Wohlgeformte XML-Dokumente

Freie Universität Berlin

1. Jedes Anfangs-Tag
   muss ein zugehöriges Ende-Tag haben.
2. Elemente dürfen sich nicht
   überlappen.
3. XML-Dokumente haben genau ein
   Wurzel-Element.
4. Element-Namen müssen
   bestimmten Namenskonventionen entsprechen.
5. XML beachtet grundsätzlich
   Groß- und Kleinschreibung.
6. XML belässt White Space
   im Text.
7. Ein Element darf niemals zwei Attribute
   mit dem selben Namen haben.

DTD-Beispiel

```
<!ELEMENT BookStore (Book+)>
```

```
<!ELEMENT BookStore (Book | (Book, BookStore))>
```

Die komplette Beispiel-DTD

```
<!ELEMENT BookStore (Book+)>
<!ELEMENT Book
        (Title, Author, Date, ISBN, Publisher)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>
```

Äquivalentes XML-Schema

Freie Universität Berlin

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                targetNamespace="http://www.books.org">
   <xsd:element name="BookStore">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="Book"  maxOccurs="unbounded">
               <xsd:complexType>
                  <xsd:sequence>
                     <xsd:element name="Title" type="xsd:string"/>
                     <xsd:element name="Author" type="xsd:string"/>
                     <xsd:element name="Date" type="xsd:string"/>
                     <xsd:element name="ISBN" type="xsd:string"/>
                     <xsd:element name="Publisher" type="xsd:string"/>
                  </xsd:sequence>
               </xsd:complexType>
            </xsd:element>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

XSLT-Beispiel

Freie Universität Berlin

Stylesheet

```
<xsl:template match="A">
  <xsl:value-of select="@id"/>
</xsl:template>

<xsl:template match="B">
   <xsl:value-of select="@id"/>
 </xsl:template>

 <xsl:template match="C">
   <xsl:value-of select="@id"/>
 </xsl:template>

<xsl:template match="D">
  <xsl:value-of select="@id"/>
 </xsl:template>
```

Dokument

```
<source>
  <A id="a1">
      <B id="b1"/>
      <B id="b2"/>
  </A>
  <A id="a2">
      <B id="b3"/>
      <B id="b4"/>
      <C id="c1">
          <D id="d1"/>
      </C>
      <B id="b5">
          <C id="c2"/>
      </B>
  </A>
</source>
```

Freie Universität Berlin

Warum XML (und abhängige Technologien), wenn es doch Datenbanken gibt?

Markus Luczak-Rösch
Freie Universität Berlin
Institut für Informatik
Netzbasierte Informationssysteme
markus.luczak-roesch@fu-berlin.de

Das Ende der Welt

Web Services

Markus Luczak-Rösch
Freie Universität Berlin
Institut für Informatik
Netzbasierte Informationssysteme
markus.luczak-roesch@fu-berlin.de

heutige Vorlesung

- Was sind Web Services?
- Web Services – Basistechnologien:
  - SOAP
  - WSDL
  - UDDI
- RPC vs. Messaging

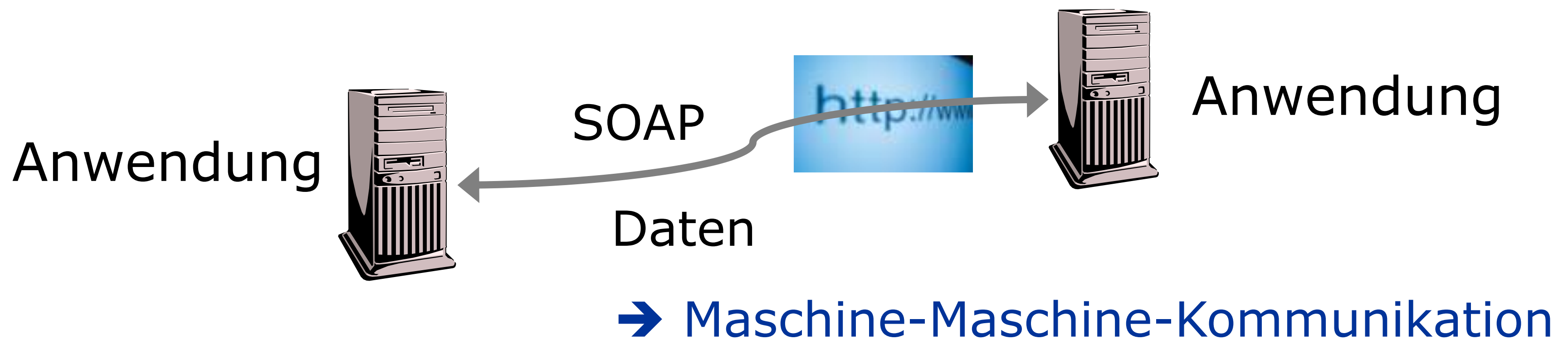
Was sind Web Services?

Was sind Web Services?

traditionelle Web-Anwendung

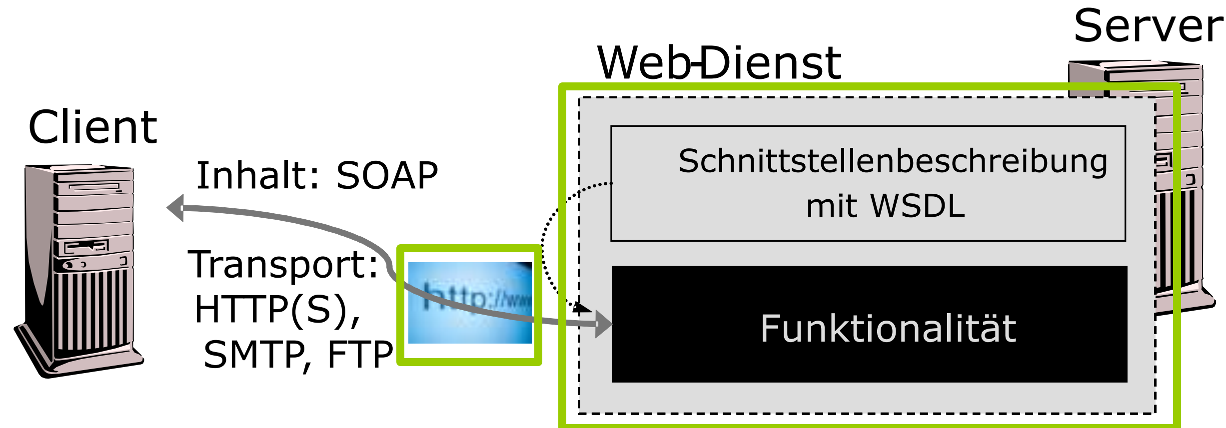
➔ Mensch-Maschine-Kommunikation

Web Service



➔ Maschine-Maschine-Kommunikation

Definition



Ein **Web Service ist eine Softwareanwendung**, die

1. mit einer **URI** eindeutig identifizierbar ist,
2. über eine **WSDL**-Schnittstellenbeschreibung verfügt,
3. nur über die in ihrer WSDL beschriebenen Methoden zugreifbar ist und
4. über **gängige Internet-Protokolle** unter Benutzung von XML-basierten Nachrichtenformaten wie z.B. SOAP zugreifbar ist.

Eigenschaften von Web Services

Web Services

- implementieren häufig keine neuen Systeme, sondern sind **Fassade** für bestehende Systeme
- abstrahieren von Programmiersprache und Plattform mit der die Anwendung realisiert ist: **Virtualisierung von Software**
- zwei Erscheinungsformen:
  - **RPCs** (synchron)
  - **Messaging** (asynchron)

Protokolle

Layer	Protokoll/Standards
Messaging	HTML
Transport	HTTP, FTP, SMPT
Network	TCP/IP, UDP

XML-gekapselte Protokolle

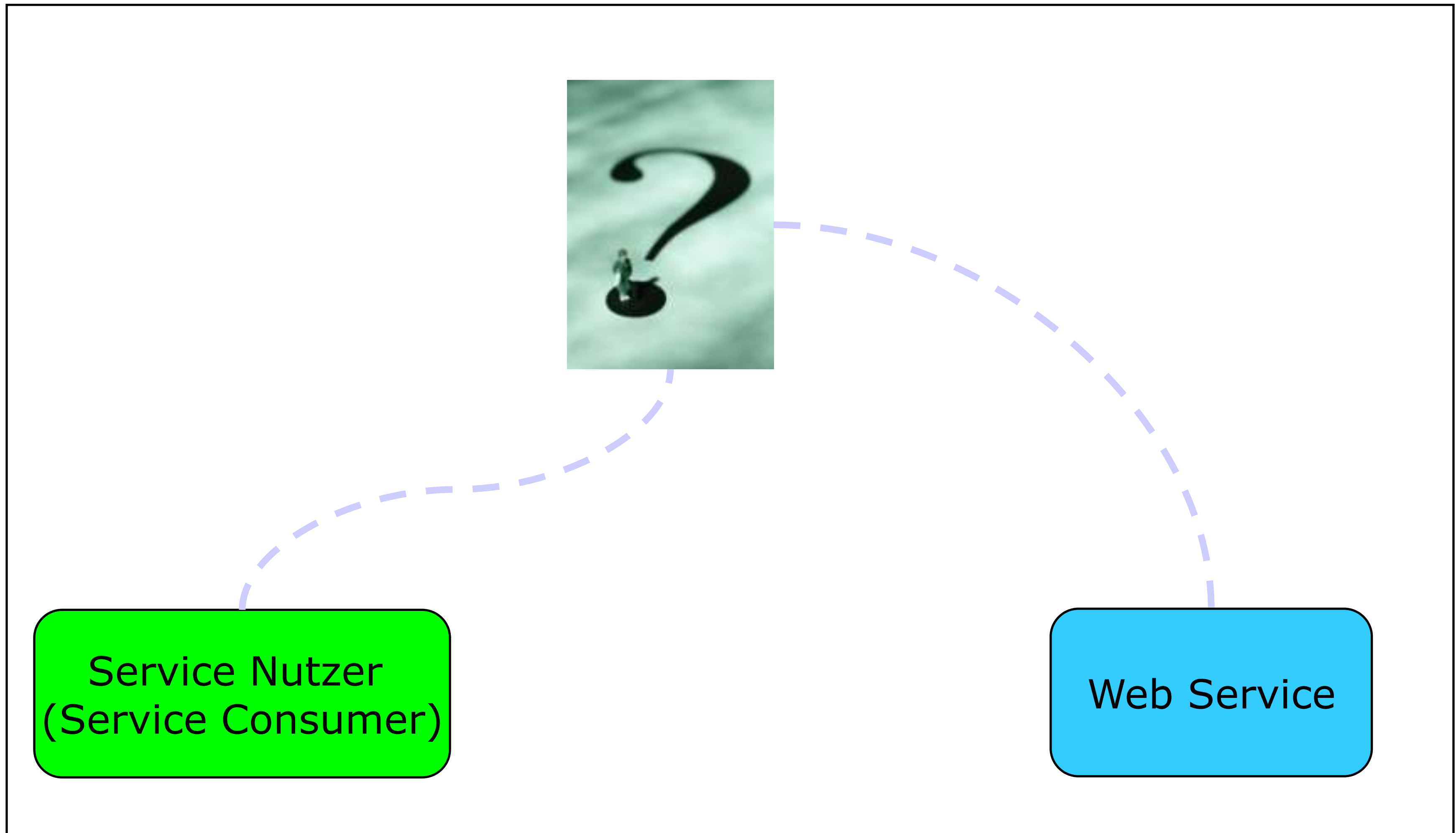
Freie Universität Berlin

Layer	**Protokoll/Standards**
Content	Content Information
Messaging	SOAP, XML-RPC
Transport	HTTP, FTP, SMPT
Network	TCP/IP, UDP

Web Service Technology Stack

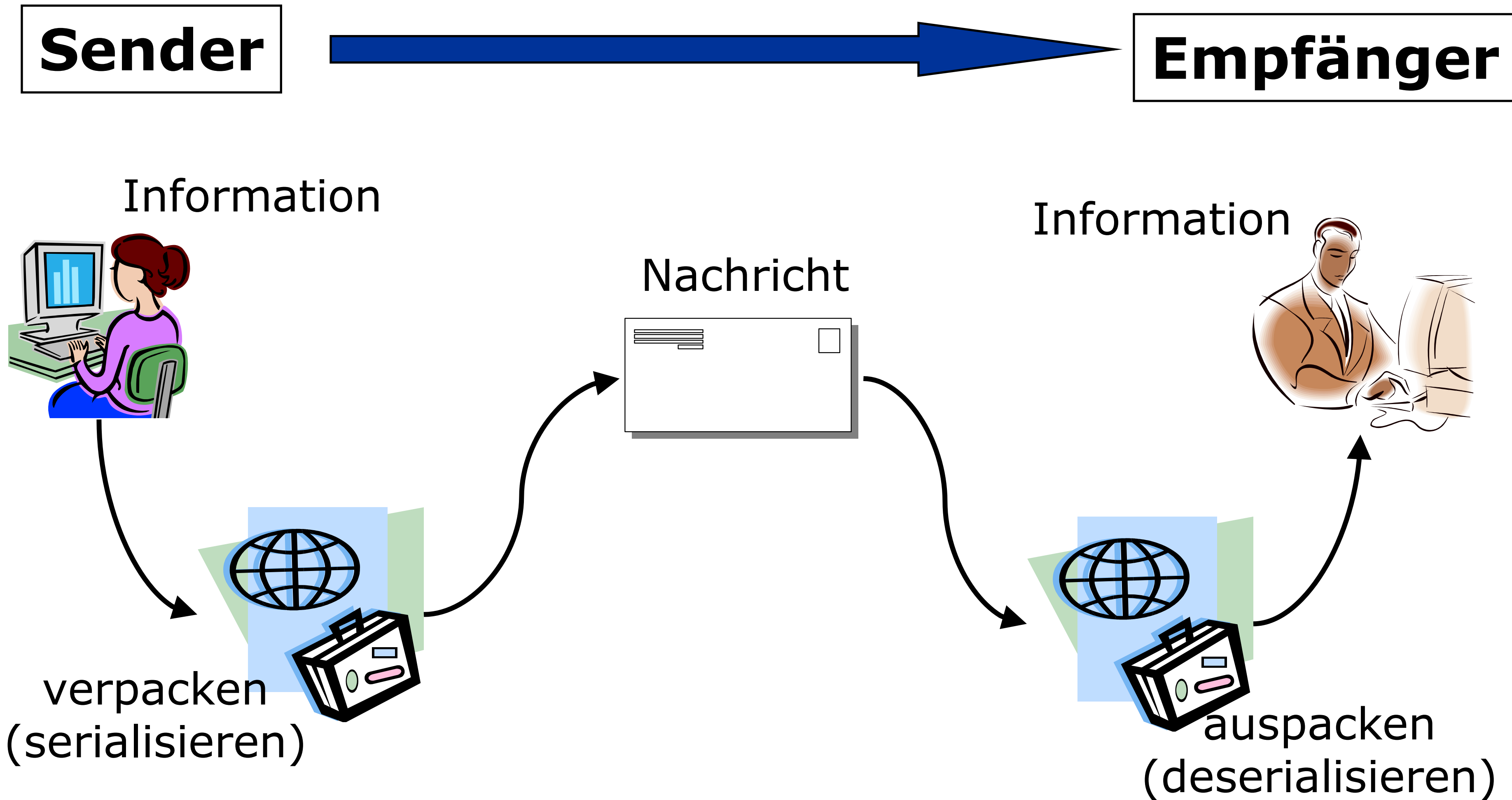
Layer	Protokoll/Standards
Discovery *(holt Service-Beschreibung von Prodiver)*	UDDI, DISCO, WSIL
Description *(beschreibt Services)*	WSDL
Messaging	SOAP, XML-RPC
Transport *(Applikation-zu-Applikation Kommunikation)*	HTTP, FTP, SMPT
Network *(Netzwerk Layer)*	TCP/IP, UDP

Web Services



Web Service Basiskomponenten: *SOAP*

Austausch einer SOAP-Nachricht



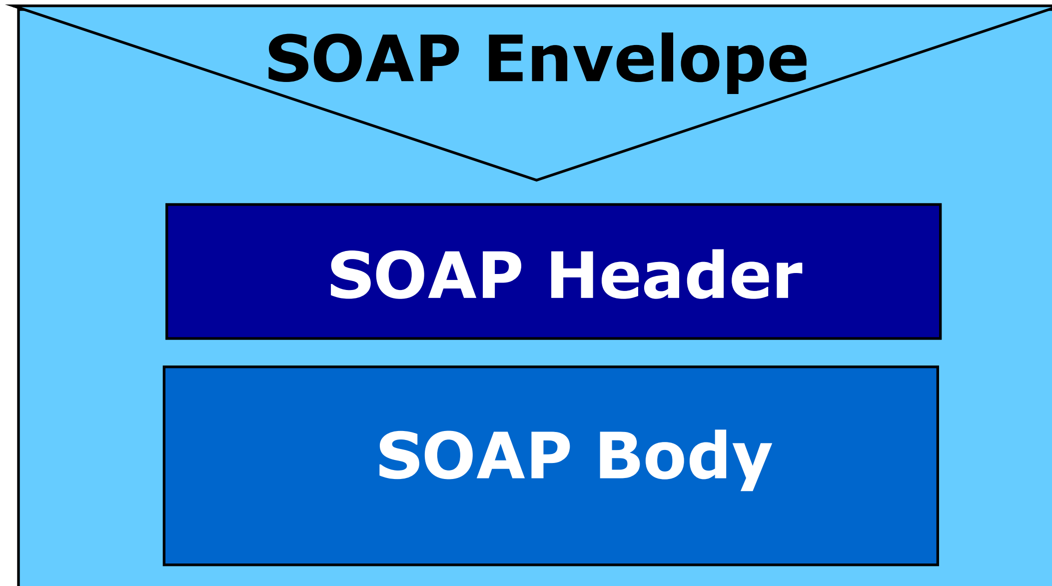
SOAP ist…

Freie Universität Berlin

- …eine **Kommunikationskomponente** von Web Services
- …ein **Protokoll** für Nachrichtenaustausch zwischen Web Service-Konsument und Web Service-Anbieter
- …**XML-basiert** (nutzt XML für die Darstellung von Nachrichten)
- …Plattform- & Programmierspracheunabhängig

Die SOAP-Spezifikation legt fest, wie eine Nachricht übertragen wird. Die Umsetzung der Nachricht ist nicht Gegenstand der SOAP-Spezifikation

Aufbau einer SOAP-Nachricht



XML-Deklaration

```
<?xml version='1.0' encoding='UTF-8'?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap/envelope/">
    <!- SOAP Header -->
    <!- SOAP Body -->
</env:Envelope>
```

SOAP Version 1.2

SOAP Version 1.1:
http://schemas.xmlsoap.org/soap/envelope/

Nachrichtenformat SOAP

Freie Universität Berlin

HTML

```
<html>
  <head>
    Zusatzinformationen
  </head>
  <body>
    Inhalt: Webseite
  </body>
</html>
```

SOAP

```
<Envelope>
  <Header>
    Zusatzinformationen
  </Header>
  <Body>
    Inhalt: XML-Daten
  </Body>
</Envelope>
```

- XML-basierter W3C-Standard
  - SOAP 1.1: W3C Note von 2000
  - SOAP 1.2: W3C Recommendation von 2003

Prinzipieller Aufbau (SOAP V.1.1)

Freie Universität Berlin

```
<?xml version='1.0' encoding='UTF-8'?>
<env:Envelope
xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
>
   <env:Header>
      Zusatzinformationen
   </env:Header>
   <env:Body>
      Nachrichtinhalt
    </env:Body>
</env:Envelope>
```

- Wurzel-Element: Envelope aus SOAP-Namensraum
- Header: optional
- Body: obligatorisch
- Im Beispiel: kein W3C-Namensraum → SOAP 1.1

SOAP Block

SOAP Nachricht → ein XML Dokument das beinhaltet:

- obligatorisches **Envelope Element** – identifiziert ein XML Dokument als SOAP Nachricht
- optionales **Header Element** – Header Informationen
- obligatorisches **Body Element** – Call & Response Informationen
- optionales **Fault Element** – Informationen über Fehler

SOAP Envelope Element

Freie Universität Berlin

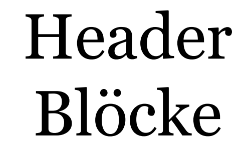
```
<?xml version="1.0"?>
<env:Envelope
  xlmns:env="http://www.w3.org/2003/05/soap-envelope">
   …
</env:Envelope>
```

- Im Beispiel: W3C-Namensraum → SOAP 1.2

- Name des Elements: Envelope
- Envelope: Wurzel-Element einer SOAP Nachricht
- beinhaltet SOAP Namespace
- identifiziert SOAP Nachricht

Briefkopf (Header)


```
<env:Envelope ...>
    <env:Header xmlns:ns="URI" >
        <ns:Zusatzinformation-1>...</ns:Zusatzinformation-1>
          ...
        <ns:Zusatzinformation-n>...</ns:Zusatzinformation-n>
    </env:Header>
    <env:Body>...</env:Body>
</env:Envelope>
```



- Header: beliebige XML-Inhalte erlaubt
- Struktur von Anwendung festgelegt
- Header Block
  - Kind-Element von Header
  - Zusatzinformation zur eigentlichen Nachricht

Nachrichteninhalt (Body)

```
<env:Envelope ...>
  <env:Body xmlns:ns="URI">
    <ns:Nachrichtinhalt-Teil-1>...</ns:Nachrichteninhalt-Teil-1>
      ...
    <ns:Nachrichteninhalt-Teil-n>...</ns:Nachrichteninhalt-Teil-n>
  </env:Body>
</env:Envelope>
```

- Body: beliebige XML-Inhalte erlaubt
- Struktur von Anwendung festgelegt, z.B. durch:
  - speziellen Namensraum und/oder
  - WSDL-Beschreibung

Beispiel


```
<env:Envelope ...>
  <env:Header>
    <alertcontrol xmlns="http://example.org/alertcontrol">
      <priority>1</priority>
      <expires>2007-06-24T14:00:00-05:00</expires>
    </alertcontrol>
  </env:Header>
  <env:Body>
    <alert-msg xmlns="http://example.org/alert">
      Pick up Mary at 2pm!
    </alert-msg>
  </env:Body>
</env:Envelope>
```

Zusatz-information (Header Block)

Nachricht

mustUnderstand Attribut

Freie Universität Berlin

```
<env:Header>
    <alertcontrol xmlns="http://example.org/alertcontrol"
                env:mustUnderstand="true">
       …
    </alertcontrol>
</env:Header>
```

- **mustUnderstand="true"**: Empfänger muss Header Block verstehen oder mit Fehlermeldung antworten
- **mustUnderstand="false"**: Empfänger kann Header Block (ohne Fehlermeldung) ignorieren
- kann für jeden Header Block unterschiedlich sein
- Beachte: Standard-Wert ist "false"

SOAP Verarbeitung

Freie Universität Berlin

Empfänger **muss verarbeiten**:

- Body
- Header Blocks mit mustUnderstand="true"

Empfänger **darf ignorieren**:

- Header Blocks mit mustUnderstand="false"
- Header Blocks ohne mustUnderstand-Attribut
  Grund: "false" Standardwert von mustUnderstand

Eine SOAP-Anfrage an Google


```
<?xml version='1.0' encoding='UTF-8'?>
<env:Envelope
    xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="..."  xmlns:xsi="...">
  <env:Body>
    <doGoogleSearch xmlns="urn:GoogleSearch">
      <key xsi:type="xsd:string">3289754870548097</key>
      <q xsi:type="xsd:string">Eine Anfrage</q>
      <start xsi:type="xsd:int">0</start>
      <maxResults xsi:type="xsd:int">10</maxResults>
       …
    </doGoogleSearch>
  </env:Body>
</env:Envelope>
```

- **doGoogleSearch**(key, q, start, maxResults,...)
- hier kein Header
- Beachte: **Web Service-Aufruf kann als URL kodiert werden (REST)**

Und die SOAP-Antwort von Google


```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
                    xmlns:xsd="..."  xmlns:xsi="...">
  <env:Body>
    <ns1:doGoogleSearchResponse xmlns:ns1="urn:GoogleSearch" ...>
        <return xsi:type="ns1:GoogleSearchResult">
            ...
        </return>
    </ns1:doGoogleSearchResponse>
  </env:Body>
</env:Envelope>
```

- Antwort: **doGoogleSearchResponse**(return(...))
- Datentyp ns1:GoogleSearchResult in WSDL-Beschreibung definiert

Exkurs: Was fällt Ihnen auf?

Freie Universität Berlin

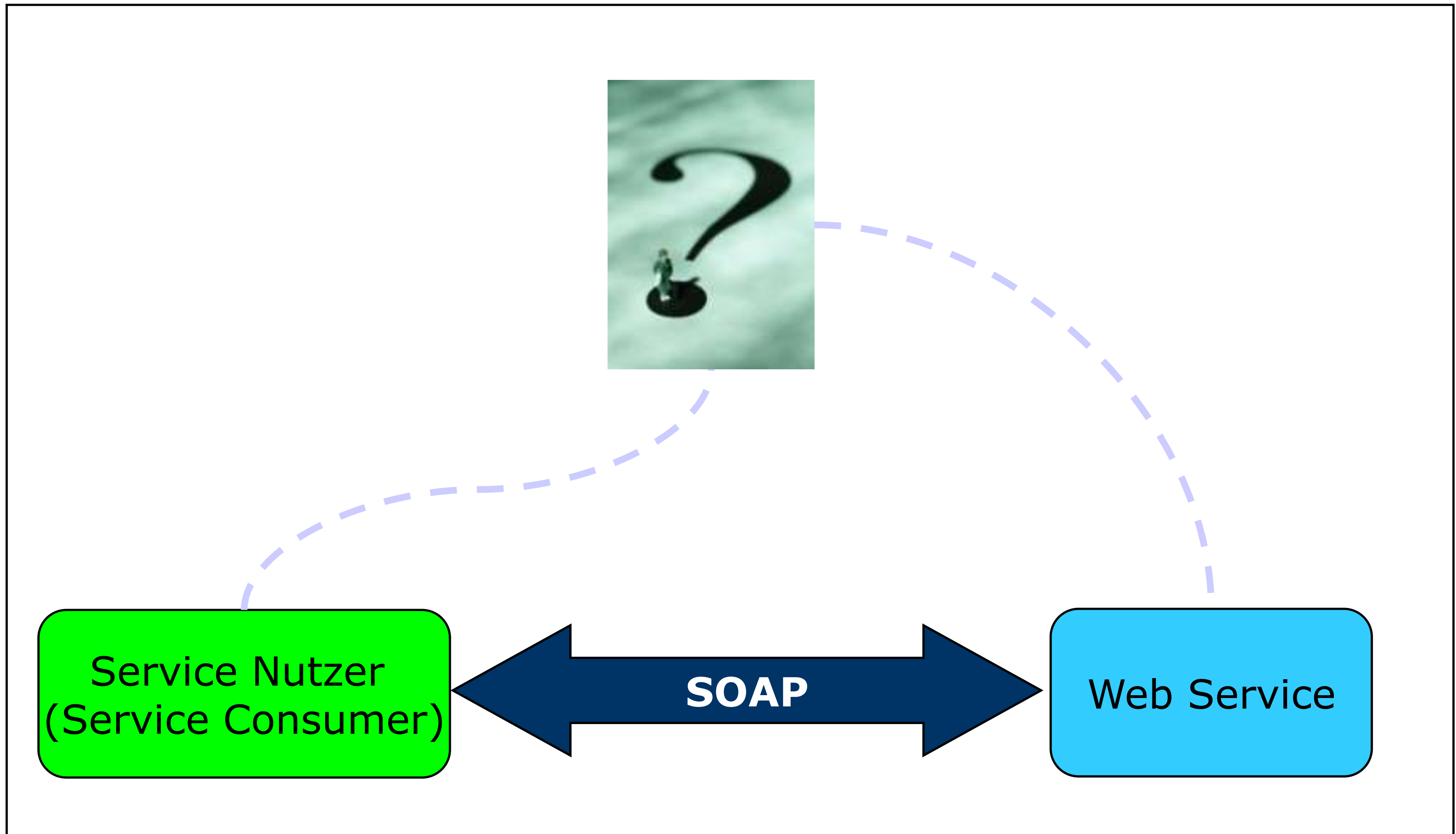
```
<?xml version='1.0' encoding='UTF-8'?>
<env:Envelope
    xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="..."  xmlns:xsi="...">
  <env:Body>
    <doGoogleSearch xmlns="urn:GoogleSearch">
      <key xsi:type="xsd:string">3289754870548097</key>
      <q xsi:type="xsd:string">Eine Anfrage</q>
      <start xsi:type="xsd:int">0</start>
      <maxResults xsi:type="xsd:int">10</maxResults>
       ...
    </doGoogleSearch>
  </env:Body>
</env:Envelope>
```

- **Was ist eine URN?**
  - **Ortsunabhängiger, globaler, lebenslanger Identifier**
  - **Übersetzung in URI übernimmt Anwendung**

Übertragung von SOAP-Nachrichten

Freie Universität Berlin

- heute meist über HTTP oder HTTPS
- Request-Response-Verhalten von HTTP unterstützt entfernte Prozeduraufrufe (RPCs)
- auch z.B. mit SMTP (Messaging) möglich


Service Nutzer
(Service Consumer)
SOAP
Web Service

SOAP - Vorteile

- + unabhängig von Übertragungsprotokollen
- + sowohl für RPCs als auch für Messaging geeignet
- + einfach erweiterbar
- + Erweiterungen unabhängig voneinander
- + Plattformunabhängig
- + Programmiersprachenunabhängig

SOAP - Nachteile

- zusätzlicher Verarbeitungsaufwand
- nicht so einfach zu erlernen
- für viele notwendige Erweiterungen noch kein etablierter Standard

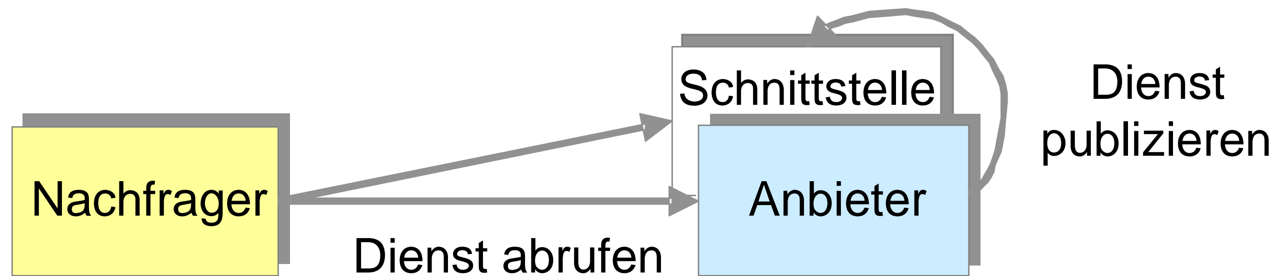
  Beispiel: wsu:identifier vs. wsa:MessageID

Web Service Basiskomponenten: *WSDL*

Wozu dient WSDL?

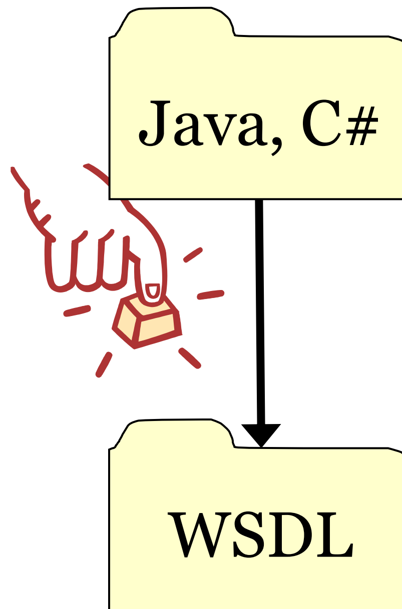
Freie Universität Berlin

Formale Beschreibung der Schnittstelle von Services



- Client möchte bestimmten Web Service nutzen
- Client benötigt hierfür:
  - Struktur des Aufrufes: Name, Parameter, Ergebnis, Fehlermeldungen
  - Übertragungsprotokoll und Web-Adresse
- genau dies wird mit WSDL beschrieben

Wozu WSDL-Syntax verstehen?



- WSDL = zu veröffentlichende Schnittstellenbeschreibung (Vertrag)
- Nutzer des Web Service kennt nur WSDL, nicht Programm-Code

⇨ Web-Service-Anbieter/Nutzer sollten WSDL (Vertrag) verstehen!

- mögliche Probleme bei generierten WSDLs:
  - Fehlermeldungen nicht korrekt beschrieben
  - optionale RPC-Parameter ungünstig beschrieben

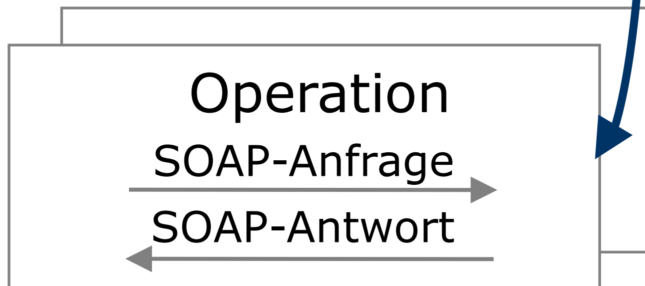
WSDL Aufbau

- beschreibt Netzwerkdienste als Kommunikationsendpunkte (Ports), die bestimmte Nachrichten über bestimmte Protokolle austauschen

Grundidee

abstrakte Schnittstelle

versch. Bindungen



abstrakte Schnittstelle

- Beschreibung der Schnittstelle unabhängig von
  - Nachrichtenformaten wie SOAP
  - Übertragungsprotokollen wie HTTP

Bindung

- Realisierung einer abstrakten Schnittstelle mit bestimmtem Nachrichtenformat und Übertragungsprotokoll

Beispiel Google™ (fiktiv)

- ein Dienst (**abstrakte Schnittstelle**):
  - Name der Operation: doGoogleSearch
  - Eingangsparameter: key:string, q:string, ...
  - Rückgabewert: doGoogleSearchResponse
    - Kind-Elemente von *doGoogleSearchResponse*: Rückgabewerte (komplexer Datentyp)

- eine Beschreibung (**WSDL**), aber 4 Zugriffs-möglichkeiten (**Bindungen**):
  1. SOAP/HTTP-POST
  2. SOAP/HTTP-GET (REST)
  3. SOAP/SMTP (asynchron)
  4. HTML/HTTP-GET (Browser/REST)

SOAP/HTTP-GET?


```
<env:Envelope xmlns:env=http://www.w3.org/2001/12/soap-envelope>
        <env:Body>
                <r:GetLastTradePrice
                  env:encodingStyle=http://www.w3.org/2001/12/soap-encoding
                  xmlns:r=http://example.org/2001/06/quotes>
                        <r:Symbol>DEF</r:Symbol>
        </r:GetLastTradePrice>
        </env:Body>
</env:Envelope>
```

GET example.org/stockService/LastTradingPrice&xmlnsuri=http://example.org/2001/06/quotes&xmlns=r&encodingStyle=http://www.w3.org/2001/12/soap-encoding&Symbole=DEF

SOAP/HTTP-GET?

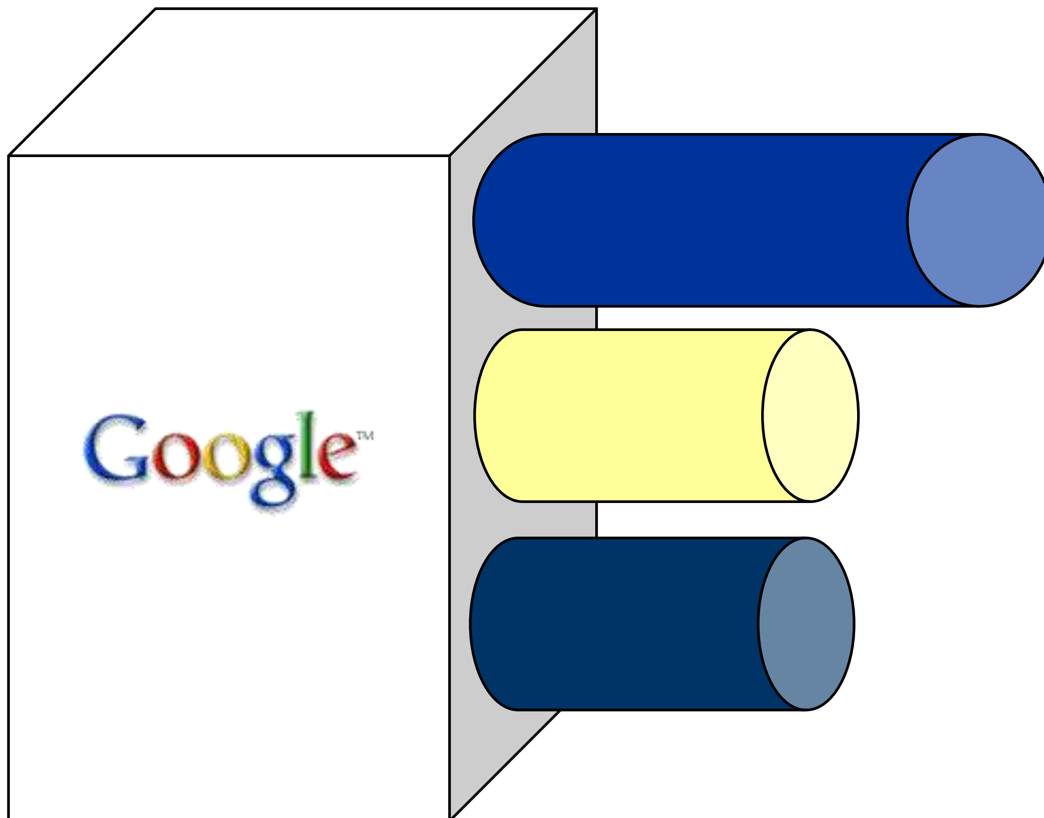
Freie Universität Berlin

Request

```
GET
/travelcompany.example.org/reservations?code=FT35ZBQ
HTTP/1.1
Host: travelcompany.example.org
Accept: text/html;q=0.5, application/soap+xml
```

Response

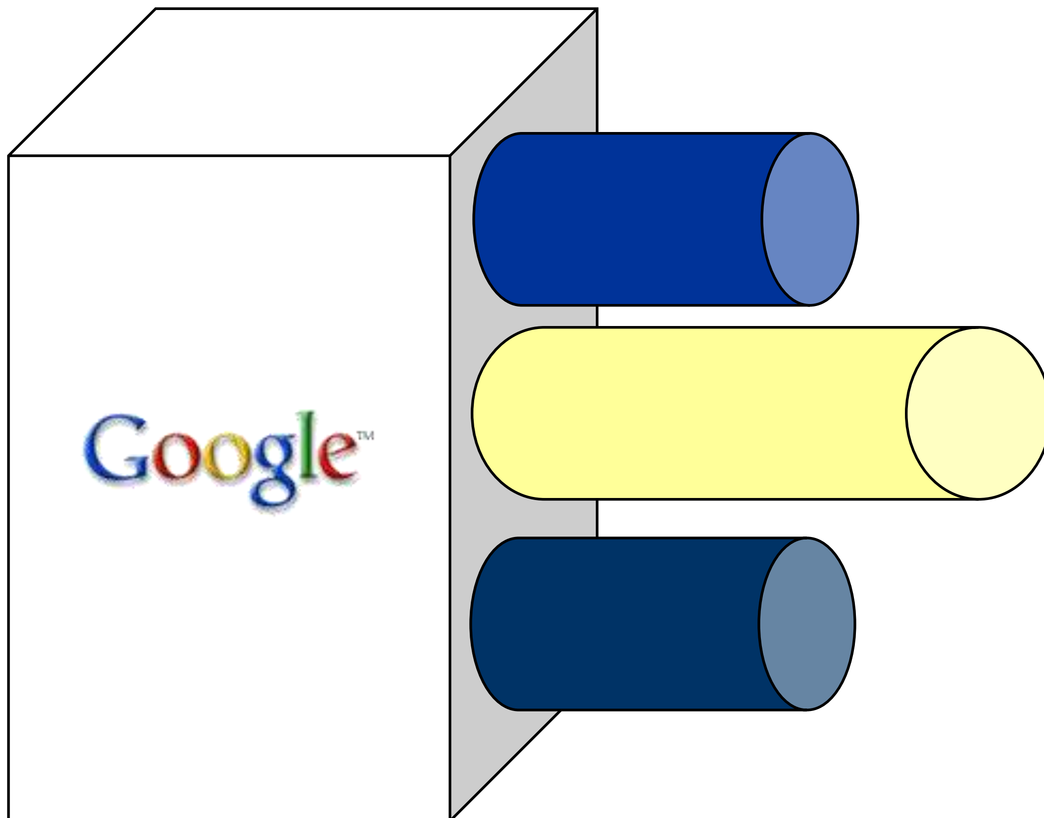
```
HTTP/1.1 200 OK
Content-Type: application/soap+xml;
charset="utf-8„
Content-Length: nnnn

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
…
</env:Envelope>
```



Dienst: Suche

Name der Operation: doGoogleSearch

Rückgabe: doGoogleSearchResponse

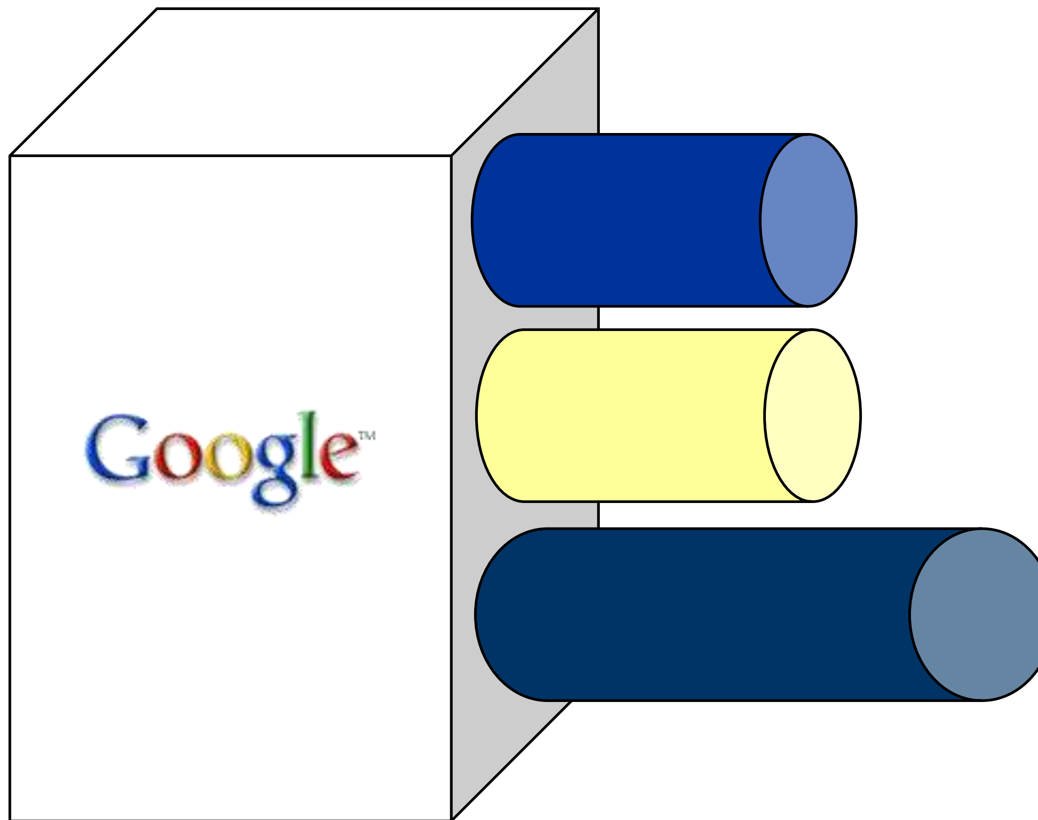


Dienst:
Zugriff auf Web-Cache

Name der Operation:
doGetCachedPage

Rückgabe:
doGetCachedPageResponse

Web Service von Google



Dienst:
Rechtschreibkorrektur

Name der Operation:
doSpellingSuggestion

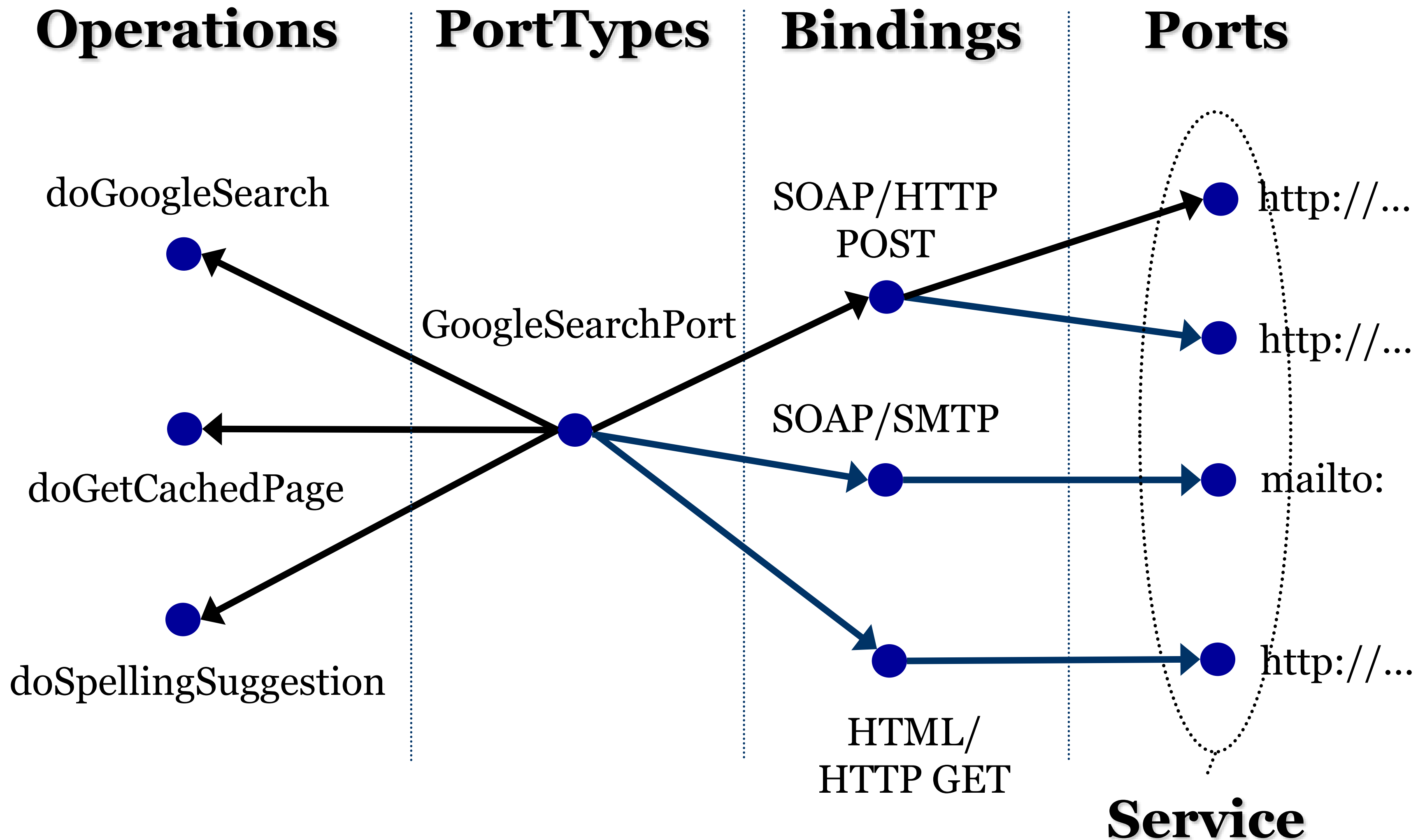
Rückgabe:
doSpellingSuggestionResponse

WSDL-Inhalt

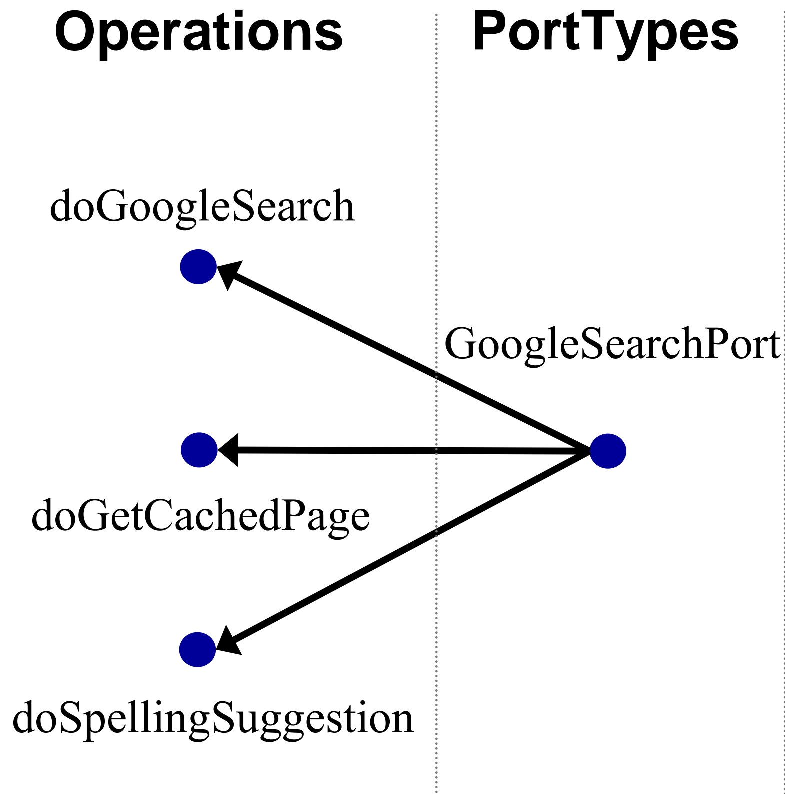
Freie Universität Berlin

- Was? → Typen, Messages, PortTypes (Interfaces)
  - Deklaration der verfügbaren Operationen
  - Struktur der ausgetauschten Nachrichten (Aufruf und Rückruf, Fehlermeldungen)
- Wie → Bindings
  - unterstützte Transportprotokolle
  - verwendete Nachrichtenformate
- Wo → Service
  - Wie heißt der Service?
  - Unter welchen URLs kann er gefunden werden?

Grundstruktur

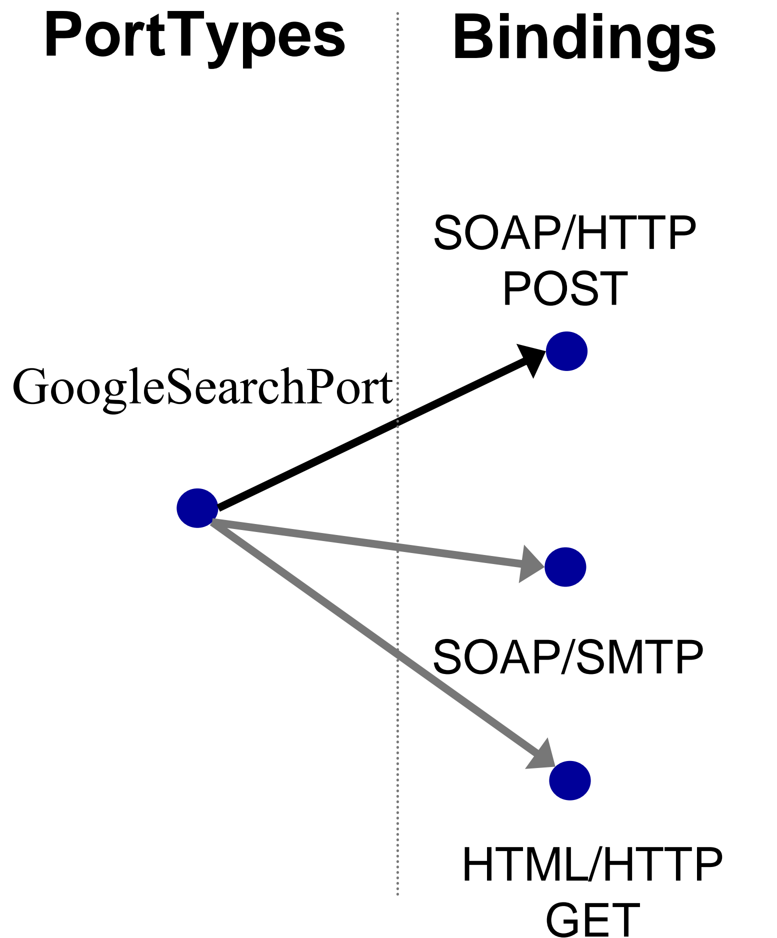


Abstrakte Schnittstellen



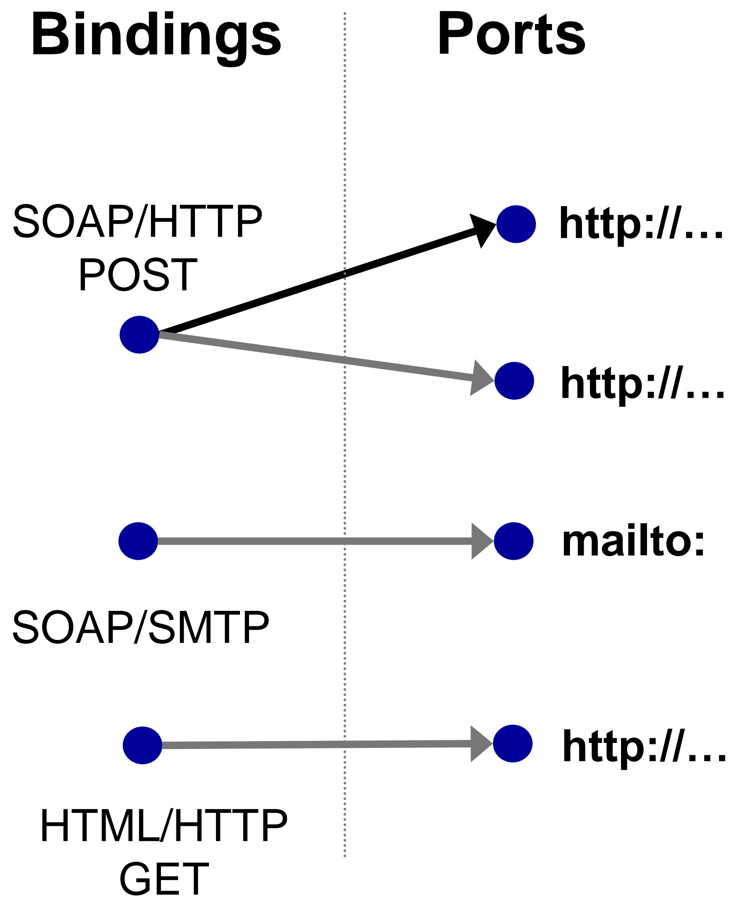
- **portType** (WSDL 1.1) **=** **interface** (WSDL 2.0)
- portType = Menge von abstrakten Operationen
- jede abstrakte Operation beschreibt Eingangs- und Ausgangsnachricht
- meist nur ein portType, aber in WSDL 1.1 auch mehrere möglich

Bindungen



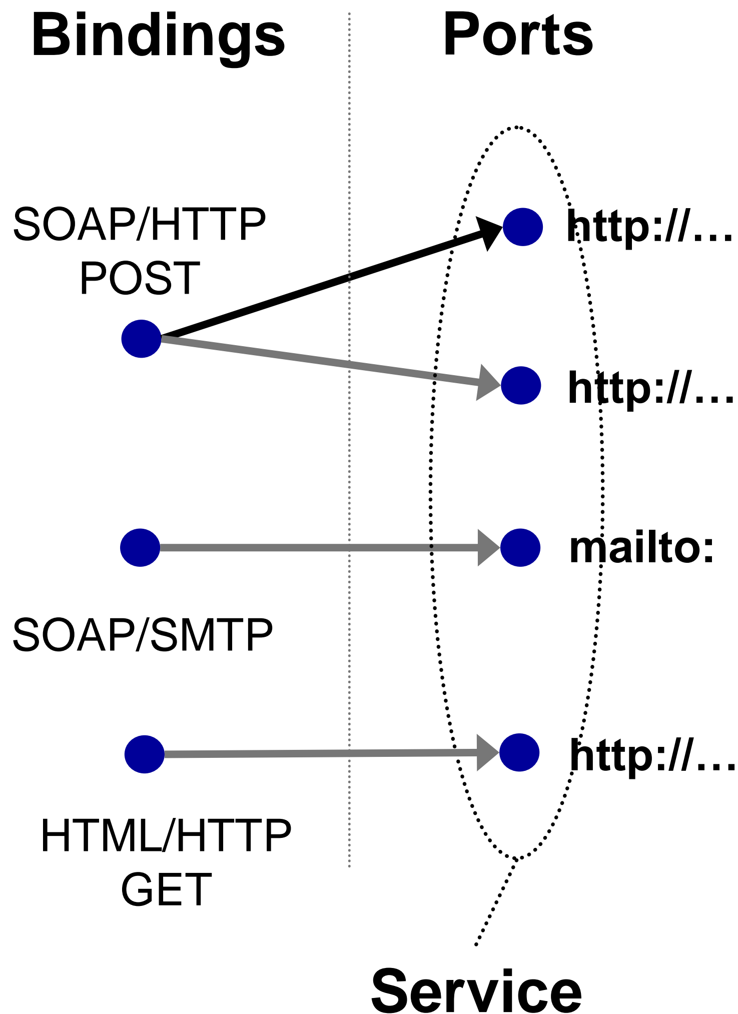
- in WSDL **binding** genannt
- für jede abstrakte Schnittstelle (**portType**) mindestens eine Bindung
- ein **portType** kann also mit **unterschiedlichen Bindungen** realisiert sein

Kommunikationsendpunkte



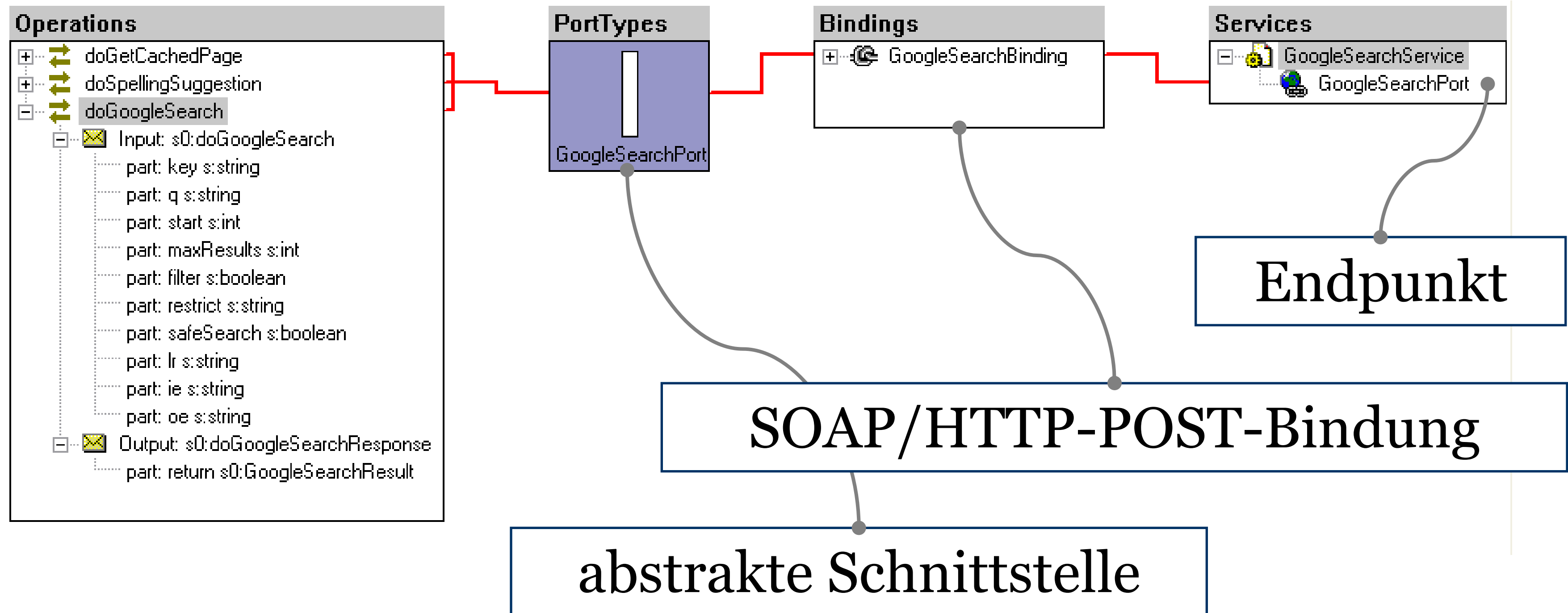
- **port** (WSDL 1.1) **=** **endpoint** (WSDL 2.0)
- **port** = Bindung + Web-Adresse
- für jede Bindung (**binding**) mindestens ein **port**
- ein **binding** kann also über unterschiedliche Web-Adressen zugänglich sein

Service



- Menge von **port**s bilden zusammen einen **Service**
- **port**s können in verschiedene Services gruppiert werden
- **port**s eines Service = semantisch äquivalente Alternativen

Die WSDL-Beschreibung von Google



WSDL 1.1. – Elemente (1)

Freie Universität Berlin

Element	Beschreibung
Abstrakte Beschreibung	
`<types> … </types>`	- Maschinen- und sprachunabhängige Typdefinitionen → definiert die verwendeten **Datentypen**
`<message> … </message>`	- **Nachrichten**, die übertragen werden sollen - Funktionsparameter (Trennung zwischen Ein- und Ausgabeparameter) oder Dokumentbeschreibungen
`<portType> … </portType>`	- Nachrichtendefinitionen im Messages-Abschnitt - definiert **Operationen**, die beim Web Service ausgeführt werden

WSDL 1.1. – Elemente (2)

Element	Beschreibung
Konkrete Beschreibung	
`<binding>…</binding>`	- Kommunikationsprotokoll, das beim Web Service benutzt wird - Gibt die Bindung(en) der einzelnen Operationen im portType-Abschnitt an
`<service>…</service>`	- gibt die Anschlussadresse(n) der einzelnen Bindungen an (Sammlung von einem oder mehreren Ports)

WSDL-Beispiel

```
<definitions name="HelloService" targetNamespace="http://www.examples.com/wsdl/HelloService.wsdl" …>
        <message name="SayHelloRequest"> <part name="firstName"  type="xsd:string"/> </message>
        <message name="SayHelloResponse"> <part name="greeting"  type="xsd:string"/> </message>

 <portType name="Hello_PortType">
        <operation name="sayHello">
                <input message="tns:SayHelloRequest"/>
                <output message="tns:SayHelloResponse"/>
        </operation>
</portType>

<binding name="Hello_Binding" type="tns:Hello_PortType">
        <soap:binding style="rpc"  transport=http://schemas.xmlsoap.org/soap/http />
        <operation name="sayHello">
                 <soap:operation soapAction="sayHello" />
                <input> <soap:body  encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
                        namespace="urn:examples:helloservice" use="encoded"/> </input>
                <output><soap:body encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
                        namespace="urn:examples:helloservice" use="encoded"/> </output>
        </operation>
</binding>
<service name="Hello_Service">
        <documentation>WSDL File for HelloService</documentation>
        <port binding="tns:Hello_Binding" name="Hello_Port">
                <soap:address location="http://www.examples.com/SayHello/">
        </port>
</service>
</definitions>
```

an abstract, typed definition of the data being communicated (message)

an abstract set of operations supported by one or more endpoints (portType)

an abstract description of an action supported by the service (operation)

a concrete protocol and data format specification for a particular port type (binding)

a collection of related endpoints (service)

a single endpoint defined as a combination of a binding and a network address (port)

Beispiesource von http://www.tutorialspoint.com/wsdl/wsdl_example.htm

Eigenschaften von WSDL

- baut auf **XML-Schema** auf

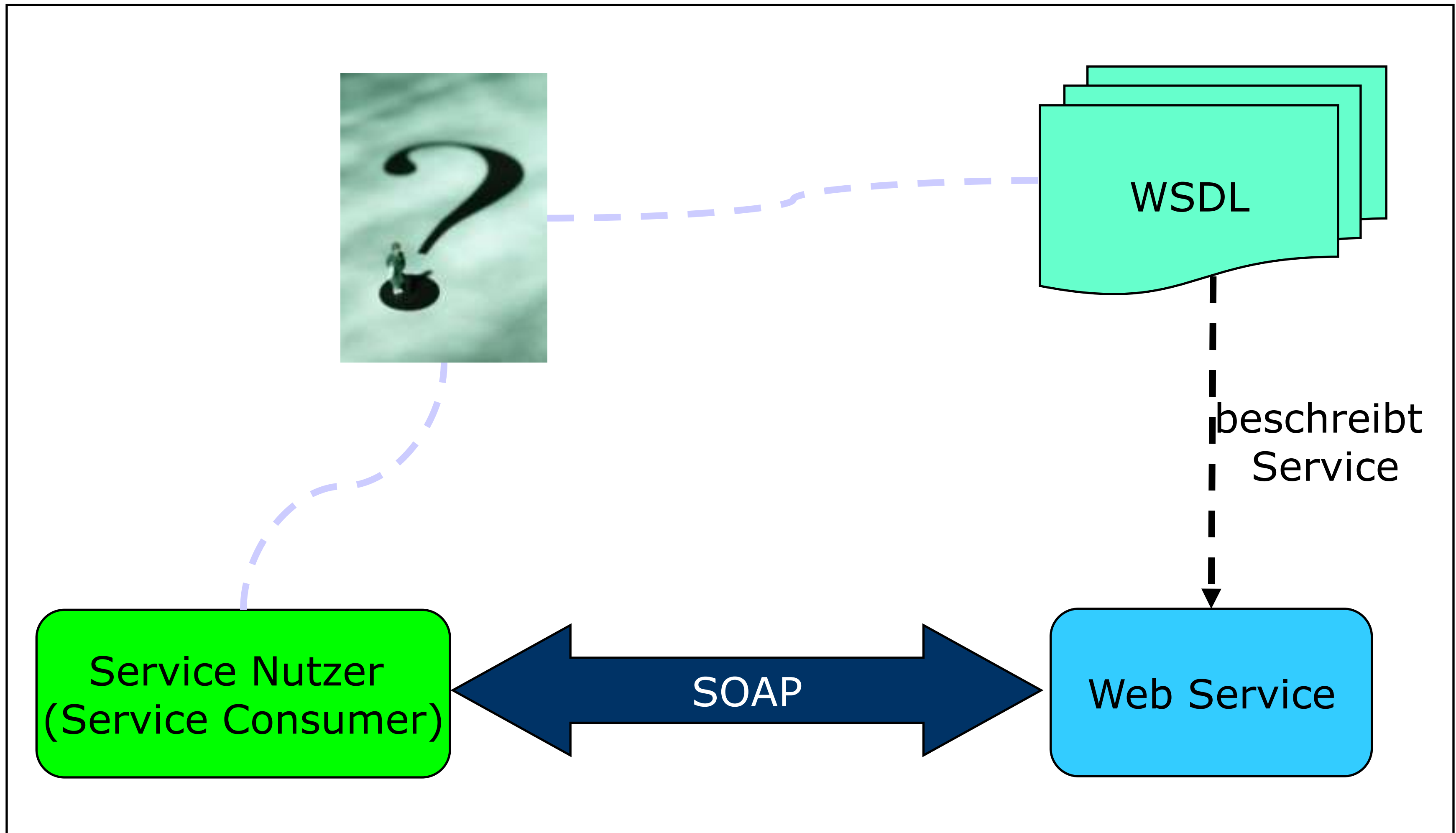
⇨ Syntax einer Schnittstelle kann bis ins kleinste Detail festgelegt werden

- beschreibt **Schnittstelle(n**) eines Web Services und **wo** dieser **abgerufen** werden kann

- **grundlegende Interaktionsmuster** (wie Anfrage-Antwort)

- keine Möglichkeit semantische Eigenschaften zu beschreiben

Web Services



Vorteile von WSDL

- **Ziel: Interoperabilität**
  → Interoperabilität zwischen unterschiedlichen Implementierungsplattformen

Vorteile

+ Plattformunabhängig
+ Syntax der Schnittstelle kann genau festgelegt werden
+ Unterschiedliche Realisierungen einer abstrakter Schnittstelle möglich (z.B. SOAP über HTTP und SMTP)

Nachteile von WSDL

schafft neue Probleme

- nicht alle Entwicklungen werden akzeptiert (vgl. UDDI, REST vs. SOAP)
- nicht alle geforderte Funktionalitäten sind verfügbar (Sicherheit, Transaktionen, Schnittstellenversionierung, etc.)
- eine weitere Schnittstellentechnologie, die gewartet werden muss

Nachteile

- verschiedene Protokoll-Bindungen (wie HTTP vs. SMTP) können unterschiedliche Semantik haben
- keine komplexen Interaktionsmuster
- keine qualitativen Aspekte (quality of service)
- keine Sicherheitsaspekte
- unzureichend, um automatisch die Kompatibilität (Interoperabilität) zweier Web Services feststellen zu können → Semantic Web Services

Web Service Basiskomponenten: *UDDI*

UDDI

- Universal (Service) Description, Discovery, and Integration
- entwickelt seit Herbst 2000 von Ariba, Microsoft & IBM
- beschreibt einen Verzeichnisdienst für Web Services
- SOAP basierter Dienst

UDDI

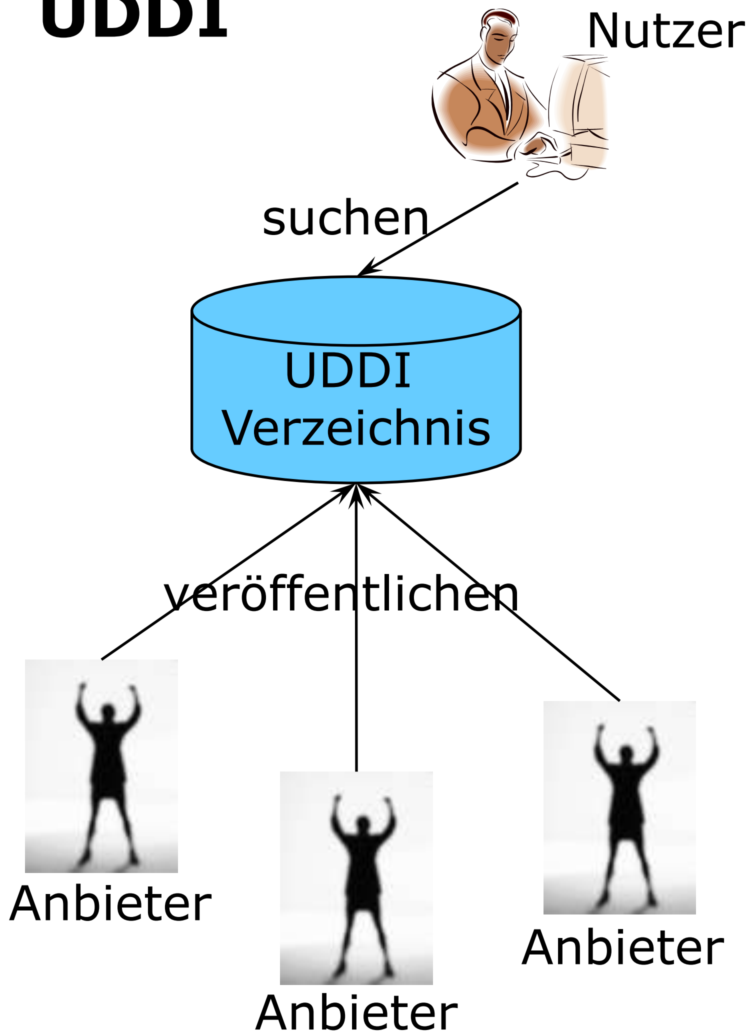
- White Pages
- Yellow Pages
- Green Pages

Quelle: http://www.tecchannel.de/webtechnik/soa/472223/index4.html

UDDI Komponenten

- UDDI-XML-Schema
  - Business-Entity
  - Business-Service
  - Binding-Template
  - Technische Modelle (TModel)
- UDDI-API
  - Anwendungsschnittstelle in Form von Web Services

UDDI vs. WS-Inspection



WS-Inspection

Nutzer

suchen

suchen

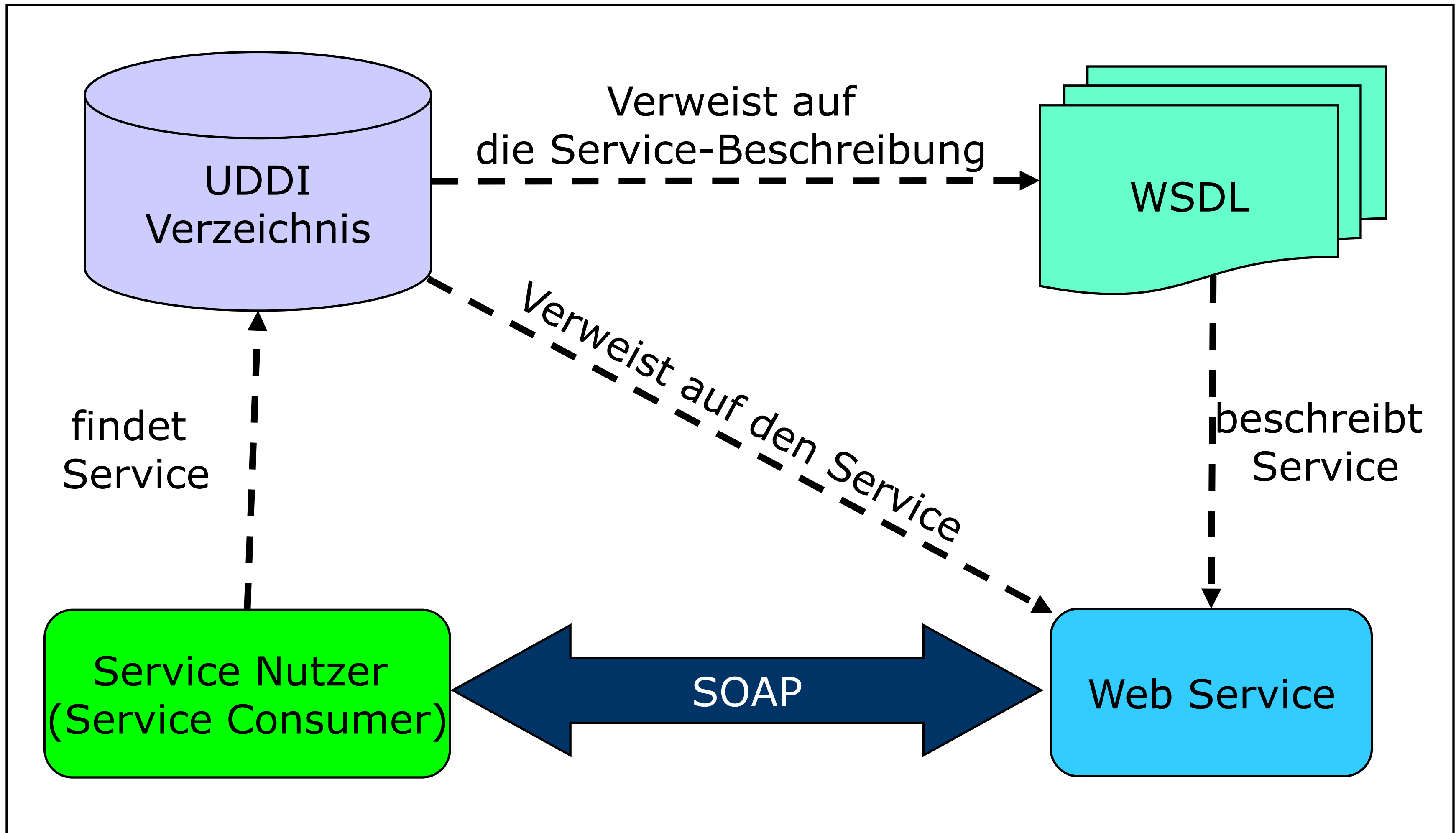
suchen

Anbieter

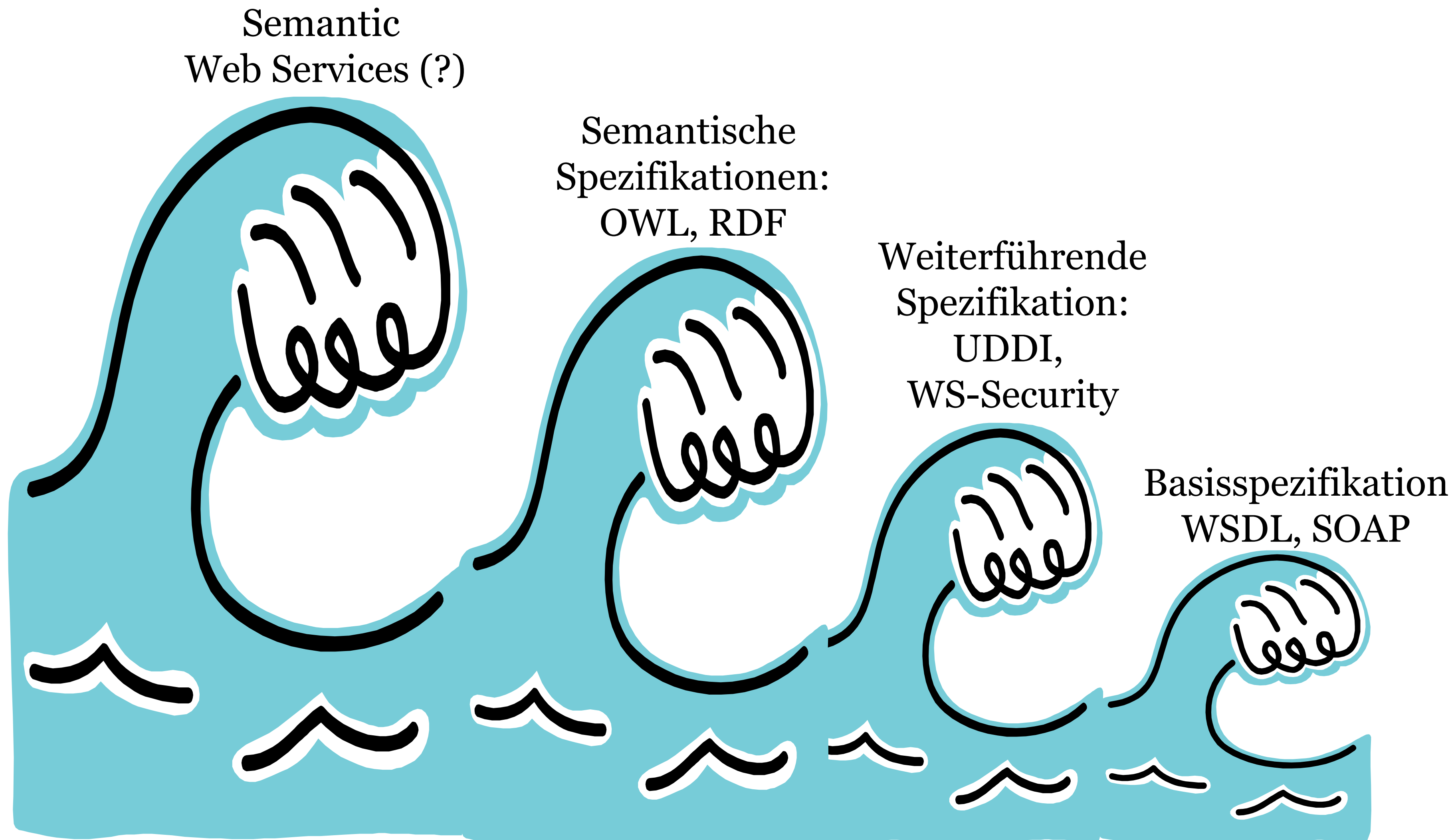
Anbieter

Anbieter

Web Services



Web Services Wellen



RPC vs. Messaging

Freie Universität Berlin

Wie SOAP-Nachrichten übertragen?

über HTTP

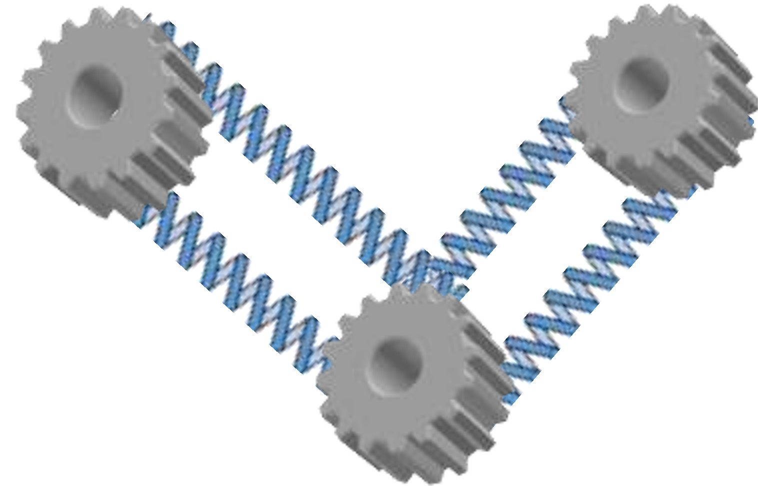
- heute üblich
- Request-Response-Verhalten von HTTP unterstützt **RPC**s
- **verbindungsorientiert**
- Übertragung: Sender und Empfänger müssen präsent sein.
- typischerweise **synchron**

über SMTP

- heute eher selten
- realisiert **Messaging**
- **persistente** Kommunikation
- Übertragung: Weder Sender noch Empfänger muss präsent sein.
- typischerweise **asynchron**
- erlaubt Lastverteilung und Priorisierung

⇨ weitreichende Designentscheidung!

Weitreichende Designentscheidung



enge Kopplung

- **verbindungs-orientierte**, **synchrone** Kommunikation
- z.B. SOAP/HTTP

lose Kopplung

- **gepufferte**, **asynchrone** Kommunikation
- z.B. SOAP/SMTP
- robuster, aber auch komplexer zu entwerfen

RPC mit SOAP

- Nachrichten konform:
  - zu einer vordefinierten Beschreibung des Funktionsaufrufes
  - zu der zu erwartenden Rückantwort
- Arten der Kommunikation:
  - Anfrage (Request)
  - Antwort (Response)
  - Fehlerfall (Fault)

asynchroner Nachrichtenaustausch

- auch nach Timeout Antwort noch möglich
- Was tun mit der Antwort?
- häufig muss Absender informiert werden, dass Antwort nicht verarbeitet werden konnte
- Was tun, wenn diese Warnung nicht rechtzeitig beim Absender ankommt?
- Absender hat vielleicht im falschen Glauben, dass seine Antwort verarbeitet wurde, weitergearbeitet.
- Dieser Vorgang muss dann evtl. rückgängig gemacht werden.

Messaging

- Anwendungen interagieren durch Austausch von Nachrichten miteinander:
- Kommunikation in Anwendung sichtbar: senden und empfangen
- verschiedene Formen des Messaging:
  1. Kommunikationsstruktur
  2. Interaktionsmuster
  3. flüchtig vs. persistent
  4. synchron vs. asynchron

1. Kommunikationsstruktur

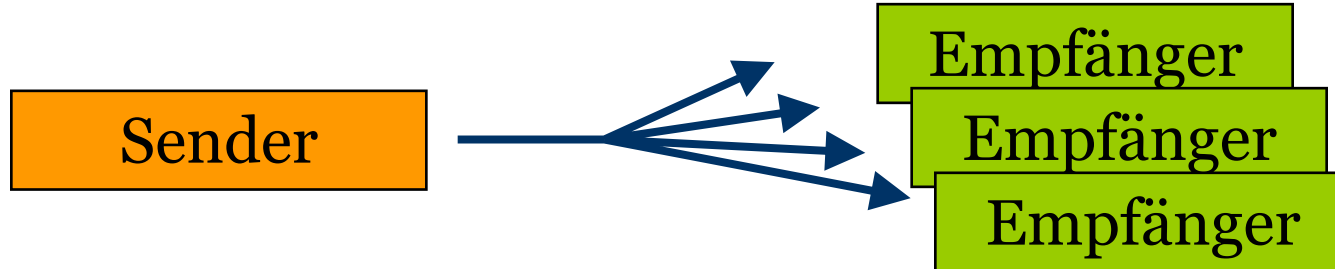
- Anzahl und Organisation der Kommunikationspartner

- wichtigste Kommunikationsstrukturen:
  - One-to-One-Kommunikation
  - One-to-Many-Kommunikation

1. Kommunikationsstruktur: One-to-One-Kommunikation

Freie Universität Berlin

- Sender sendet Nachricht an bestimmten Empfänger.
- Beispiel: Kunde sendet Bestellung per E-Mail an Firma



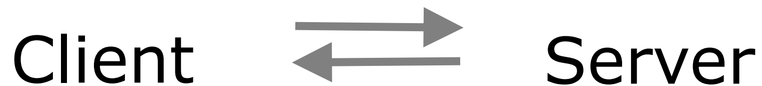
- Sender sendet identische Kopie gleichzeitig an mehrere Empfänger
- Sender (publisher) veröffentlicht Nachricht zu bestimmten Thema (topic), zu dem sich die Empfänger (subscriber) angemeldet haben.

⇨ auch **publish-subscribe** oder **topic-based messaging** genannt

- Beispiel: Mailing-Liste

2. Interaktionsmuster

Einweg (one way, fire and forget)



Anfrage-Antwort (request-response)

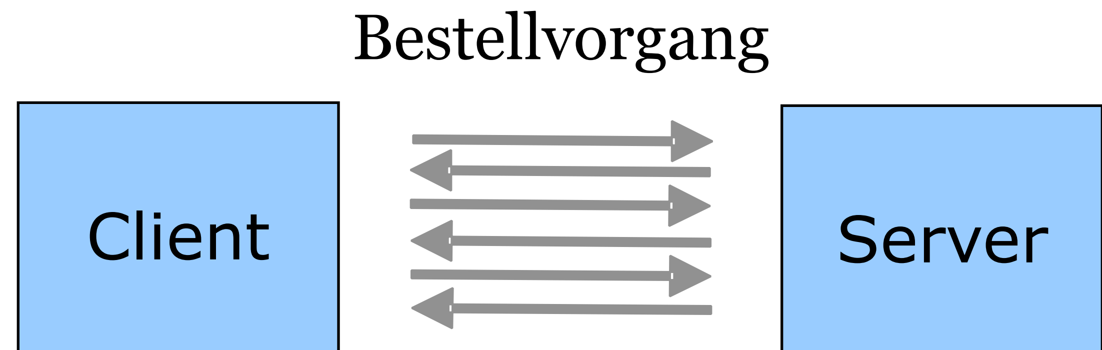
Benachrichtigung (notification)

Benachrichtigung-Antwort (notification-response)

Beachte: „Antwort“ bezieht sich auf jeweilige Anwendung, nicht auf das Netzwerk.

2. Interaktionsmuster: Komplexe Interaktionsmuster

- → Bestellanfrage mit spätestem Liefertermin
- ← Angebot mit zugesichertem Liefertermin
- → Bestellung
- ← Bestätigung des Eingangs der Bestellung
- → Bestätigung der Lieferung
- ← Rechnung

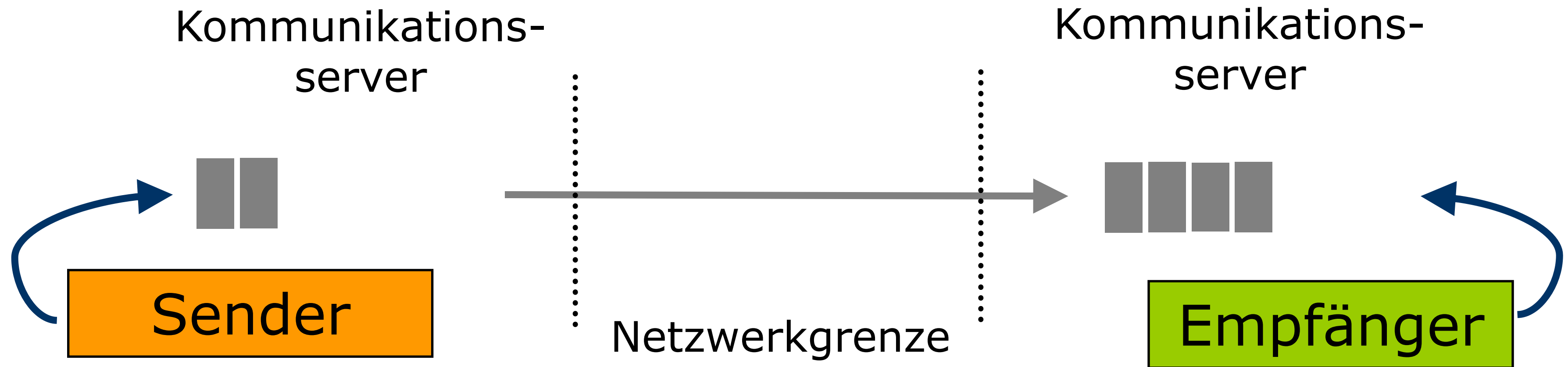


3. **Flüchtig** vs. Persistent

flüchtige Kommunikation

- Sender und Empfänger kommunizieren direkt ohne Puffer miteinander.
- Sender und Empfänger müssen während der gesamten Übertragung präsent sein
- engl. transient
- Beispiele: HTTP, UDP

Flüchtig vs. **Persistent**



persistente Kommunikation

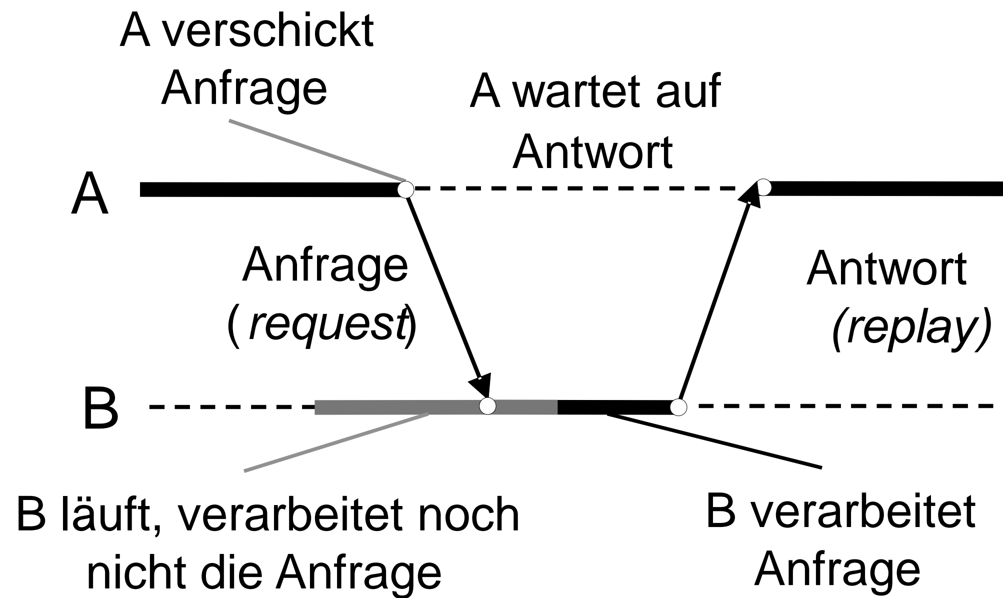
- Nachricht solange gespeichert, bis sie tatsächlich zugestellt wurde.
- Weder Sender noch Empfänger müssen während Übertragung präsent sein.
- Beispiel: E-Mail

4. Synchron vs. Asynchron

- **Asynchron**
  - Senden und Empfangen zeitlich versetzt
  - ohne Blockieren des Prozesses (ohne Warten auf die Antwort)
- **Synchron**
  - Senden/Empfangen synchronisieren → warten (blockieren) bis die Kommunikation abgeschlossen ist.

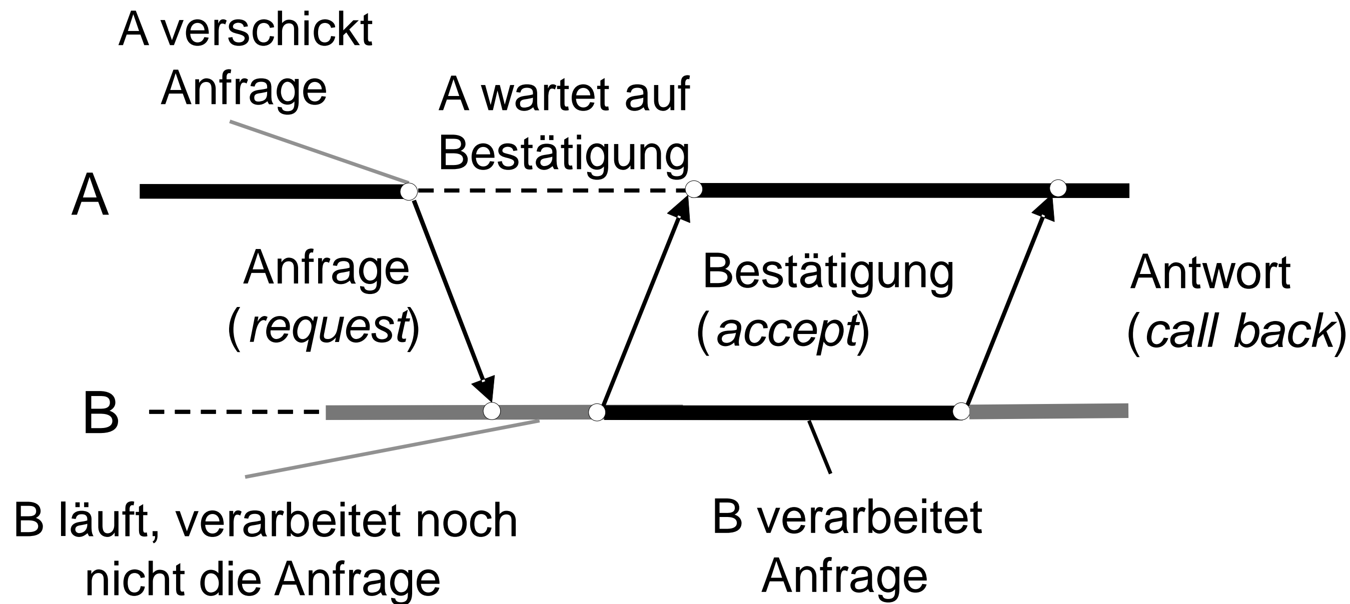
- bei **persistenter Kommunikation** → Messaging normalerweise asynchron
- bei **flüchtiger Kommunikation** → Messaging kann sowohl synchron als auch asynchron sein

Beispiel 1



- **flüchtige**, **synchrone** Kommunikation
- auch **antwortorientiert** (response-based) genannt
- implementiert synchrone RPCs
- jedoch ≠ RPC: Kommunikation für Anwendung sichtbar

Beispiel 2



- **flüchtige**, **asynchrone** Kommunikation
- auch **bestätigungsorientiert** (delivery-based) genannt
- implementiert asynchrone RPCs
- jedoch ≠ RPC: Kommunikation für Anwendung sichtbar

RPC oder Messaging?

RPC ⇨ **eng gekoppelte, starre Systeme**

- \+ einfach, abstrahiert von Kommunikation
- \- nur Eins-zu-Eins-Kommunikation
- \- Client und Server müssen präsent sein
- \- skaliert weniger gut

Messaging ⇨ **lose gekoppelte, robuste Systeme**

- \- abstrahiert nicht von Kommunikation
- \+ erlaubt auch One-to-Many-Kommunikation
- \+ Weder Sender noch Empfänger müssen präsent sein
- \+ erlaubt Priorisierung und Lastverteilung
- \+ skaliert sehr gut

heutige Vorlesung

- ☑ Was sind Web Services?
- ☑ Basistechnologien (SOAP, WSDL, UDDI)
- ☑ RPC vs. Messaging