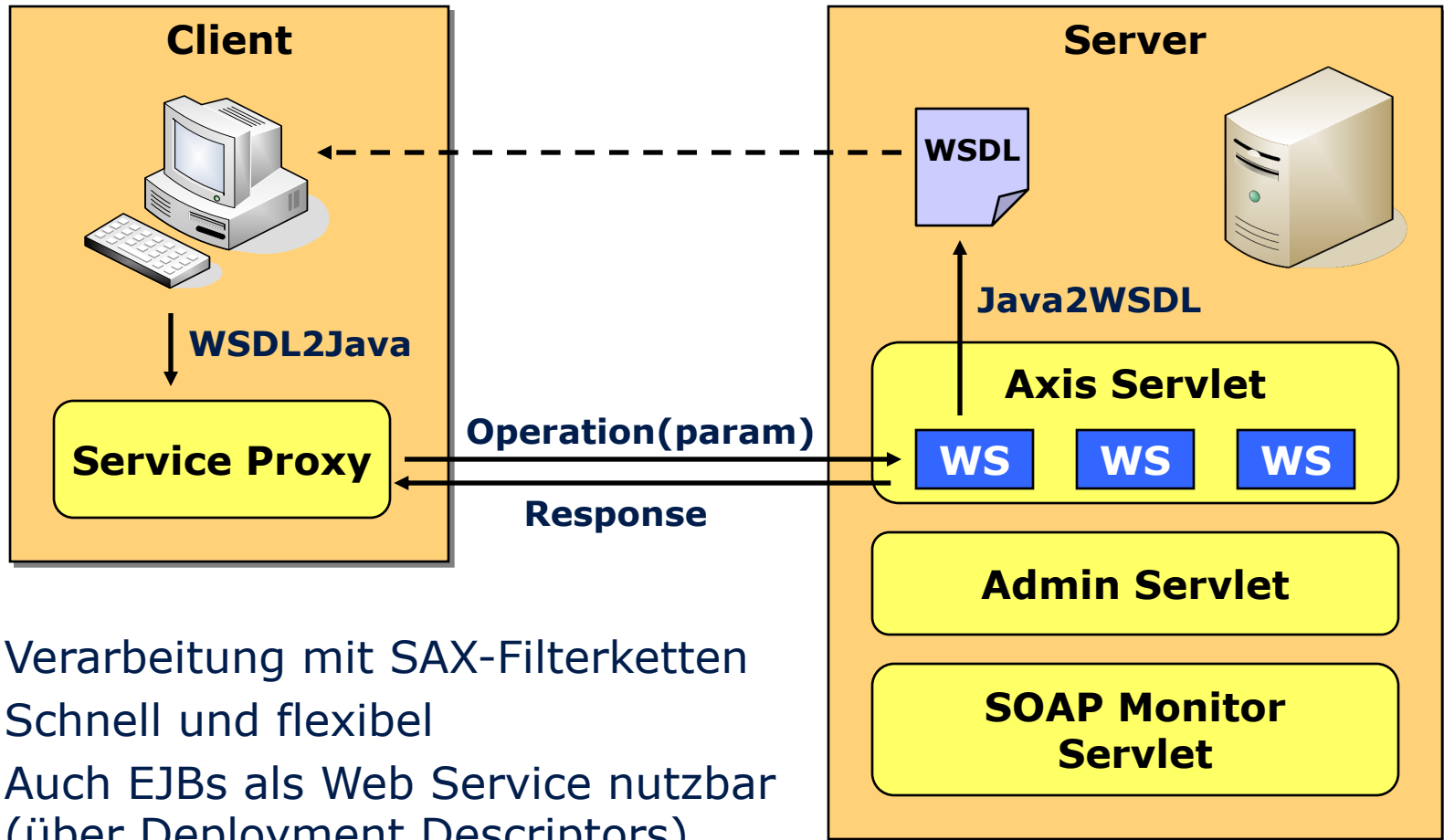
- auch Skriptsprachen wie PHP, Perl, Ruby, Python, etc.



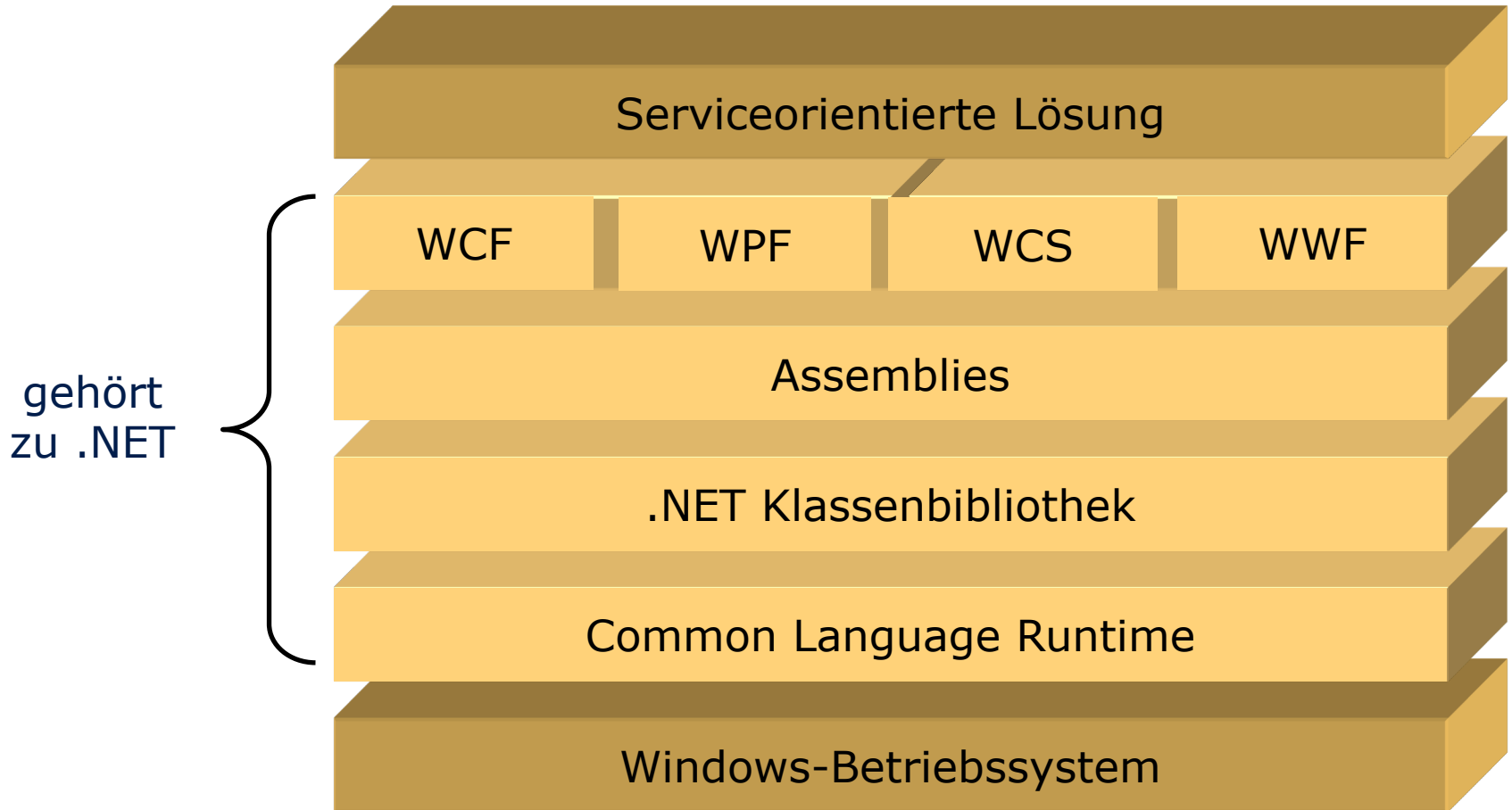
- 2 grundlegende Arten von Dienst Anbietern:
 - JAX-RPC Service Endpoints
 - EJB Service Endpoints
- 3 Arten von Dienstanwender-Proxys:
 - Generated Stub
 - Dynamic Proxy
 - Dynamic Invocation Interface
- J2EE unterstützt wie .NET die Entwicklung von WS-I Basic-Profile-kompatiblen Diensten
 - fördert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

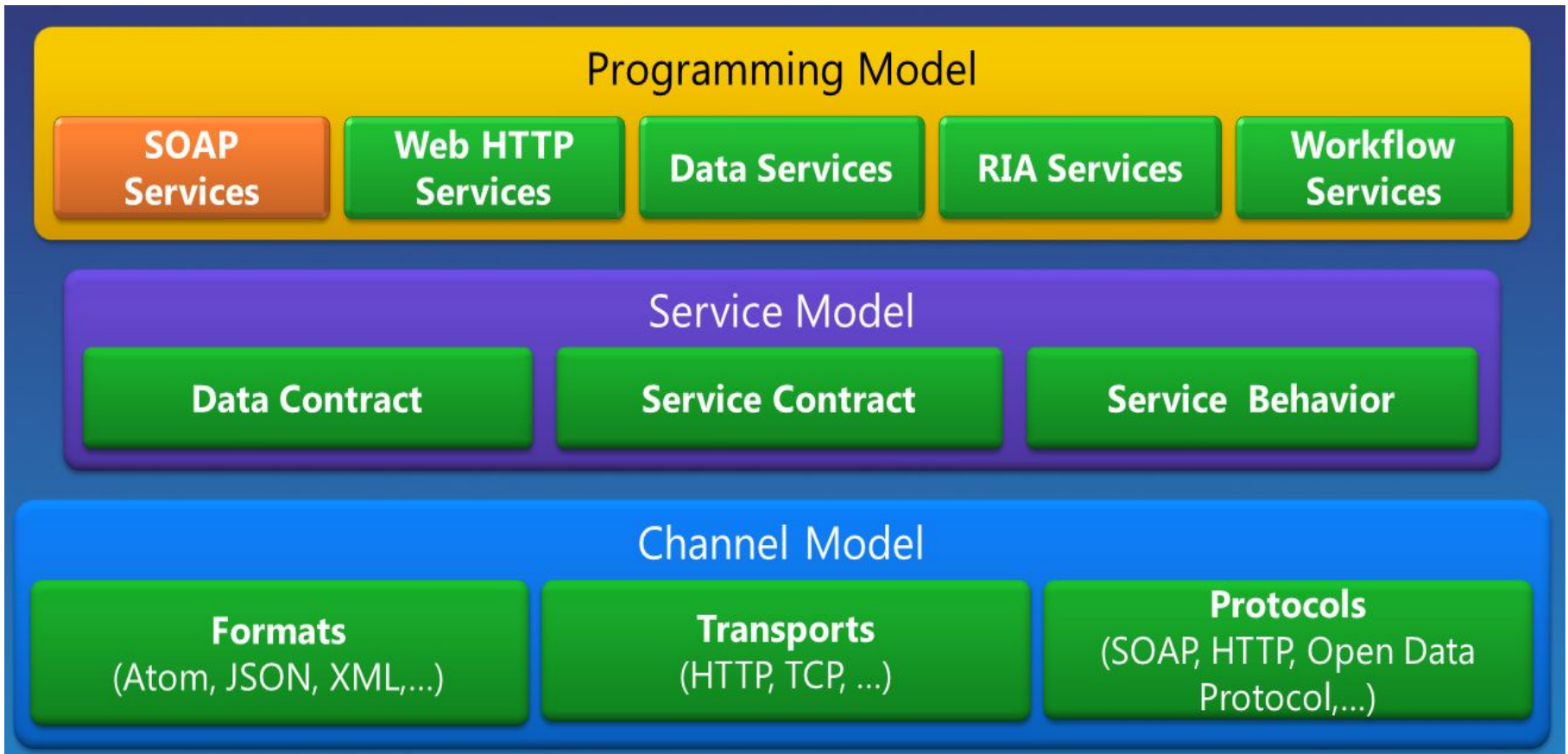
- **JAXP**: Java API for XML Processing, zur Verarbeitung von XML
 - `javax.xml.parsers`: Klassen für SAX- und DOM-Parser
 - `org.w3c.dom` / `org.xml.sax`: jeweilige Dokumentenmodelle
 - `javax.xml.transform`: XSLT-Transformationsfunktionen
- **JAX-RPC**: Java API for XML-based RPC, verarbeitet SOAP-Nachrichten (sowohl `rpc`- als auch `document`-Style)
 - `javax.xml.rpc` / `javax.xml.rpc.server`: Kernfunktionen
 - `javax.xml.rpc.handler` / `handler.soap`: Messagehandler
 - `javax.xml.soap` / `javax.xml.rpc.soap`: API-Funktionen zur Verarbeitung von SOAP-Inhalten und Bindings
- **JAXR**: Java API for XML Registries: ebXML- und UDDI-Register
- **JAXM**: Java API for XML Messaging: asynchrone SOAP-Komm.
- **SAAJ**: SOAP with Attachments API for Java
- **JAXB**: Java Architecture for XML Binding, Klassen aus XSD

- 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



- Verarbeitung mit SAX-Filterketten
⇒ Schnell und flexibel
- Auch EJBs als Web Service nutzbar (über Deployment Descriptors)





Quelle: MSDN.microsoft.com

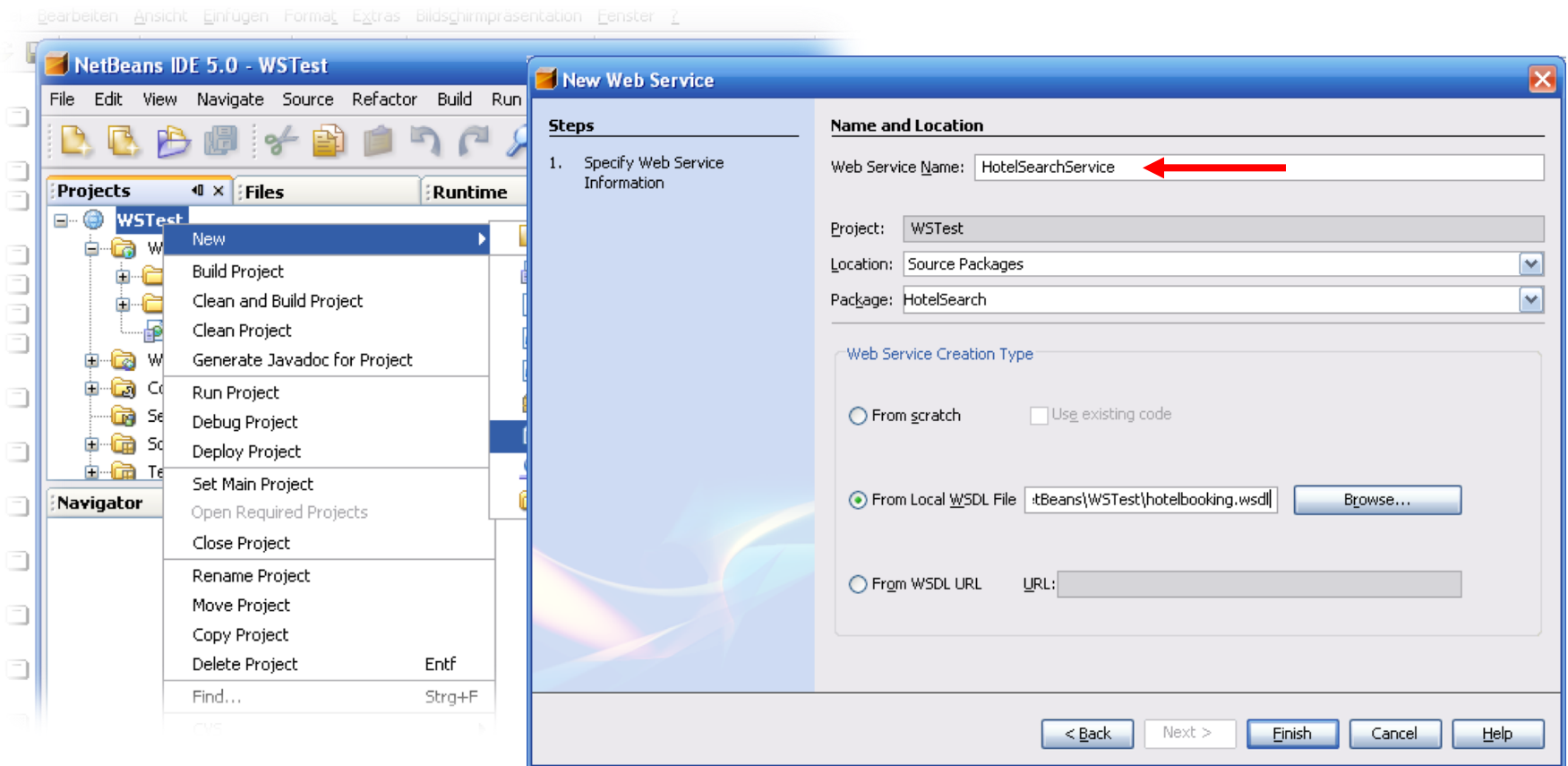
Service Model:

- **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,...)

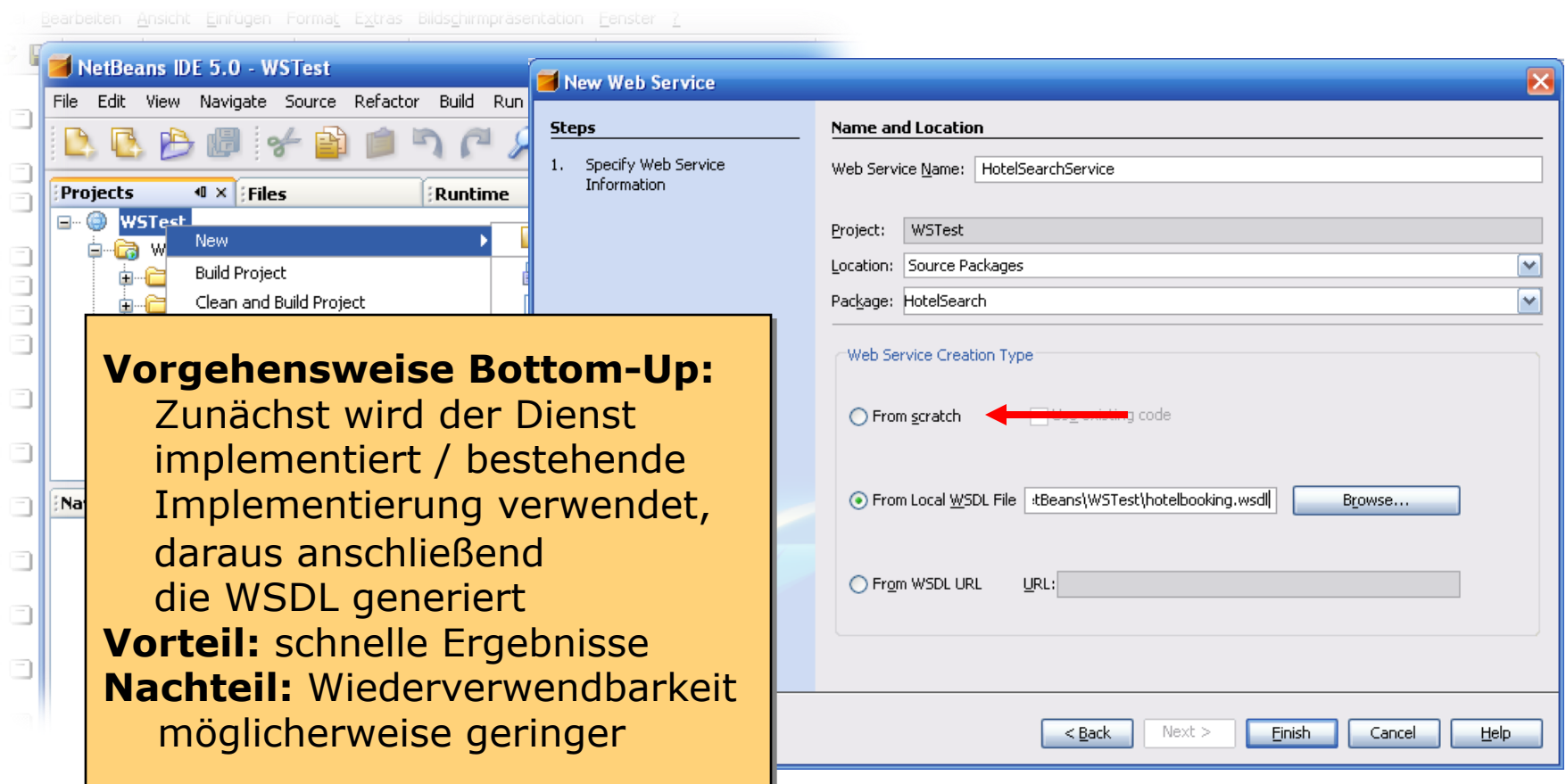
Channel Model:

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

Beispiel: Java-Entwicklungsumgebung NetBeans IDE 5.0 (aktuell 8.2)



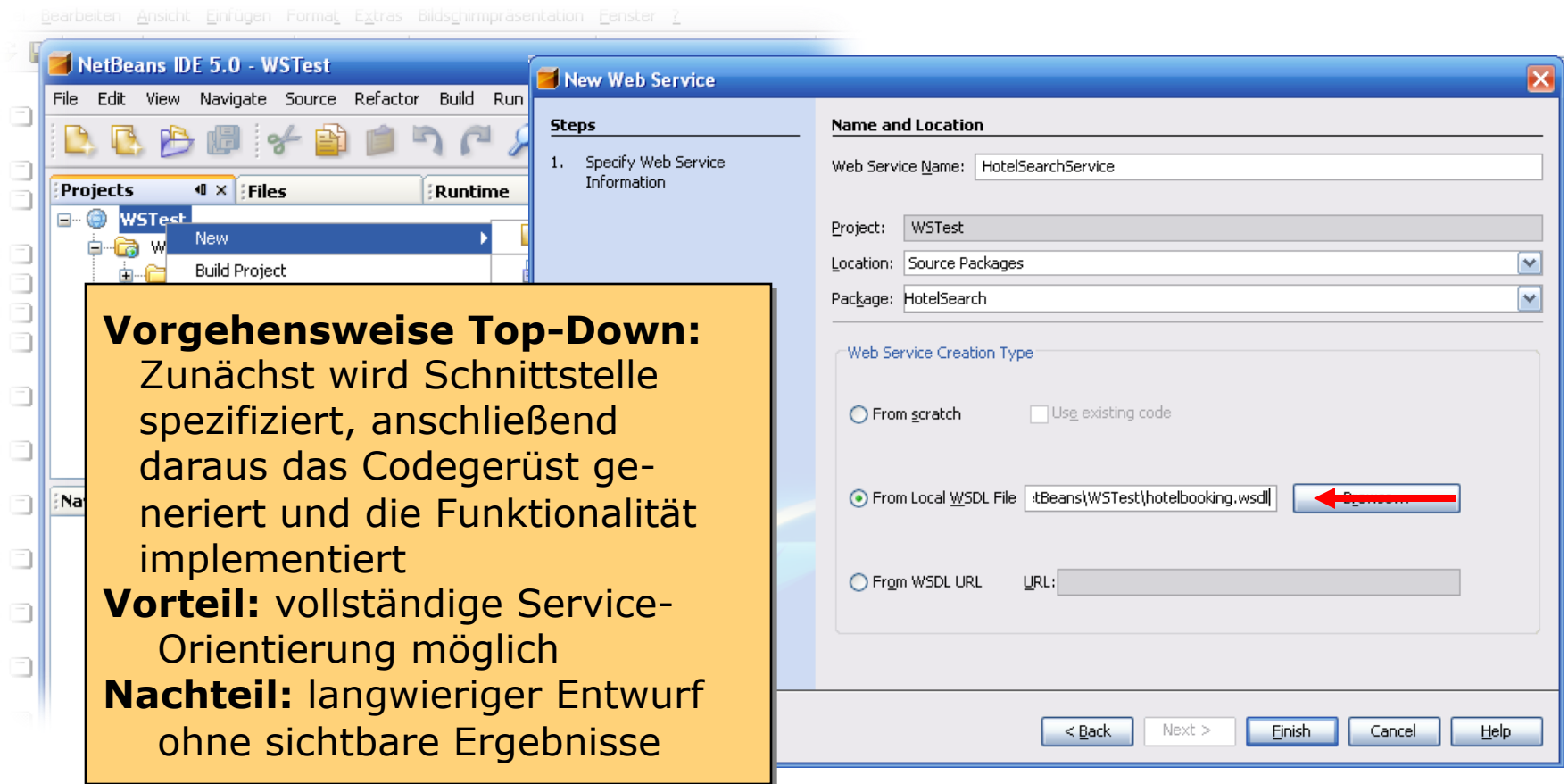
- Beispiel: Java-Entwicklungsumgebung NetBeans IDE 5.0



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

Vorteil: schnelle Ergebnisse
Nachteil: Wiederverwendbarkeit möglicherweise geringer

- Beispiel: Java-Entwicklungsumgebung NetBeans IDE 5.0

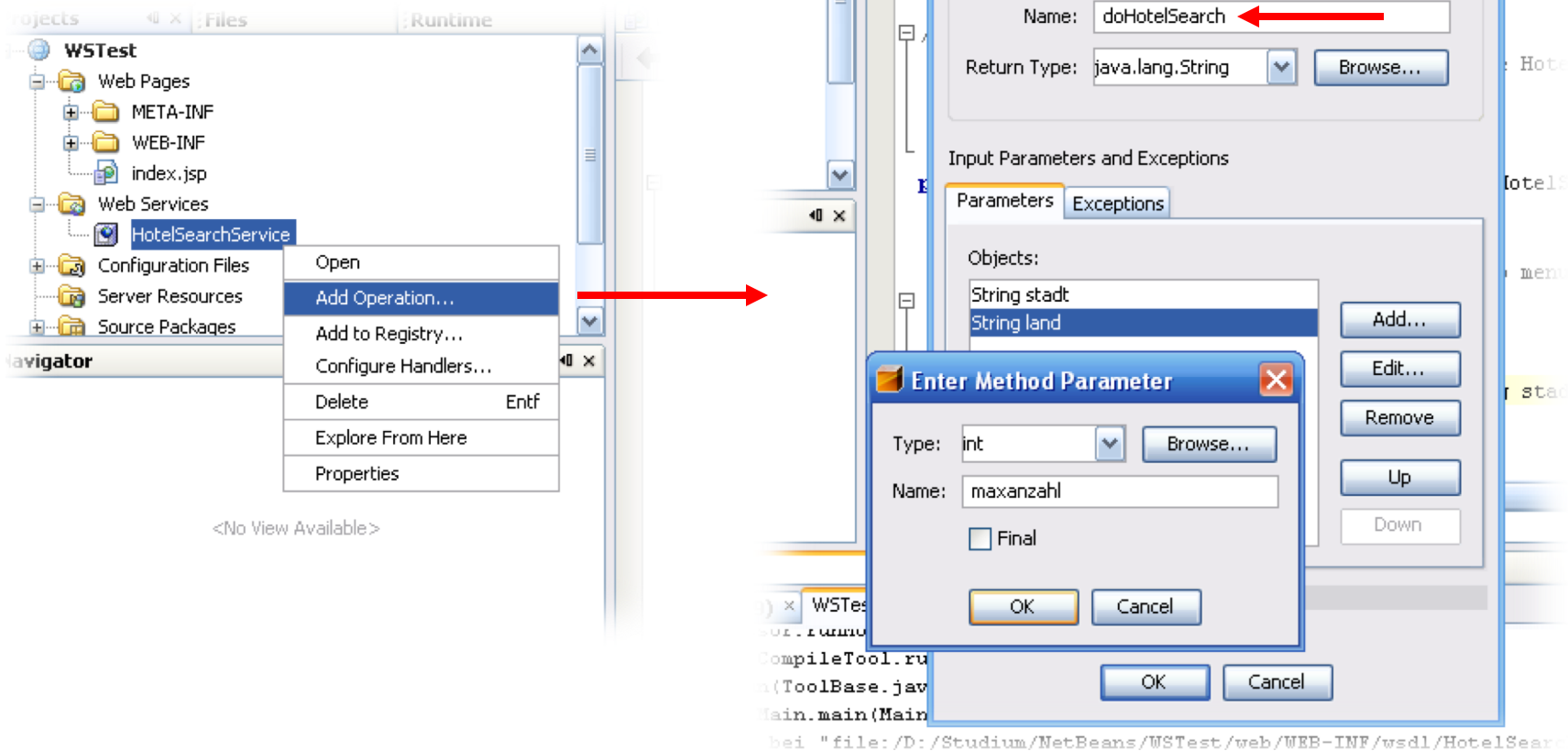


Vorgehensweise Top-Down:
Zunächst wird Schnittstelle spezifiziert, anschließend daraus das Codegerüst generiert und die Funktionalität implementiert

Vorteil: vollständige Service-Orientierung möglich

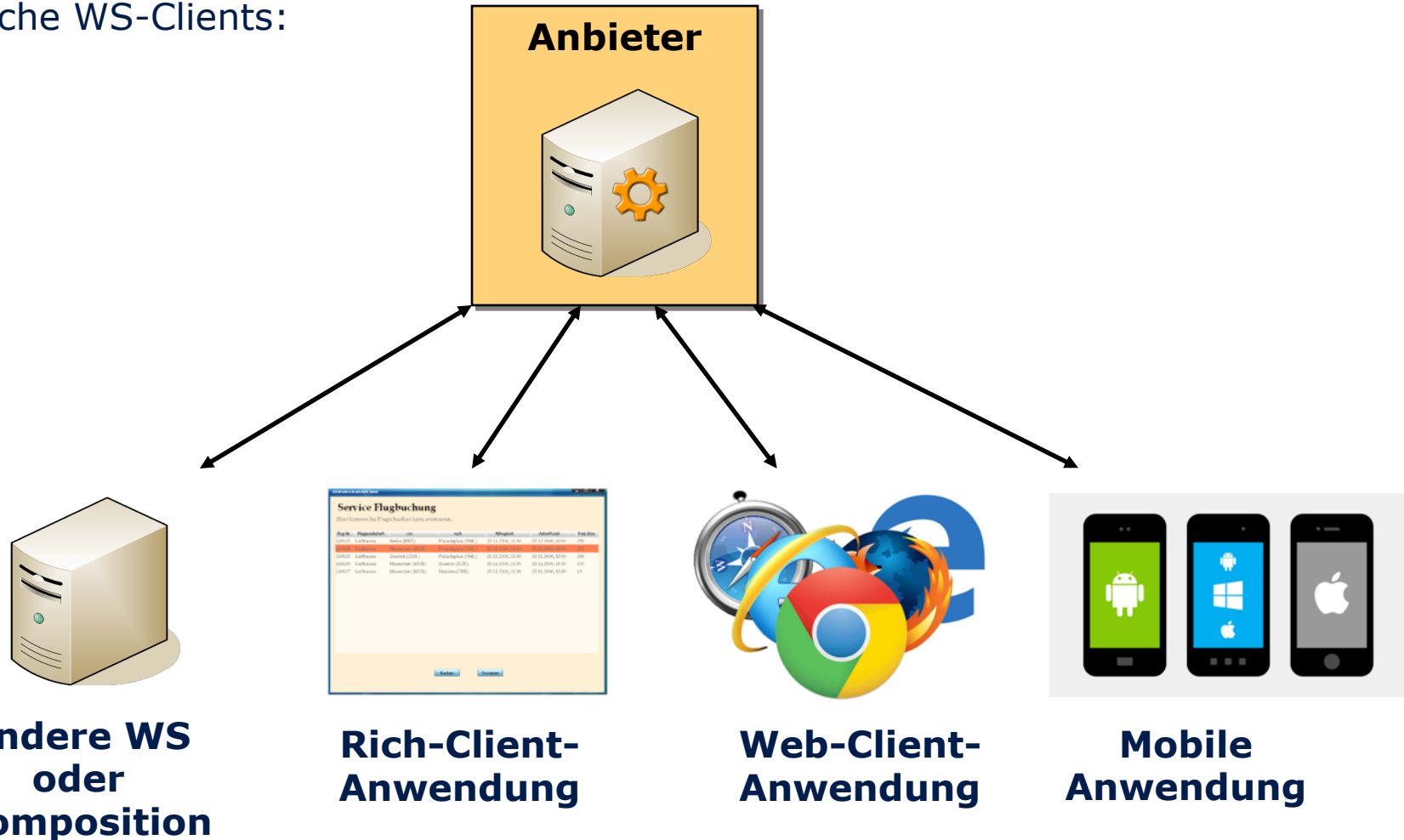
Nachteil: langwieriger Entwurf ohne sichtbare Ergebnisse

- Beispiel: NetBeans IDE 5.0



The screenshot illustrates the process of adding a new operation to a web service in NetBeans IDE 5.0. The 'Add Operation...' dialog is open, showing the 'Method' section with the name 'doHotelSearch' and the return type 'java.lang.String'. The 'Input Parameters and Exceptions' section is also visible, showing the 'Parameters' tab with the parameter 'maxanzahl' of type 'int'. The 'Enter Method Parameter' dialog is also open, showing the parameter 'maxanzahl' of type 'int'.

Mögliche WS-Clients:



- 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);
    }
}
```

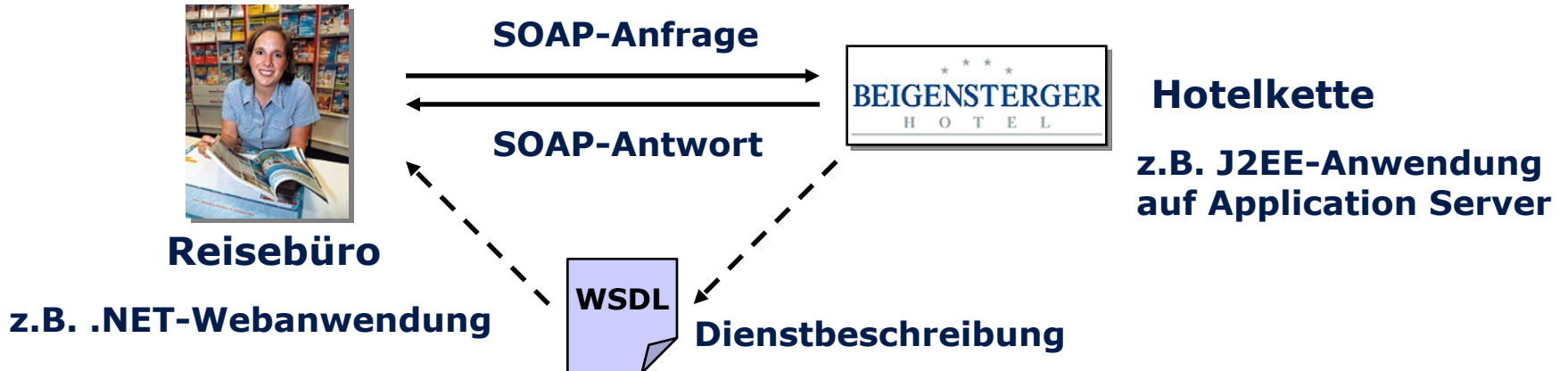
- Vorteil: Dynamische Stub-Generierung zur Laufzeit
- Beispiel: Ruby on Rails
 - Web Service Unterstützung mit SOAP4R Package (in Ruby enthalten)

```
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

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

- 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



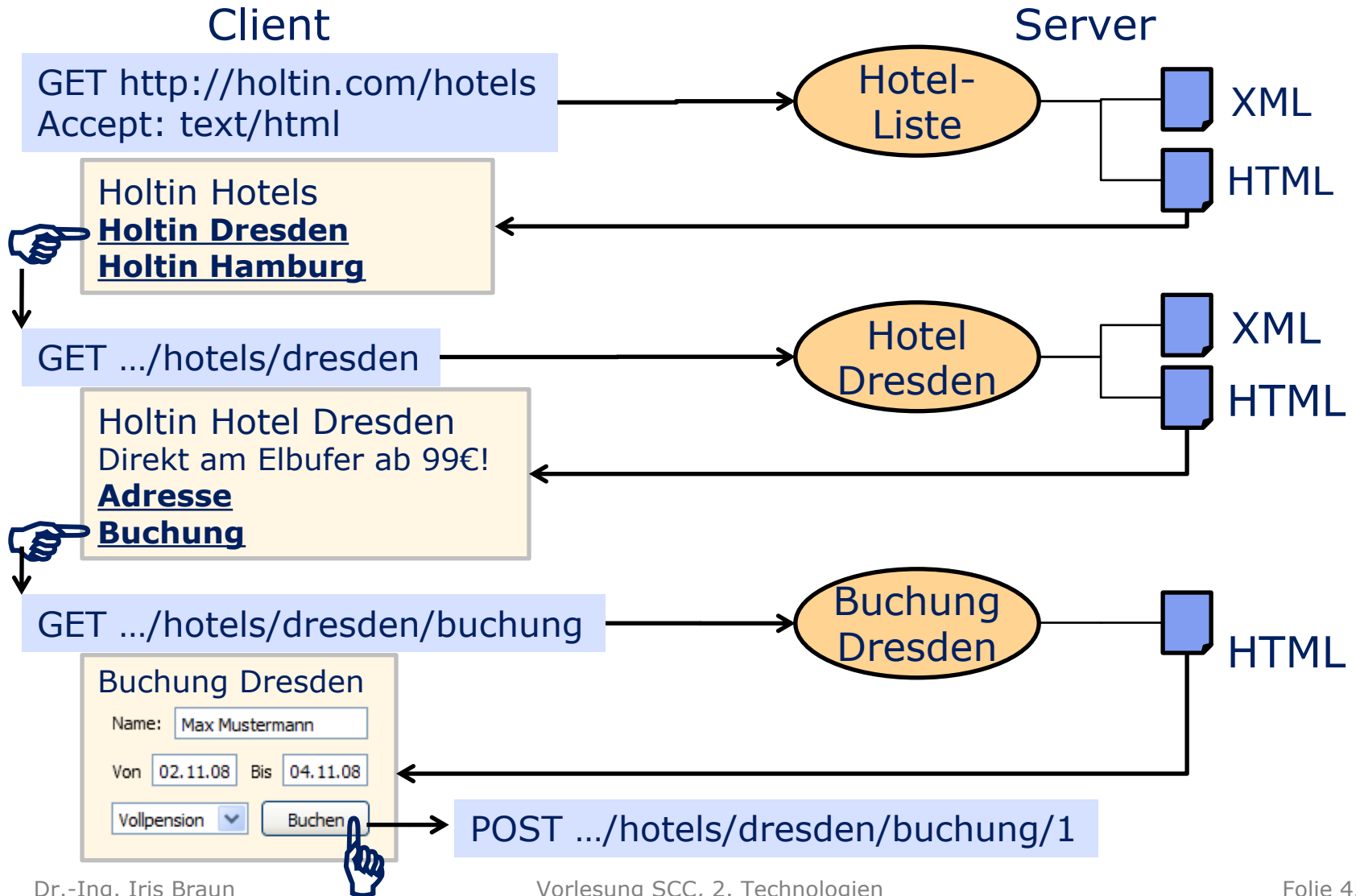
REST: Representational State Transfer

- 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





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

Beispiel SOAP Web Service

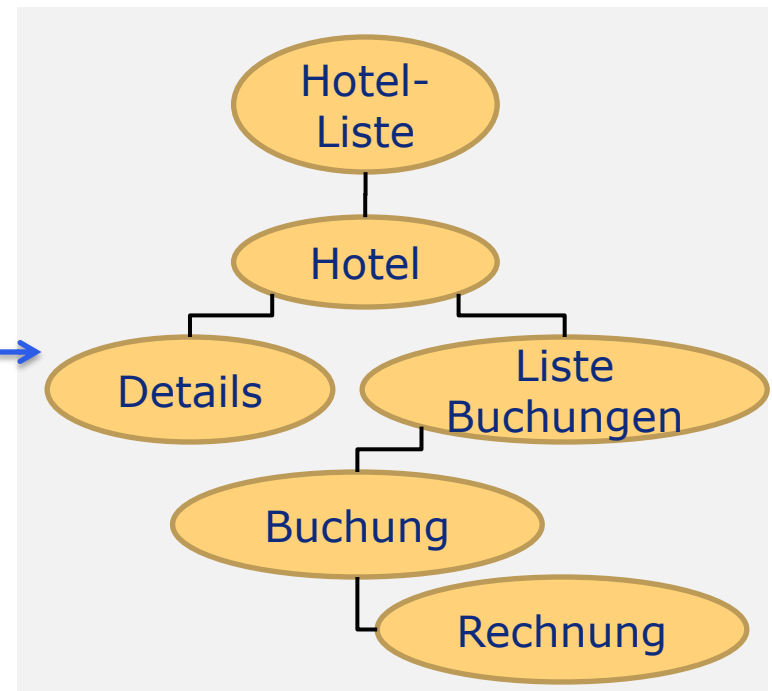
<http://holtin.com/HotelService>

- getHotelList
- checkRoomAvailability
- bookHotelRoom
- cancelBooking
- obtainInvoice

- Z.B. cancelBooking = DELETE Buchung

GET
PUT
POST
DELETE

REST Web Service

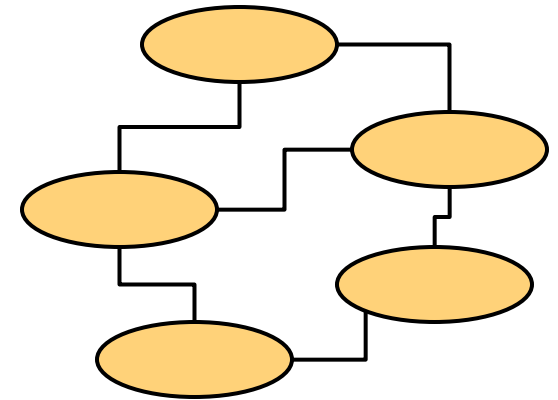


- Häufig: „REST ist XML über HTTP“ (HTTP+POX: „HTTP and plain old XML“)
 - Query-Parameter kaschieren spezielle Methodenaufrufe, oft gibt es nur eine Ressource, die die gesamte Funktionalität bereitstellt
- Kein REST im eigentlichen Sinn!
- Besser: Ressourcen-orientierte Architektur (ROA) [RR07]

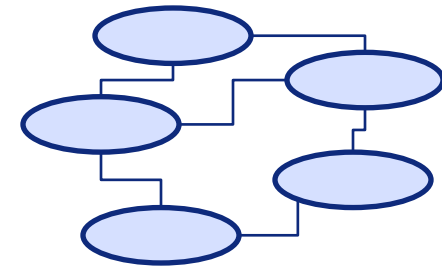
~~GET .../rest?action=cancelbooking&bid=1
Accept: application/xml~~

~~...~~

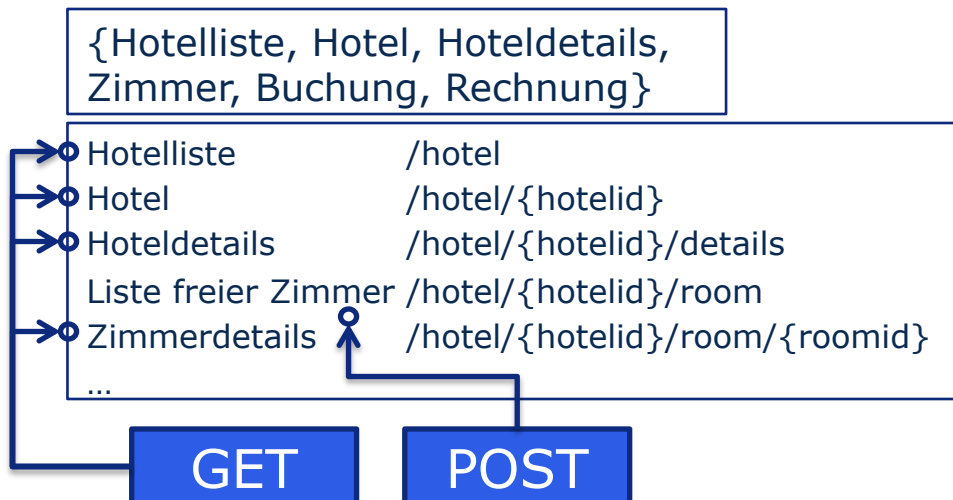
~~<?xml version="1.0"?>
<booking id="1">
 <state>cancelled</state>
</booking>~~



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)

- 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

```
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();
    }
}
```

[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](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/>

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>

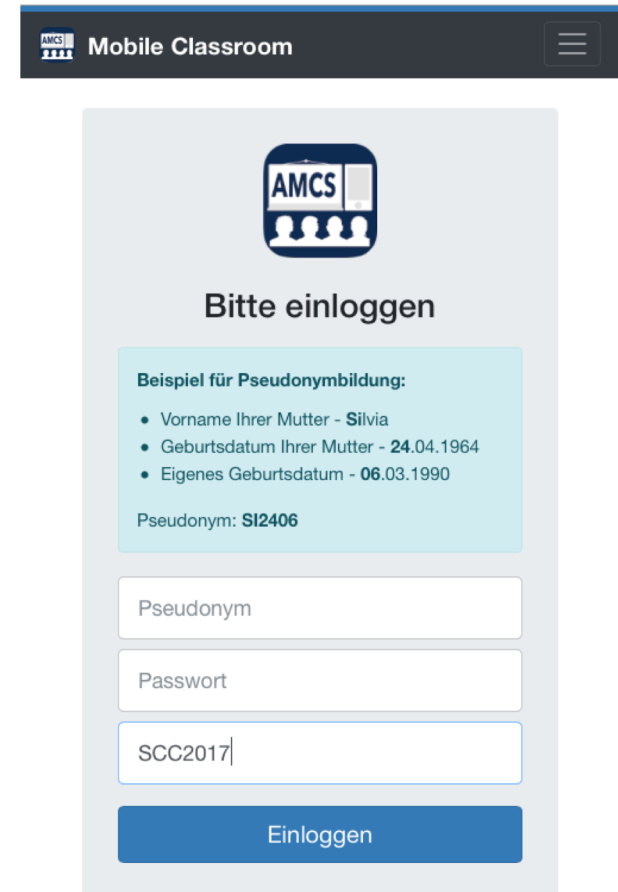
Web-Zugang:

<https://amcs.website>

PIN:

SCC2020

Bitte erstellen Sie sich ein Pseudonym, das keine Rückschlüsse auf Ihre Person erlaubt!



The screenshot shows the mobile interface for the AMCS Mobile Classroom. At the top, there is a dark blue header with the AMCS logo and the text "Mobile Classroom" on the left, and a hamburger menu icon on the right. Below the header, the AMCS logo is centered, followed by the text "Bitte einloggen". A light blue box contains an example for pseudonym creation:

Beispiel für Pseudonymbildung:

- Vorname Ihrer Mutter - Silvia
- Geburtsdatum Ihrer Mutter - 24.04.1964
- Eigenes Geburtsdatum - 06.03.1990

Pseudonym: SI2406

Below this, there are three input fields: "Pseudonym", "Passwort", and a field containing "SCC2017". At the bottom, there is a blue button labeled "Einloggen".