Freie Universität Berlin

WSDL

Block Web Services

Freie Universität Berlin

Vorlesungs-termin	Vorlesung 4 + 1 + 1 Termine	Übung – 2 Termine	Übungs-termin
06.06.	Web Services, SOA, RPCs vs. Messaging		
20.06.	SOAP im Detail	SOAP	25./26.06
27.06. (heute)	WSDL im Detail	WSDL	02./03.07
04.07.	Web Services in der Praxis & Ausblick		
11.07.	Rückblick + (14:00-16:00) Sprechstunde vor der Klausur, Fabeckstr. 15		
18.07.	Klausur		

Heutige Vorlesung

Freie Universität Berlin

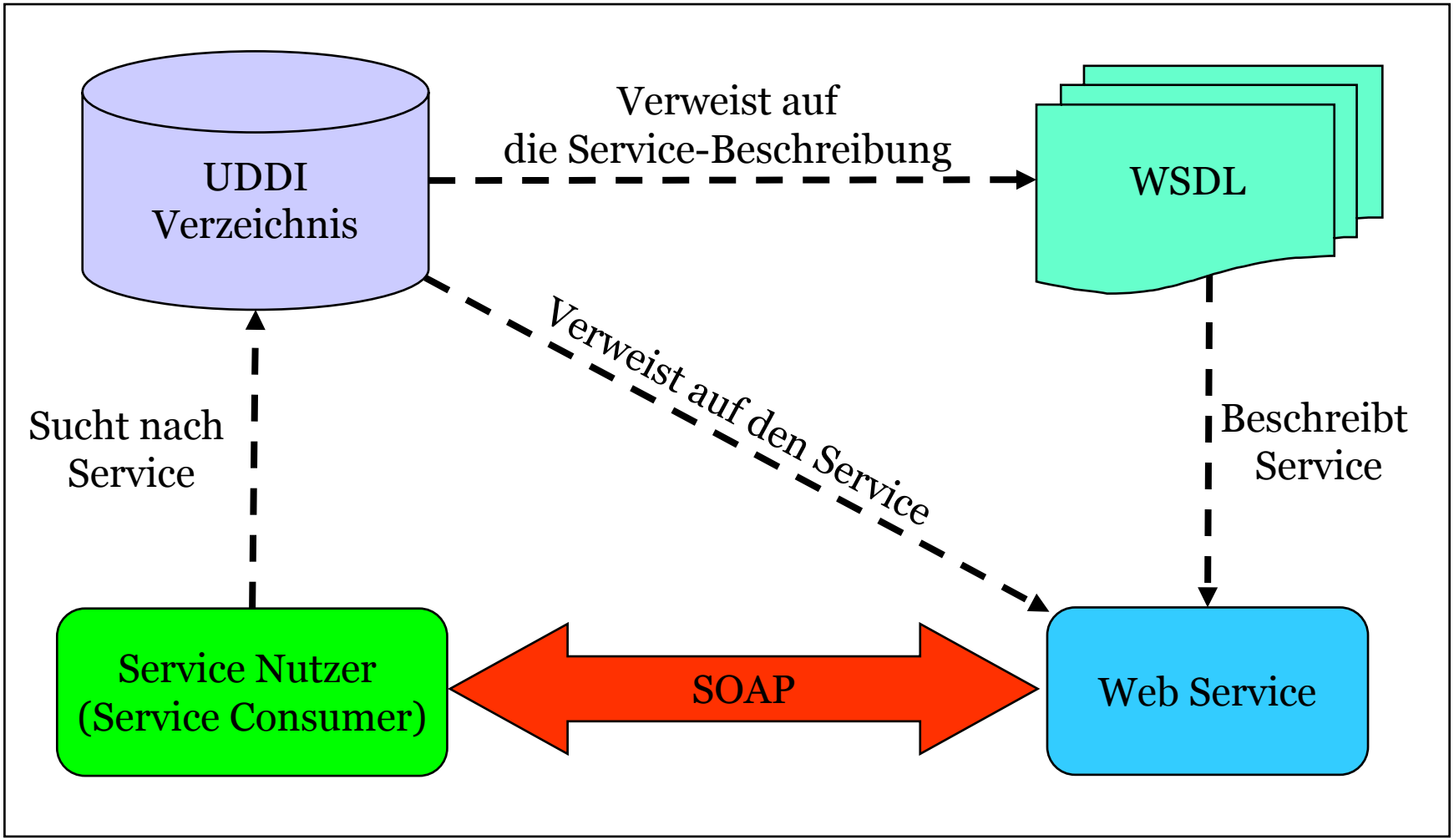
letzte Woche

- ☑ prinzipieller Aufbau
- ☑ Kodierung von RPCs
- ☑ Verarbeitung & Übertragung
- ☑ Vor- und Nachteile

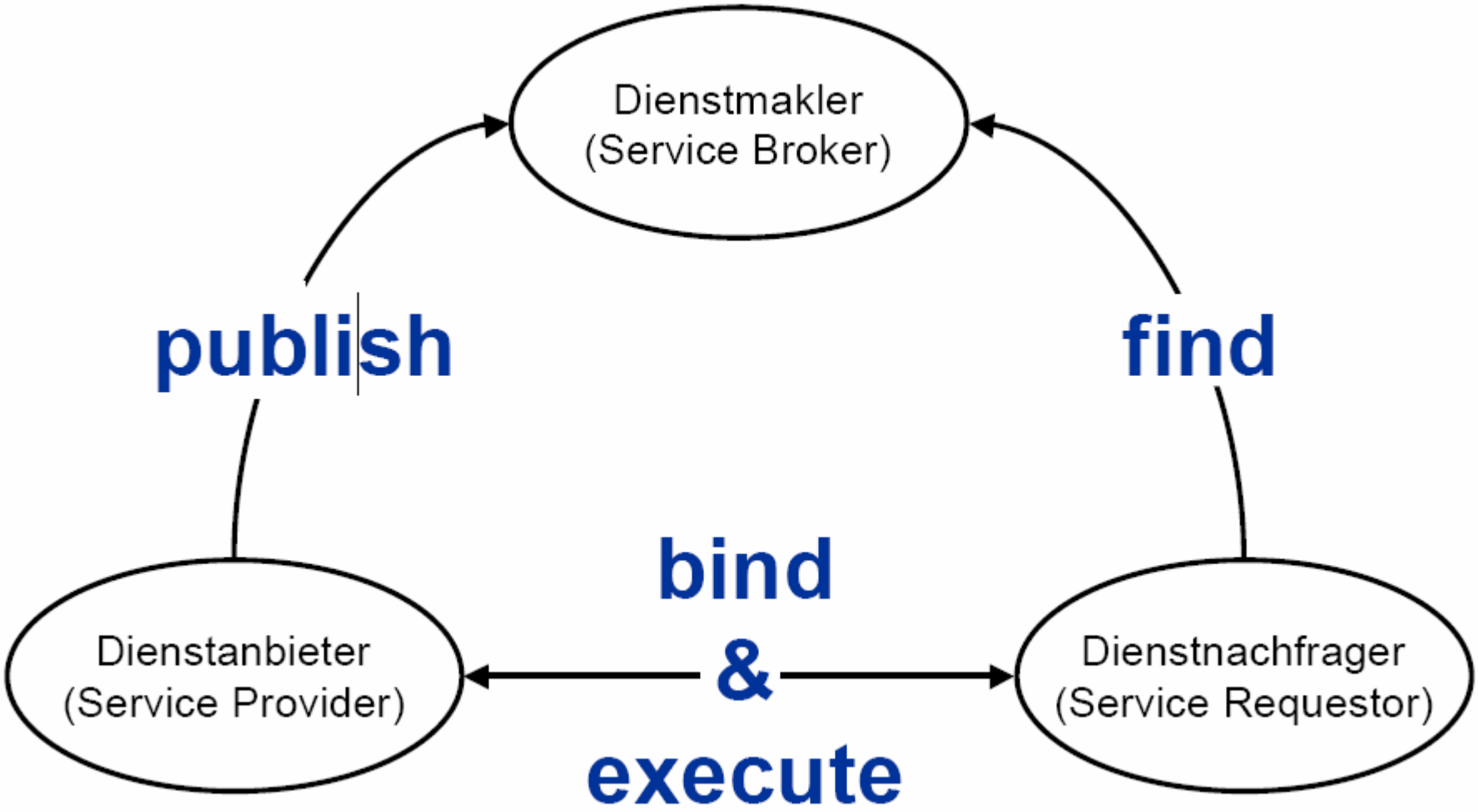
heutige Vorlesung → WSDL

- Wozu WSDL-Syntax verstehen?
- prinzipieller Aufbau
- standardisierte Bindungen (SOAP- & HTTP-Bindung)
- Vor- und Nachteile

Web Services

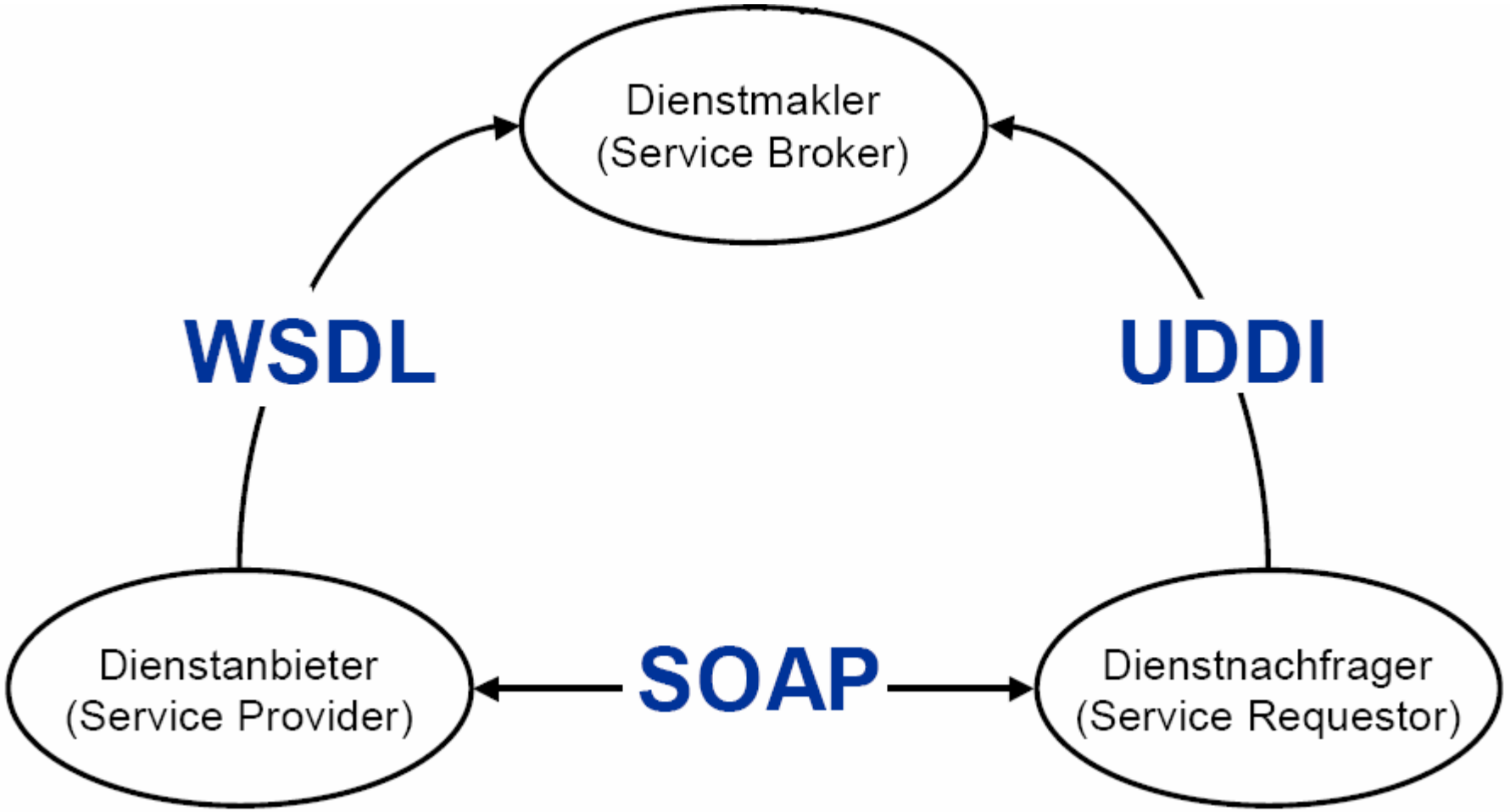


Web Services (2)



Quelle: http://www.jeckle.de/files/WSDL2002.pdf

Web Services (3)



Quelle: http://www.jeckle.de/files/WSDL2002.pdf

WSDL = Web Services Description Language

- baut auf XML-Schema auf
- stellt ein XML-Vokabular zur Beschreibung von Web Services (Schnittstellen, Operationen und Dienste) dar

  → Standard für die Beschreibung dessen, was zwischen Konsument und Anbieter geschickt wird

  → Syntax einer Schnittstelle kann bis ins kleinste Detail festgelegt werden

  → Beschreibung von Grundlegende Interaktionsmuster (wie Anfrage-Antwort)

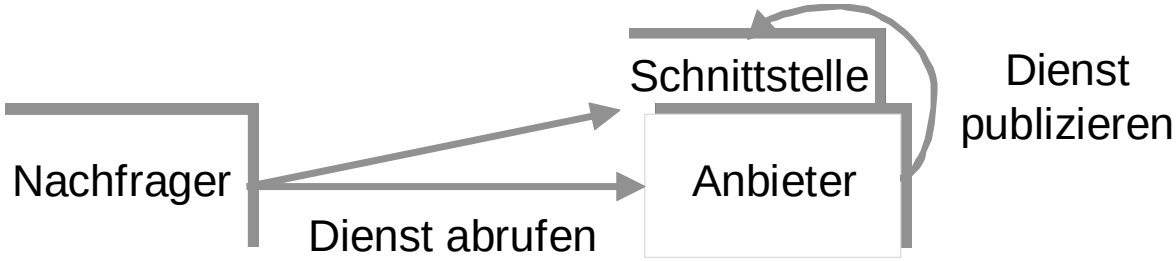
- noch KEIN W3C Standard

WSDL bei W3C

Freie Universität Berlin

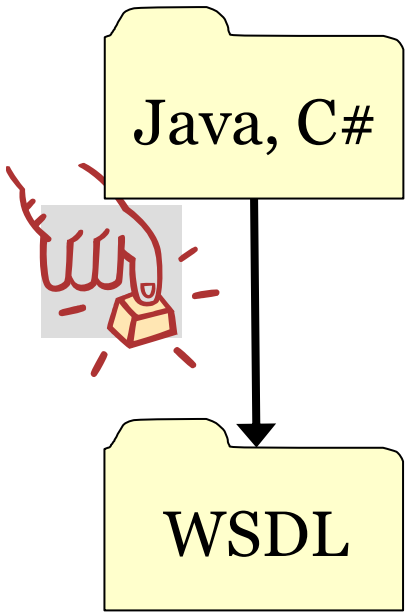
- Web Services Description Working Group → http://www.w3.org/2002/ws/desc/
- WSDL 1.1. → W3C Note, März 2001
- WSDL Version 1.2 → W3C Working Draft, Juni 2003
  - Part 1: Core Language
  - Part 2: Message Patterns
  - Part 3: Bindings
- WSDL Version 2.0 → W3C Working Draft, März 2004
  - Part 1: Core Language
  - Part 2: Message Exchange Patterns
- WSDL Version 2.0 → W3C Proposed Recommendation, März 2007
  - Part 0: Primer
- WSDL Version 2.0 → W3C Proposed Recommendation, Mai 2007
  - Part 1 : Core Language

Wozu dient WSDL?



- 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
- ähnlich wie Java-IDL, jedoch wesentlich allgemeiner

Wozu WSDL-Syntax verstehen?



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

⇨ Web-Service-Anbieter sollte WSDL (Vertrag) verstehen!

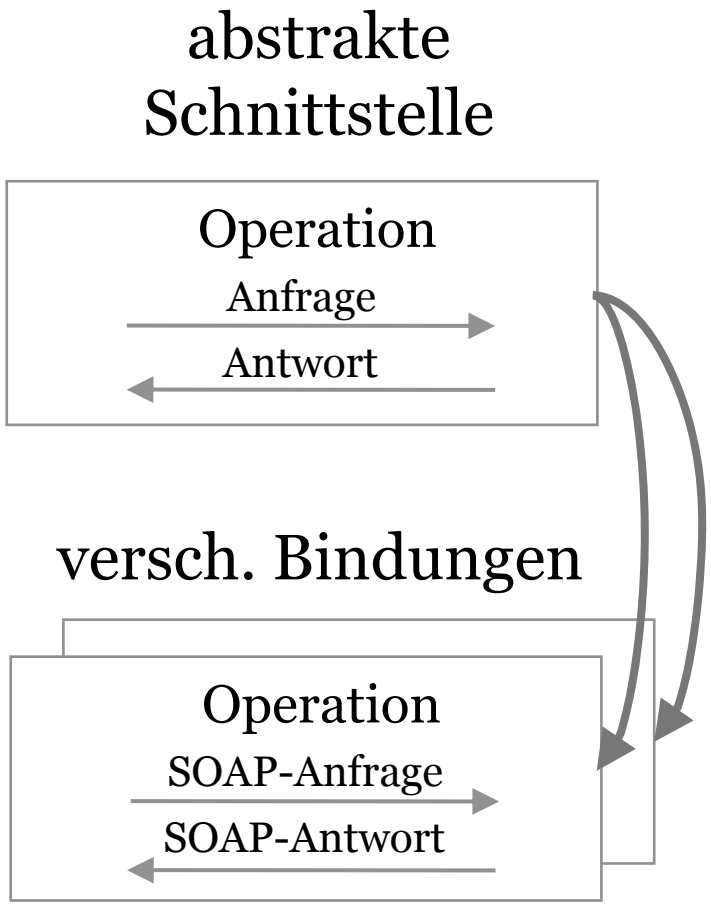
- mögliche Probleme bei generierten WSDLs:
  - Fehlermeldungen nicht korrekt beschrieben
  - optionale RPC-Parameter ungünstig beschrieben

Prinzipieller Aufbau

WSDL Aufbau

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

Grundidee



abstrakte Schnittstelle

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

Bindung

- Realisierung einer abstrakten Schnittstelle mit bestimmtem Nachrichtenformat und Übertragungsprotokoll

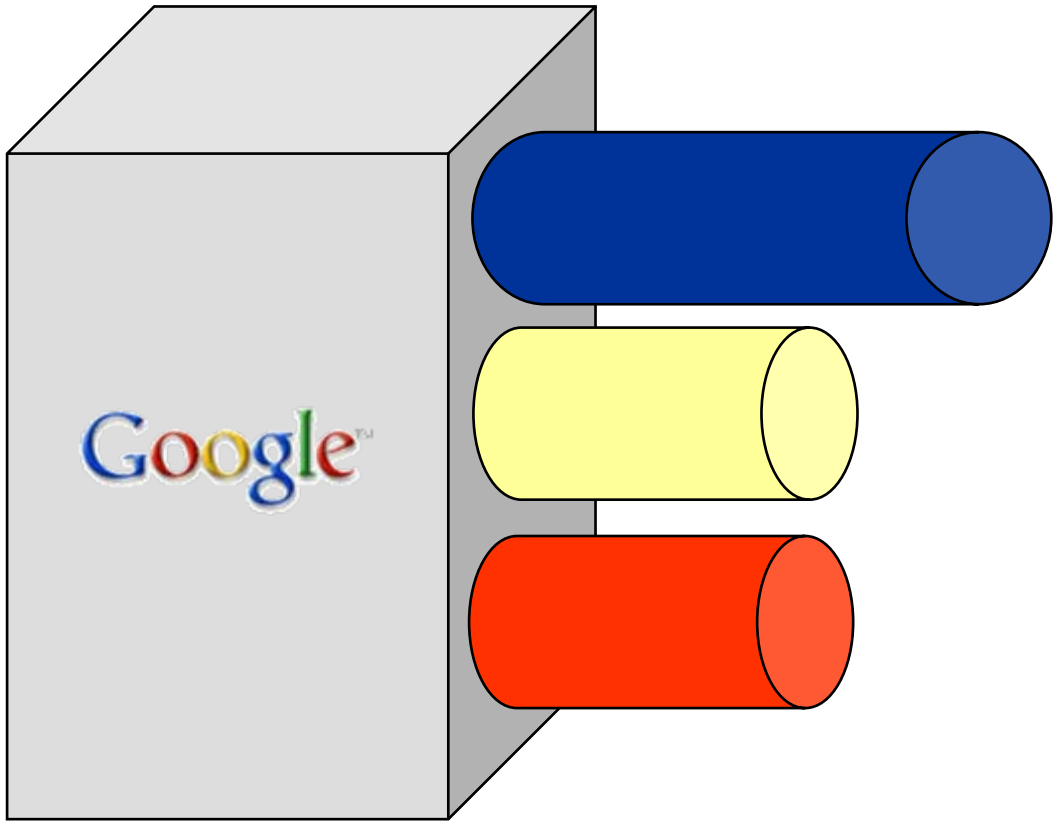
Beispiel Google (fiktiv)

Freie Universität Berlin

- 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 Zugriffmöglichkeiten (**Bindungen**):
  1. SOAP/HTTP-POST
  2. SOAP/HTTP-GET (Rest)
  3. SOAP/SMTP (asynchron)
  4. HTML/HTTP-GET (Browser)

Web Service von Google

Freie Universität Berlin



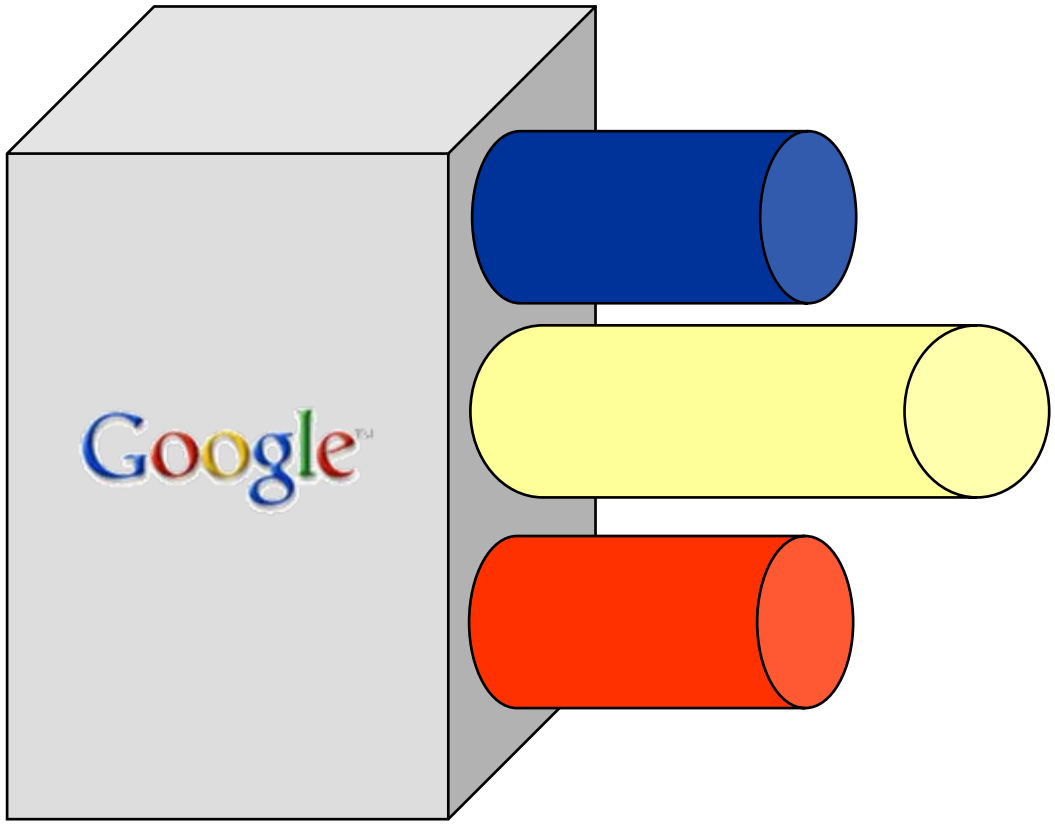
Dienst: Suche

Name der Operation:
doGoogleSearch

Rückgabe:
doGoogleSearchResponse

Web Service von Google

Freie Universität Berlin



Dienst:

Zugriff auf Web-Cache

Name der Operation:

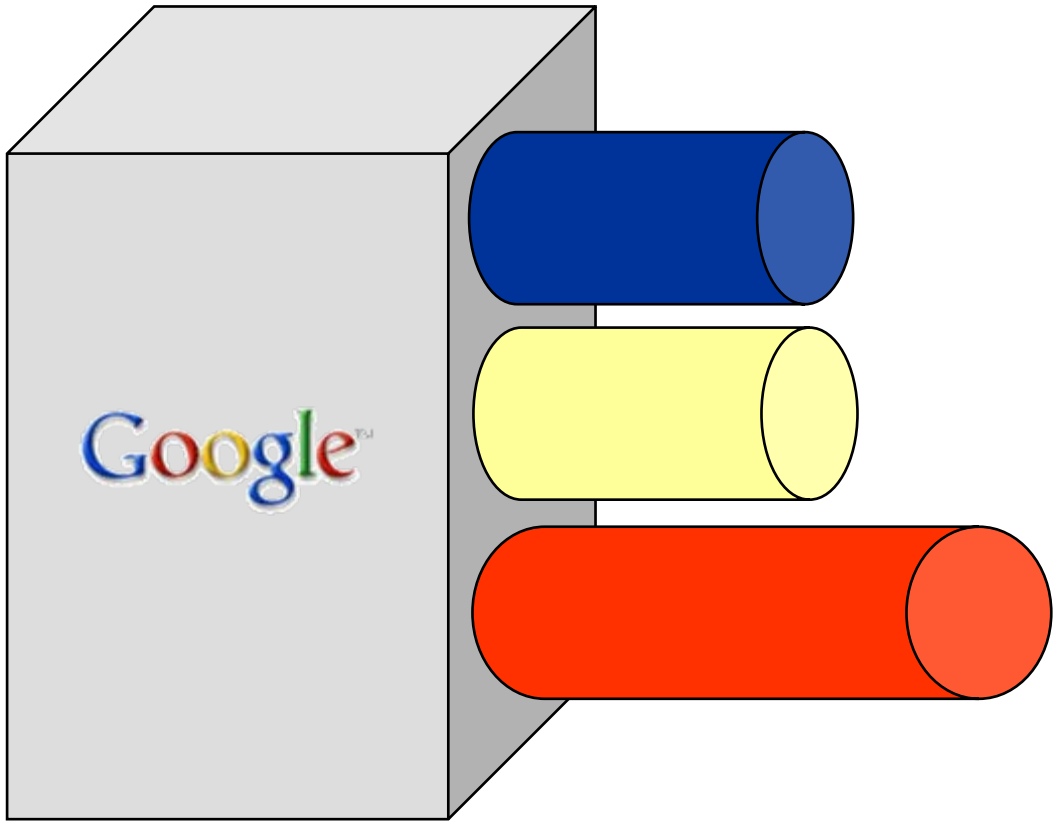
doGetCachedPage

Rückgabe:

doGetCachedPageResponse

Web Service von Google

Freie Universität Berlin



Dienst:

Rechtschreibkorrektur

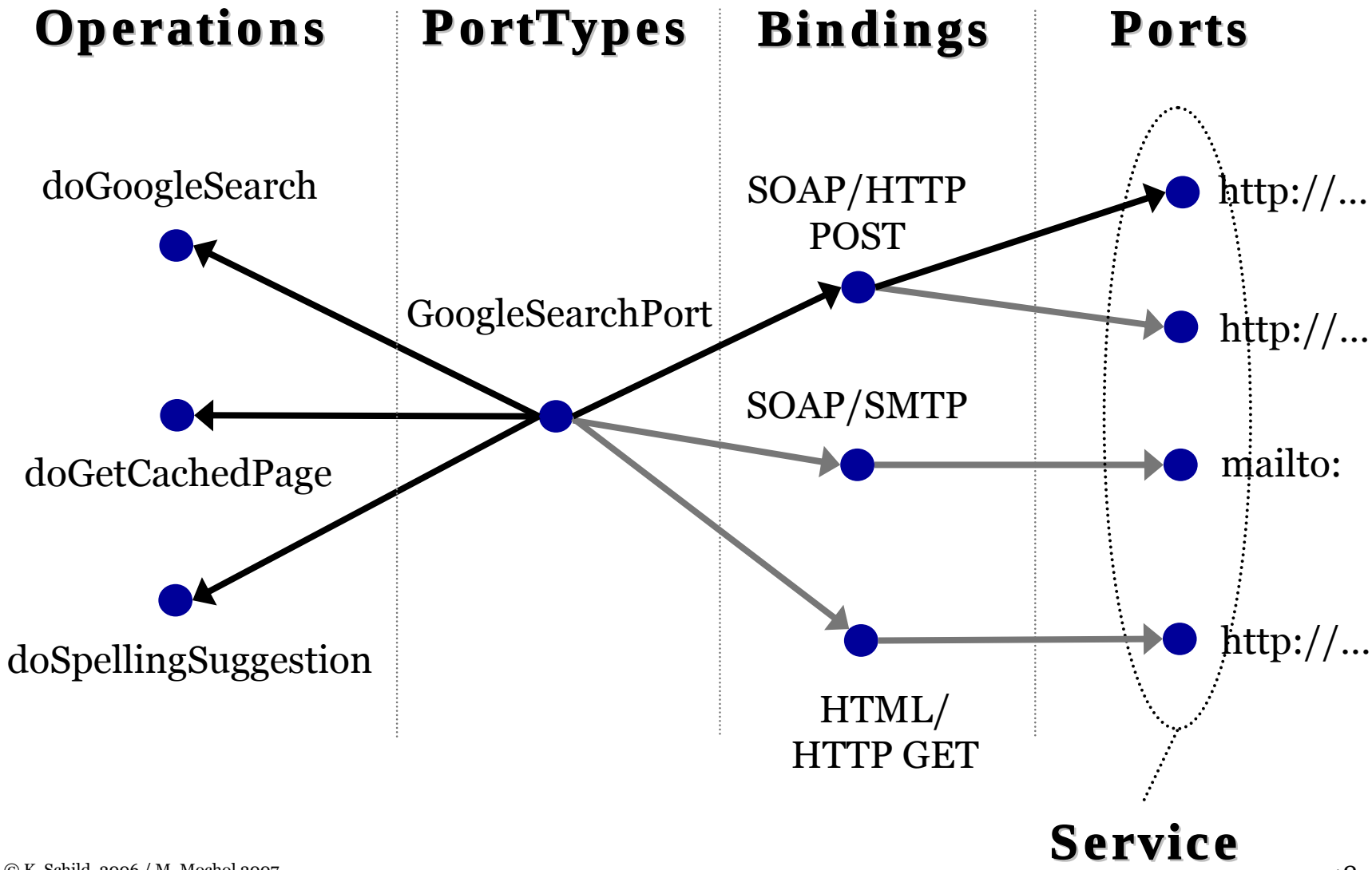
Name der Operation:

doSpellingSuggestion

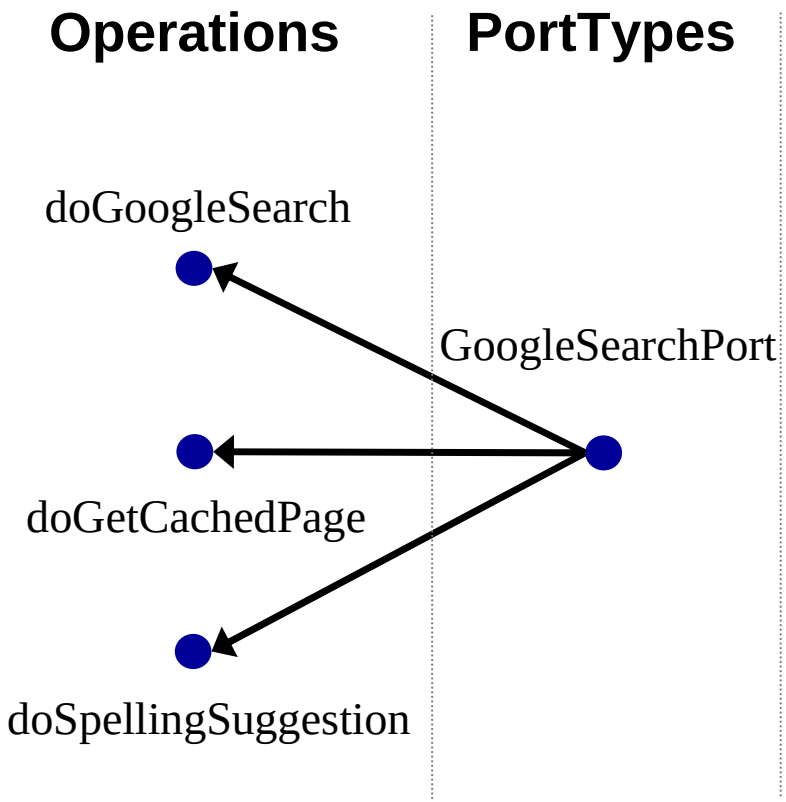
Rückgabe:

doSpellingSuggestionResponse

Grundstruktur

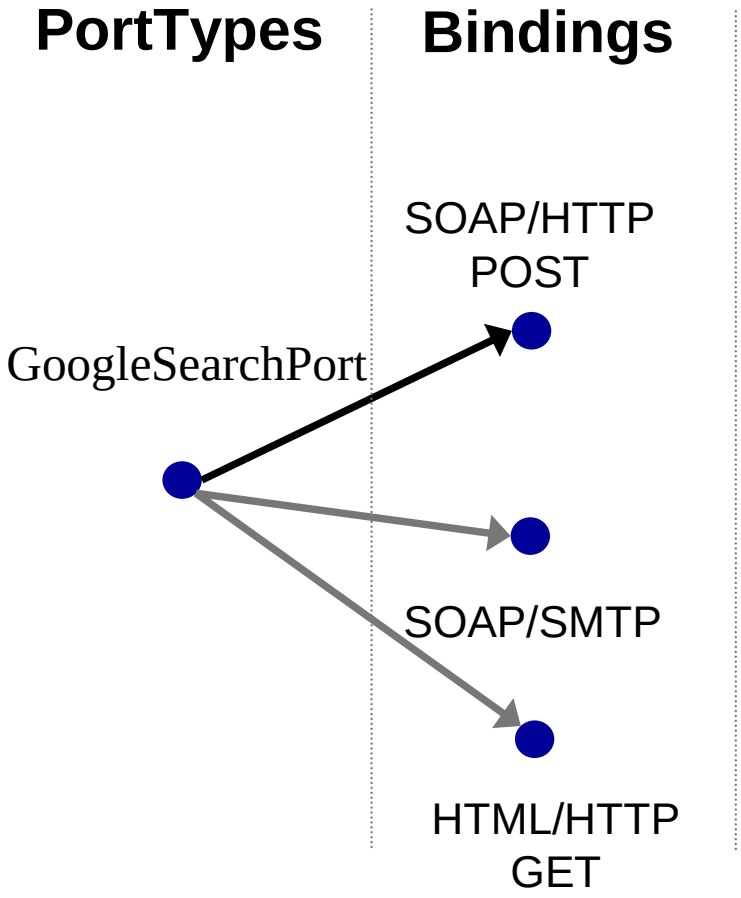


Abstrakte Schnittstellen



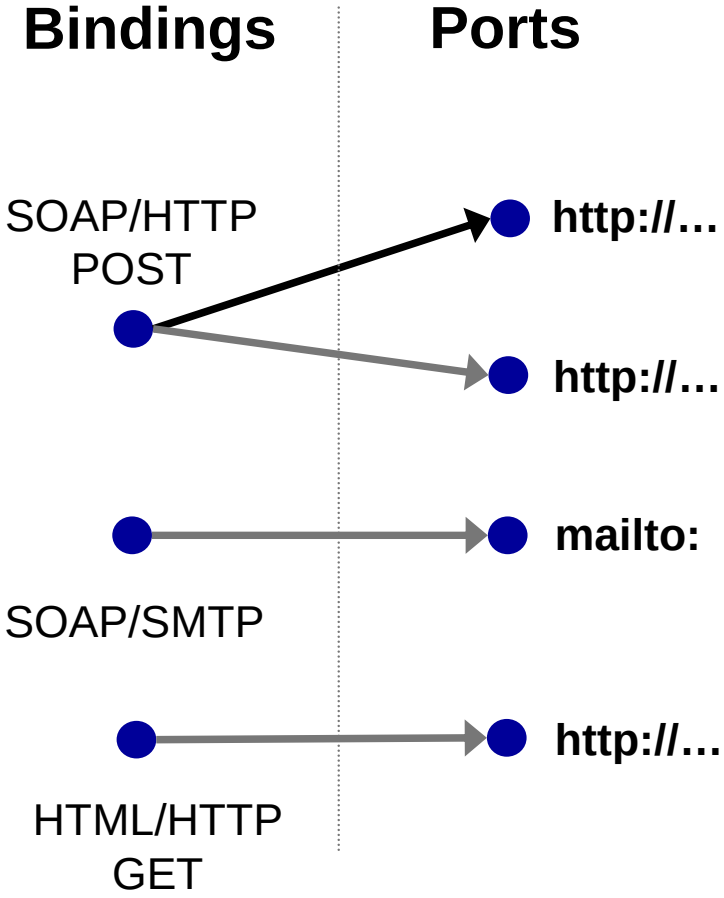
- in WSDL 1.1 portType genannt
- in WSDL 2.0 interface genannt
- portType = Menge von abstrakten Operationen
- jede abstrakte Operation beschreibt Eingangs- und Ausgangsnachricht
- meist nur ein portType, aber in WSDL 1.1 auch mehrere möglich

Bindungen



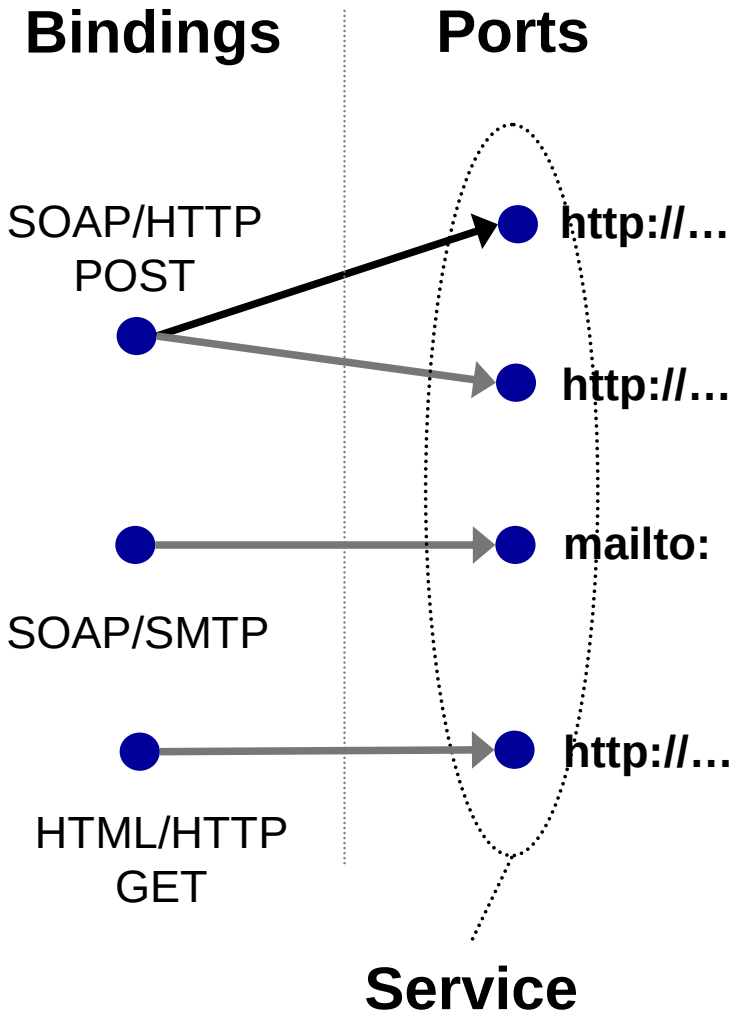
- in WSDL binding genannt
- für jede abstrakte Schnittstelle (portType) mindestens eine Bindung
- ein portType kann also mit unterschiedlichen Bindungen realisiert sein

Kommunikationsendpunkte



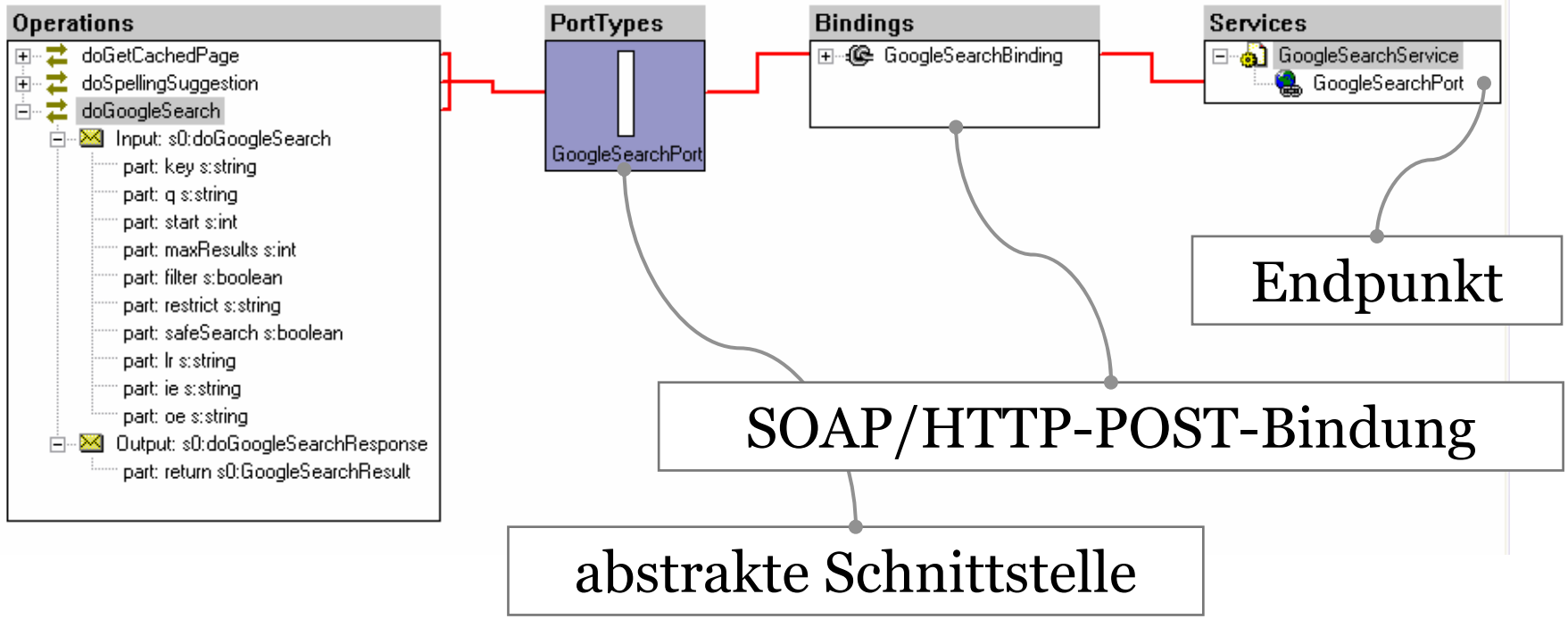
- in WSDL 1.1 port genannt
- in WSDL 2.0 endpoint genannt
- port = Bindung + Web-Adresse
- für jede Bindung (binding) mindestens ein port
- ein binding kann also über unterschiedliche Web-Adressen zugänglich sein

Service



- Menge von ports bilden zusammen einen Service
- ports können auch in verschiedene Services gruppiert werden
- ports eines Service = semantisch äquivalente Alternativen

Die WSDL-Beschreibung von Google



WSDL 1.1. – Elemente (1)

Freie Universität Berlin

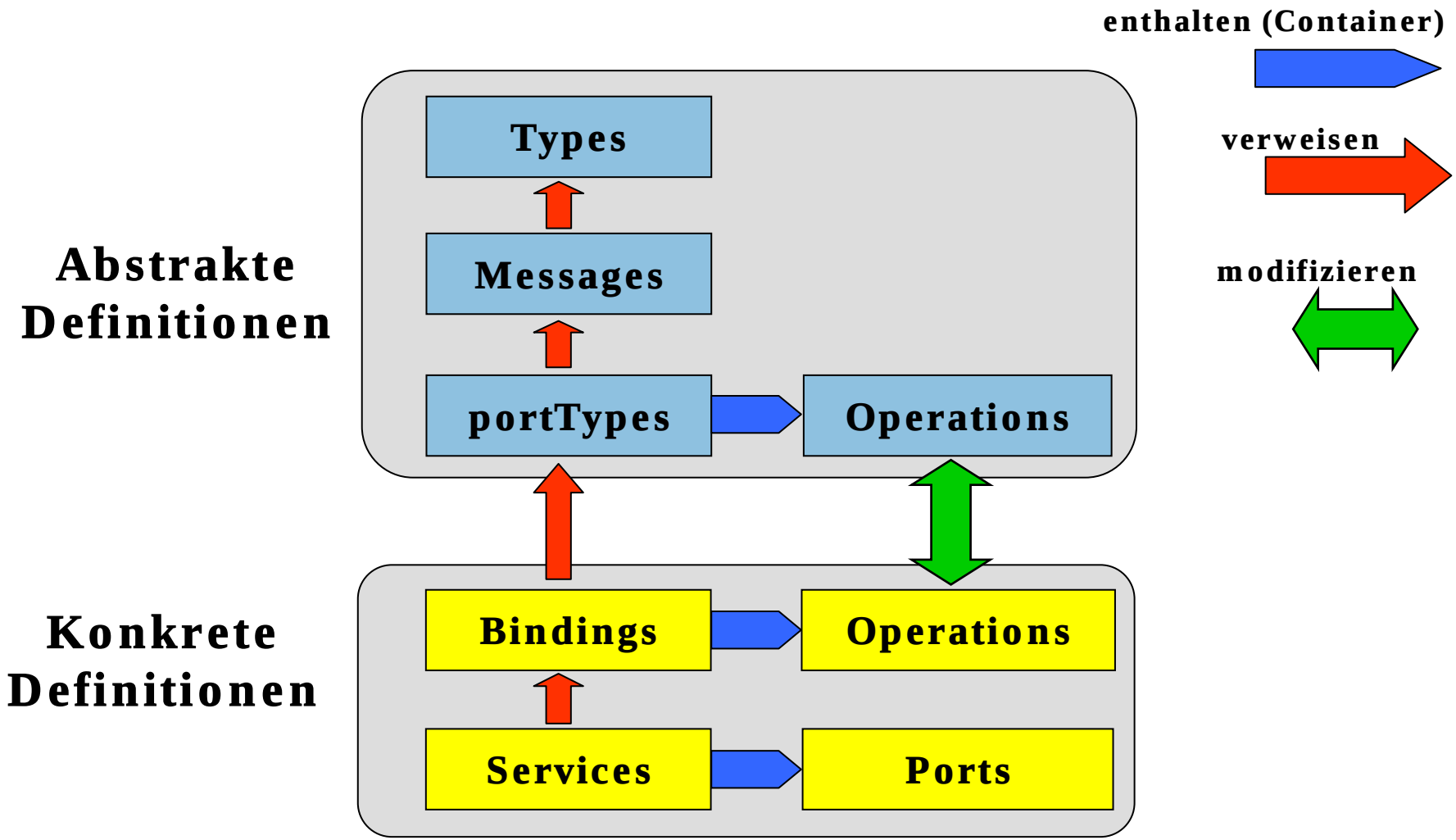
Element	Beschreibung
Abstrakte Beschreibung	
`<types>` … `</types>`	- Maschinen- und sprachunabhängige Typdefinitionen → definiert die verwendeten **Datentypen**
`<message>` … `</message>`	- **Nachrichten**, die übertragen werden sollen - Funktionsparameter (Trennung zwischen Ein- und Ausgabeparameter)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

WSDL 1.1. – Elemente (2)

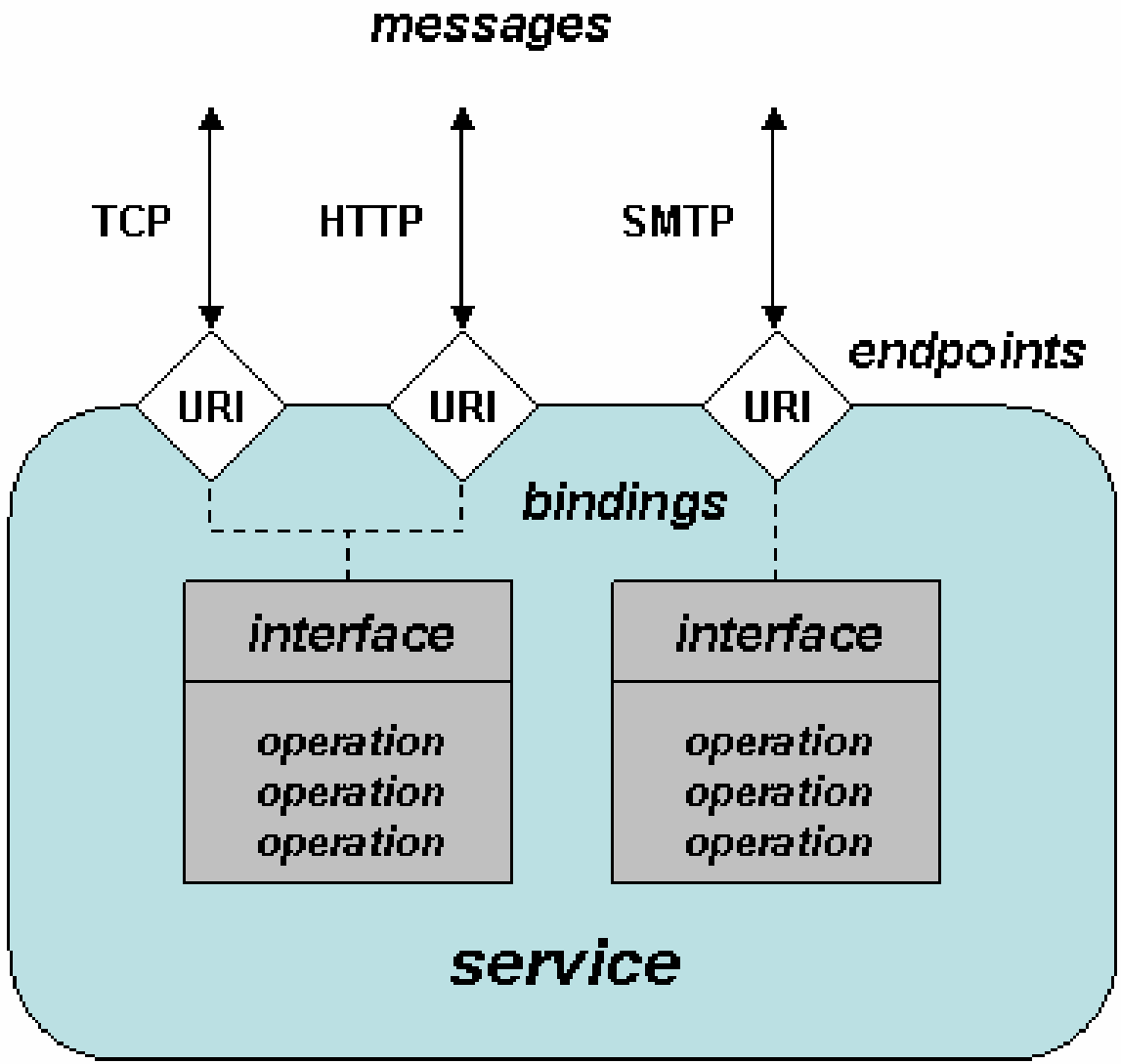
Freie Universität Berlin

Element	Definiert
Konkrete Beschreibung	
`<binding>…</binding>`	- Kommunikationsprotokoll, der 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 mehrere Ports)

Abstrakte vs. Konkrete Definition



WSDL – Schematische Darstellung



Quelle: http://msdn2.microsoft.com/en-us/library/ms996486.aspx

XML-Syntax von WSDL

```
<?xml version="1.0"?>
<definitions name="GoogleSearch"
    targetNamespace="urn:GoogleSearch"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
    ...
</definitions>
```

- **Wurzel-Element** `definitions` aus entsprechendem Namensraum
- **Namensraum** von `definitions` = Version
- WSDL-Beschreibung kann Ziel-Namensraum definieren

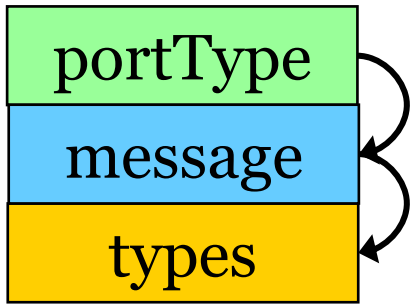
⇨ SOAP-Nachricht kann auf diesen Ziel-Namensraum verweisen

Prinzipieller Aufbau (1/4)

Datenschema

<types>

```
<?xml version="1.0"?>
<definitions name="GoogleSearch"
        targetNamespace="urn:GoogleSearch"
          ...
          xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>...</types>
  <message name="doGoogleSearch">...</message>
  <message name="doGoogleSearchResponse">...</mes
    ...
</definitions>
```



types

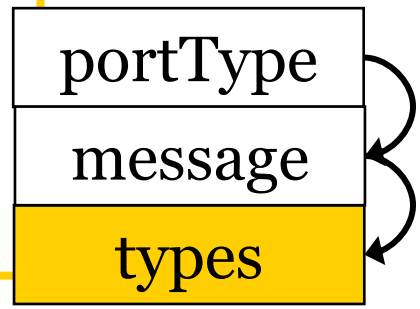
- Definition von Datentypen
- werden für Spezifikation von abstrakten Nachrichten verwendet

message

- Definition einer abstrakten Nachricht
- werden für Spezifikation der abstrakten Schnittstelle verwendet

Definition von Datentypen

```
<types>
  <schema xmlns="http://www.w3.org/2001/XMLSchema"
          targetNamespace="urn:GoogleSearch">
    ...
  </schema>
</types>
```



- Datentypen für Spezifikation von abstrakten Nachrichten
- XML-Schema als Typsystem empfohlen, theoretisch jedes andere Typsystem aber auch erlaubt
- Beachte: XML-Schema kann auch verwendet werden, wenn Nachrichten nicht in XML übertragen werden.

Beispiel: Google™-Suchresultat

Freie Universität Berlin

```
<types>
  <xsd:schema  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                    xmlns:tns="urn:GoogleSearch"
            targetNamespace="urn:GoogleSearch">
    <xsd:complexType name="GoogleSearchResult">
      <xsd:all>
        <xsd:element name="estimatedTotalResultsCount"  type="xsd:int"/>
        <xsd:element name="resultElements" type="tns:ResultElementArray"/>
        <xsd:element name="searchQuery" type="xsd:string"/>
        <xsd:element name="startIndex" type="xsd:int"/>
        <xsd:element name="endIndex" type="xsd:int"/>
                    ...
      </xsd:all>
    </xsd:complexType>
  </schema>
</types>
```

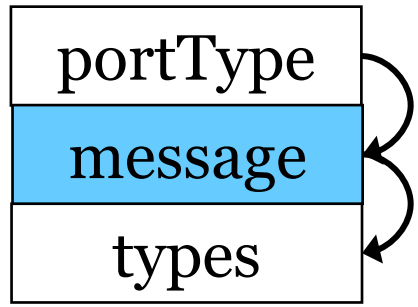
- vollständiges XML-Schema
- Ziel-Namensraum normalerweise identisch mit Ziel-Namensraum von WSDL

Prinzipieller Aufbau (2/4)

Funktionalität

<message>

```
<definitions name="GoogleSearch"
        targetNamespace="urn:GoogleSearch"
          ...
          xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>...</types>
  <message name="doGoogleSearch">...</message>
  <message name="doGoogleSearchResponse">...</message>
    ...
</definitions>
```



Definition einer abstrakten Nachricht

```
<message name="doGoogleSearchResponse">
  <part name="return" type="tns:GoogleSearchResult"/>
</message>
```

- Name muss innerhalb der WSDL eindeutig sein
- setzten sich aus logischen Bestandteilen (**`part`**s) zusammen: #**`part`**s ≥ 1
- **`part`** kann z.B. ein Parameter eines RPCs sein
- jedes **`part`** hat eindeutigen Namen
- Reihenfolge der logischen Bestandteile unerheblich

Definition strukturierter Nachrichten

zwei unterschiedliche Modellierungen

1. mehrere parts:

```
<message name="doGoogleSearchResponse">
  <part name="param1" element="tns:param1"/>
  <part name="param2" element="tns:param2"/>
</message>
```

2. ein part mit komplexen Datentyp:

```
<message name="doGoogleSearchResponse">
  <part name="return" type="tns:complexType"/>
</message>
```

tns:complexType könnte z.B. 2 Parameter enthalten

Definition strukturierter Nachrichten

zwei unterschiedliche Modellierungen

1. mehrere parts:

```
<message name="doGoogleSearchR
  <part name="param1" element=
  <part name="param2" element=
</message>
```

2. ein part mit komplexen Daten

```
<message name="doGoogleSearchR
  <part name="return" type="tns
</message>
```

tns:complexType könnte z.B.

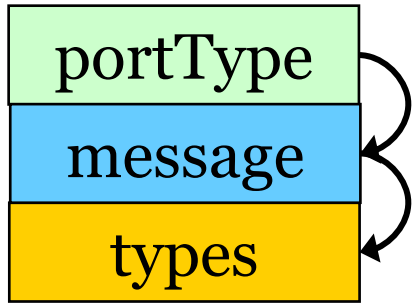
Unterschiede

- parts immer reihenfolgeunabhängig
- parts können in Bindung unterschiedlich behandelt werden, z.B.:

  ein part in Body der SOAP-Nachricht, ein anderes part in den Header

<portType>

```
<definitions name="GoogleSearch"
        targetNamespace="urn:GoogleSearch"
         ...
         xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>...</types>
  <message name="doGoogleSearch">...</message>
  <message name="doGoogleSearchResponse">...</message>
  <portType name="GoogleSearchPort">...</portType>
    ...
</definitions>
```



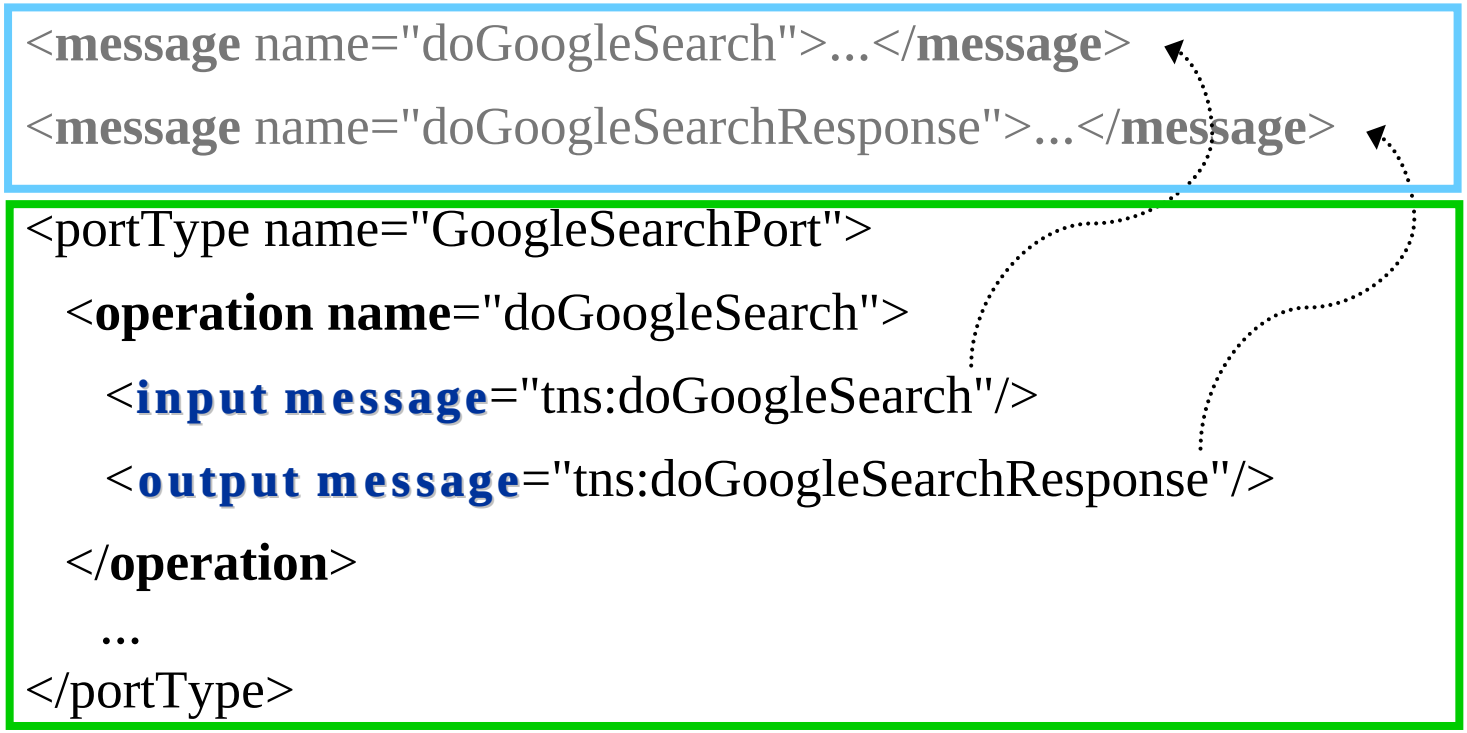
Definition der abstrakten Schnittstelle

```
<message name="doGoogleSearch">...</message>
<message name="doGoogleSearchResponse">...</message>

<portType name="GoogleSearchPort">
  <operation name="doGoogleSearch">
    <input message="tns:doGoogleSearch"/>
    <output message="tns:doGoogleSearchResponse"/>
  </operation>
  <operation name="doSpellingSuggestion">
    ...
  </operation>
    ...
</portType>
```

- abstrakte Schnittstelle = Menge von abstrakten Operationen (**operation**s)

Abstrakte Schnittstelle: operation

```
<message name="doGoogleSearch">...</message>
<message name="doGoogleSearchResponse">...</message>

<portType name="GoogleSearchPort">
  <operation name="doGoogleSearch">
    <input message="tns:doGoogleSearch"/>
    <output message="tns:doGoogleSearchResponse"/>
  </operation>
    ...
</portType>
```



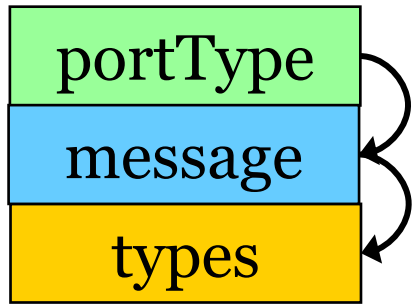
- definiert einfaches Interaktionsmuster mit Eingangs- und Ausgangs-Nachrichten.
- wichtig: verwendet keine Datentypen, sondern abstrakte Nachrichten

Abstrakte Nachricht vs. Datentyp

```
<message name="doGoogleSearchResponse">
  <part name="return" type="tns:GoogleSearchResult"/>
</message>
 ...
<portType>
  <operation name="doGoogleSearch">
    <input message="tns:doGoogleSearch"/>
    <output message="tns:doGoogleSearchResponse"/>
  </operation>
 ...
</portType>
```

Datentyp

abstrakte Nachricht



Datentyp / Nachricht /Porttyp

```
<types>
  <xsd:schema  xmlns:xsd="..."  xmlns:tns="..." targetNamespace="...">
    <xsd:complexType name="GoogleSearchResult">
      ...
    </xsd:complexType>
  </schema>
 </types>
```

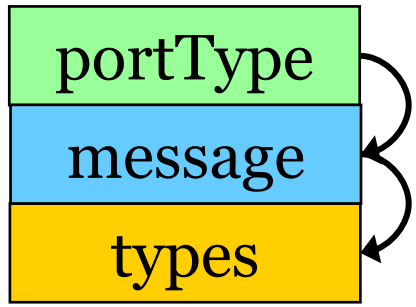
Definition des Datentyps

```
<message name="doGoogleSearch">...</message>
<message name="doGoogleSearchResponse">
  <part name="return" type="tns:GoogleSearchResult"/>
</message>
```

Definition einer abstrakten Nachricht

```
...
<portType>
  <operation name="doGoogleSearch">
    <input message="tns:doGoogleSearch"/>
    <output message="tns:doGoogleSearchResponse"/>
  </operation>
  ...
</portType>
```

Definition einer abstrakten Schnittstelle



Interaktionsmuster (2)

Einweg (oneway)

Client  Server

```
<operation name="...">
    <input message="..."/>
</operation>
```

Anfrage-Antwort (request-response)

Client  Server

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

Interaktionsmuster (2)

Benachrichtigung (notification)

Client  Server

```
<operation name="...">
    <output message="..."/>
</operation>
```

Benachrichtigung-Antwort (solicit-response)

Client  Server

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

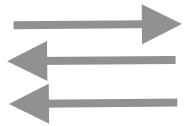
Abstrakte (!) Interaktionsmuster

Freie Universität Berlin

- Anfrage-Antwort-Muster müssen nicht mit einer Netzwerkkommunikation (z.B. mit HTTP) realisiert werden.
- auch mit zwei unabhängigen Kommunikationen (z.B. E-Mails) möglich
- Realisierung wird erst in der Bindung (binding) festgelegt

Komplexe Interaktionsmuster

- Registrierung zum Börsenticker
- Bestätigung der Registrierung
- aktueller Börsenkurs (Benachrichtigung)

Client  Server

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

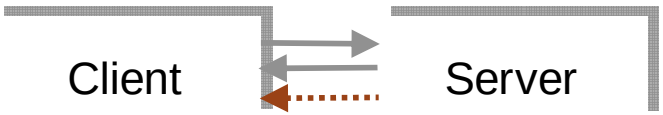
In WSDL nicht erlaubt!

Fehlermeldungen

Freie Universität Berlin

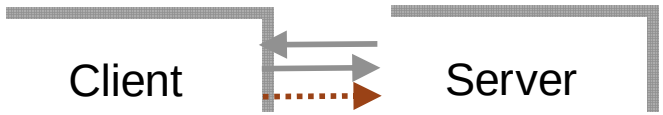
Anfrage-Antwort

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



Benachrichtigung-Antwort

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



- abstrakte Beschreibung von Fehlermeldungen
- statt Antwort/Bestätigung kann auch Fehler gemeldet werden

Prinzipieller Aufbau (3/4)

Protokollbindung

Bindungen

```
<definitions name="GoogleSearch"
        targetNamespace="urn:GoogleSearch"
         ...
         xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>...</types>
  <message name="doGoogleSearch">...</message>
  <message name="doGoogleSearchResponse">...</message>
  <portType name="GoogleSearchPort">...</portType>
  <binding name="GoogleSearchBinding" type="tns:GoogleSearchPort">
    ...
  </binding>
  <binding ...>...</binding>
    ...
</definitions>
```

Grundstruktur von binding

```
<binding name="GoogleSearchBinding" type="tns:GoogleSearchPort">
  Erweiterungselement
  <operation name="doGoogleSearch">
    Erweiterungselement
    <input>
        Erweiterungselement
    </input>
    <output>
    Erweiterungselement
    </output>
  </operation>
    ...
</binding>
```

- definiert eine Bindung
- **name**: eindeutiger Name der Bindung
- **type**: die zu realisierende abstrakte Schnittstelle (portType)
- mehrere binding-Elemente für eine abstrakte Schnittstelle erlaubt

Grundstruktur von binding

```
<binding name="GoogleSearchBinding" type="tns:GoogleSearchPort">
  Erweiterungselement
  <operation name="doGoogleSearch">
    Erweiterungselement
    <input>
        Erweiterungselement
    </input>
    <output>
        Erweiterungselement
    </output>
  </operation>
    ...
</binding>
```

- Bindung mit sog. Erweiterungselementen (extensibility elements) kodiert
- Informationen über Bindung auf allen Ebenen:
  - Bindung allgemein
  - einzelnen Operationen
  - Input- und Output-Nachrichten
  - Fehlermeldungen

Erweiterungselemente

- Platzhalter in der WSDL-Grammatik
- WSDL 1.1 standardisiert drei Bindungen:

1. SOAP

2. HTTP
   - GET & POST Methoden
   - absolute URI für jedes Port
   - relative URI für jeder Operation
   - optional: encoding für Anfrage-Nachricht (URL encoding, URL replacement)

3. MIME
   - spezifiziert MIME types (text/xml, multipart/related, ...)

Prinzipieller Aufbau (4/4)

Service-Adresse

<service>

```
<definitions name="GoogleSearch"
        targetNamespace="urn:GoogleSearch"
         ...
         xmlns="http://schemas.xmlsoap.org/wsdl/">
<!-- abstrakte Definition -->
  <types>...</types>
  <message name="doGoogleSearch">...</message>
  <message name="doGoogleSearchResponse">...</message>
  <portType name="GoogleSearchPort">...</portType>
<!-- konkrete Definition -->
  <binding name="GoogleSearchBinding"  type="tns:GoogleSearchPort">
    ...
   </binding>
  <service name="GoogleSearchService">...</service>
 </definitions>
```

Service

```
<service name="GoogleSearchService">
  <port name="GoogleSearchPort" binding="tns:GoogleSearchBinding">
    <soap:address location="http://api.google.com/search/beta2"/>
  </port>
  <port>...</port>
</service>
```

- Service = Menge von Ports
- Port = Bindung + Web-Adresse
- Ports eines Service sollen semantisch äquivalente Alternativen einer abstrakten Schnittstelle sein

Bindung

Bindungen

Freie Universität Berlin

1. SOAP

2. HTTP
- GET & POST Methoden
- absolute URI für jedes Port
- relative URI für jeder Operation
- optional: encoding für Anfrage-Nachricht (URL encoding, URL replacement)

3. MIME
- spezifiziert MIME types (text/xml, multipart/related, ...)

Bindung

SOAP-Bindung

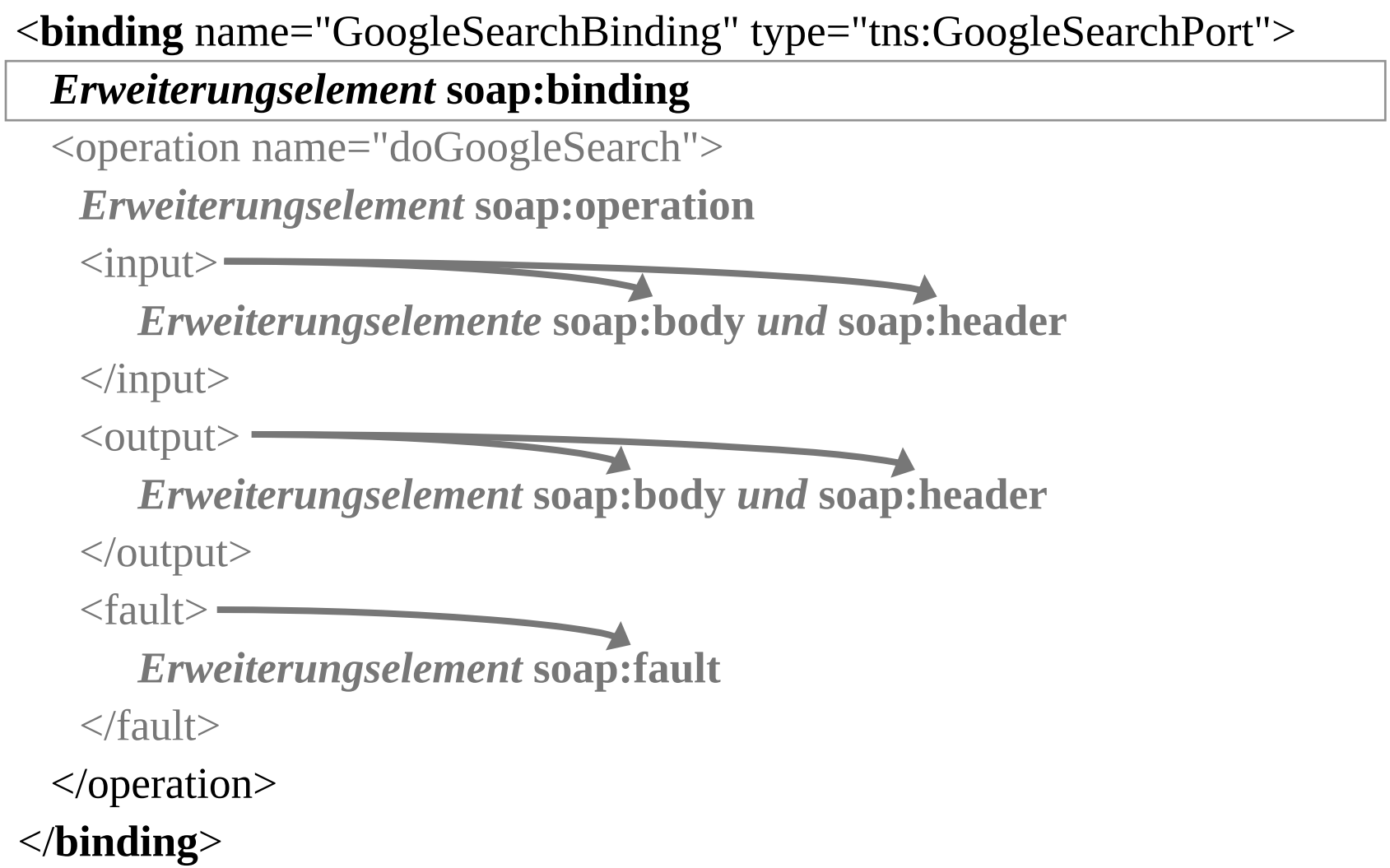
SOAP-Bindung

Freie Universität Berlin

```
<binding name="GoogleSearchBinding" type="tns:GoogleSearchPort">
  Erweiterungselement soap:binding
  <operation name="doGoogleSearch">
    Erweiterungselement soap:operation
    <input>
      Erweiterungselemente soap:body und soap:header
    </input>
    <output>
      Erweiterungselemente soap:body und soap:header
    </output>
    <fault>
      Erweiterungselement soap:fault
    </fault>
  </operation>
</binding>
```

- Erweiterungselemente beschreiben Abbildung portType ➔ SOAP-Nachricht
- Beachte: WSDL 1.1 benutzt SOAP 1.1

Bindung allgemein: soap:binding



Bindung allgemein: soap:binding

```
<binding name="GoogleSearchBinding" type="tns:GoogleSearchPort">
  <soap:binding xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="doGoogleSearch">
    ...
  </operation>
</binding>
```

- soap:binding: gibt an, dass portType mit SOAP realisiert ist
- style: entfernter Prozeduraufruf (rpc) oder Messaging (document)
- transport: Übertragungsprotokoll
- Beachte: HTTP meint hier HTTP-POST
- hier auch möglich: transport="http://.../soap/smtp"

SOAP-Stile: Etwas irreführend!

Freie Universität Berlin

style="rpc"

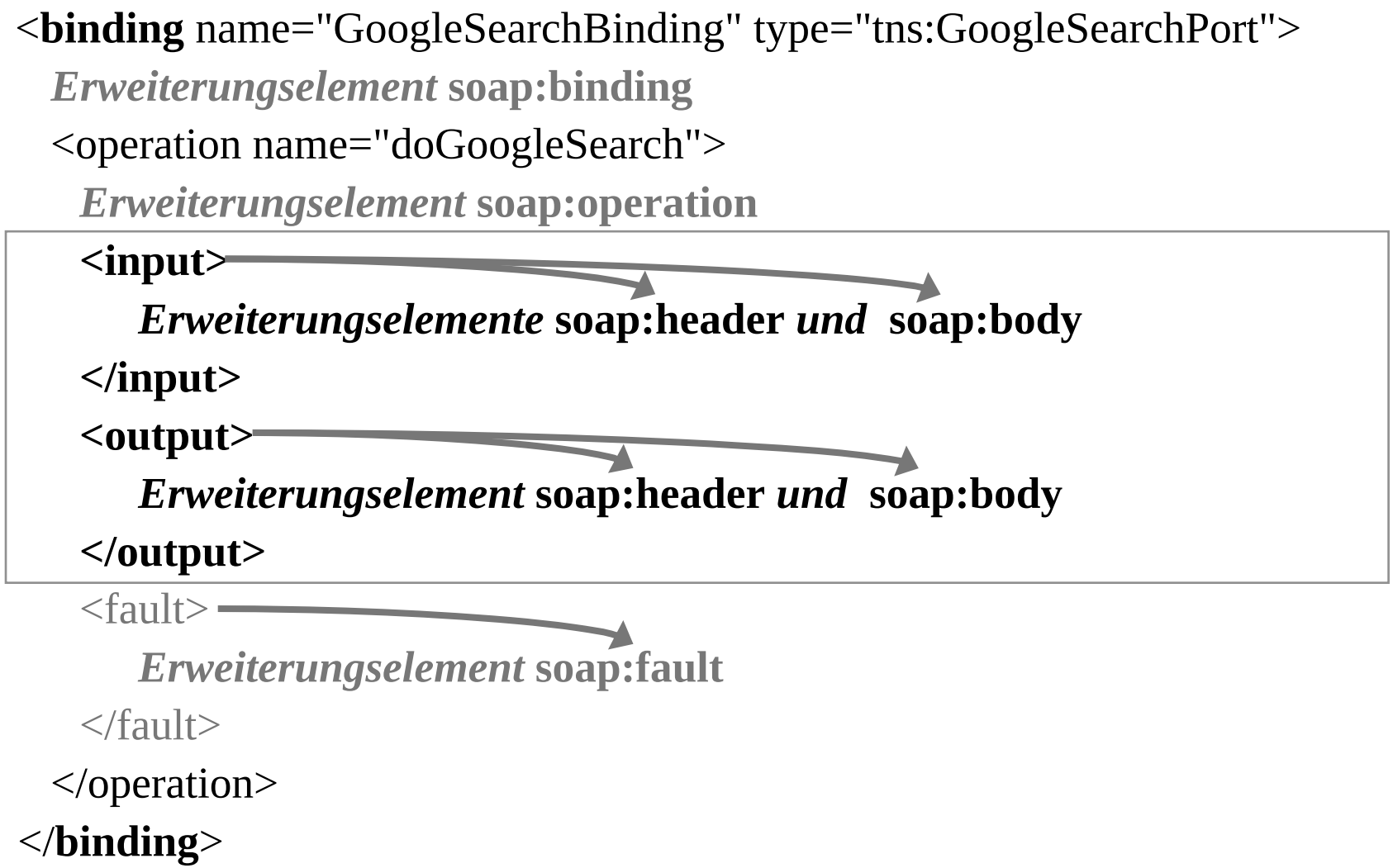
```
<body>
  <procedure-name>
    <part-1>...<part-1>
      ...
    <part-n>...<part-n>
  </procedure-name>
</body>
```

style="document"

```
<body>

    <part-1>...<part-1>
      ...
    <part-n>...<part-n>

</body>
```

- legt lediglich Struktur des SOAP-Nachrichteninhalts (Body) fest, darüber hinaus keine Bedeutung

Abbildung der abstrakten Nachrichten



soap:body

```
<operation name="doGoogleSearch">
  ...
  <input>
    <soap:body use="literal"/>
  </input>
  <output>...</output>
</operation>
```

- soap:body: Wie wird abstrakte input- bzw. output-Nachricht auf SOAP-Body abgebildet?

use="literal"

```
<operation name="doGoogleSearch">
  ...
  <input>
    <soap:body use="literal"/>
  </input>
  <output>...</output>
</operation>
```

- use="literal": abstrakte Nachricht wird unverändert übernommen

use="encoded"

```
<operation name="doGoogleSearch">
  ...
  <input>
    <soap:body use="encoded"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>...</output>
</operation>
```

- use="encoded": Abstrakte Nachricht wird mit Hilfe eines bestimmten Verfahrens (encodingStyle) kodiert.
- hier Kodierungsverfahren von SOAP (➔ RPC-Struktur, einschl. SOAP-Arrays)

soap:header

```
<operation name="doGoogleSearch">
 ...
 <input>
    <soap:header messsage="tns:doGoogleSearch" part="key"
                 use="literal"/>
   <soap:body parts="q start maxResults ..." use="encoded" .../>
 </input>
 <output>...</output>
</operation>
```

input-Nachricht wird auf SOAP-Header und -Body verteilt.

- Teile der abstrakten Nachricht ➔ SOAP-Header
- für jeden Header Block ein soap:header-Element
- Struktur von soap:header analog zu soap:body.

soap:address

```
<binding name="GoogleSearchBinding" type="tns:GoogleSearchPort">
  <soap:binding xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
   ...
</binding>
<port name="GoogleSearchPort" binding="tns:GoogleSearchBinding">
    <soap:address location="http://api.google.com/search/beta2"/>
</port>
```

- Jedem Port muss genau eine Web-Adresse (soap:address) zugeordnet sein.
- wichtig: Web-Adresse muss zum Transportprotokoll der Bindung passen.

Bindung

HTTP-Bindung

Beispiel für HTTP-Bindung (fiktiv)

Freie Universität Berlin

- HTTP-GET-Anfrage kodiert alle Parameter in URL:

```
GET /search/beta2/doGoogleSearch?key=45675353&q=Anfrage&...
    HTTP/1.1
Host: api.google.com
Content-Type: text/html; charset="utf-8"
Content-Length: nnnn
```

Antwort soll HTML-Dokument sein

HTTP-Bindung

```
<binding name="GoogleSearchBinding" type="tns:GoogleSearchPort"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/">
  <http:binding verb="GET"/>
  <operation name="doGoogleSearch">
    ...
    <input><http:urlEncoded/></input>
    <output><mime:content type="text/html"/></output>
  </operation>
</binding>

<port name="GoogleSearchPort" binding="tns:GoogleSearchBinding">
    <http:address
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
       location="http://api.google.com/search/beta2"/>
</port>
```

⇨ browser-basiertes Google mit WSDL beschrieben!

Und XML statt HTML als Antwort

Freie Universität Berlin

```
<binding name="GoogleSearchBinding" type="tns:GoogleSearchPort">
  <http:binding verb="GET"/>
  <operation name="doGoogleSearch">
    ...
    <input><http:urlEncoded/></input>
    <output><mime:mimeXml/></output>
  </operation>
</binding>
```

Freie Universität Berlin

Modularisieren von WSDL-Beschreibungen

Modularisierung

Freie Universität Berlin

Prinzip

- Gruppierung einzelner Komponente nach verschiedenen Gesichtspunkten
- Einbinden von Dokumententeilen
  - include
  - import

Vorteile

- ermöglicht Wiederverwendung von Teilen einer WSDL-Datei
- erleichtert die Wartbarkeit
- erleichtert die Lesbarkeit
- erlaubt eine klare Strukturierung

<include>

- aufteilen verschiedener Komponenten einer Dienstdefinition in unabhängige WSDL-Dokumente
- Einbindung von Komponenten aus dem gleichen (wie die der inkludierten Beschreibung) Zielnamensraum

- eingebundene Komponente
  - Teil des Komponentenmodells der inkludierten Beschreibung
  - werden über ihren qualifizierten Namen (von anderen Komponenten) referenziert

```
<include location="URI">
    ...
</include>
```

<import>

- Einbindung von Komponenten aus anderen Zielnamensräumen

- importierten Komponente werden über ihren qualifizierten Namen (von anderen Komponenten) referenziert

```
<import
        namespace="URI"
        location="URI " >
    ...
</import>
```

Vor- und Nachteile von WSDL

Vorteile von WSDL

Freie Universität Berlin

sollen unterschiedliche Probleme lösen

- Interoperabilität zwischen unterschiedlichen Implementierungs-plattformen
- gemeinsame Technologie für verschiedene Anwendungsgebiete
- geringe Entwicklungskosten durch allgemein verfügbare Basistechnologien

Vorteile

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

Nachteile von WSDL

Freie Universität Berlin

schaffen neue Probleme

- nicht alle Entwicklungen werden akzeptiert (vgl. UDDI, REST vs. SOAP)
- geforderte Funktionalitäten sind noch nicht 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 Aspekten (quality of service)
- keine Sicherheitsaspekte
- unzureichend, um automatisch die Kompatibilität (Interoperabilität) zweier Web Services feststellen zu können → Semantic Web Services

Wie geht es weiter?

Freie Universität Berlin

WSDL

- ☑ Prinzipieller Aufbau
- ☑ SOAP- und HTTP-Bindungen
- ☑ Vor- und Nachteile

Übung nächste Woche (letzte Übung)

- WSDL

Vorlesung nächste Woche

- Web Services in der Praxis & Ausblick