



WSDL

Dr. Malgorzata Mochol
Freie Universität Berlin
Institut für Informatik
Netzbasierte Informationssysteme
mochol@inf.fu-berlin.de

Vorlesungs- -termine	Vorlesung (4 + 1 + 1)	Übung – 2 Termine	Übungs- termine
10.06.	Web Services, RPCs vs. Messaging		
17.06.	SOAP im Detail	SOAP	24./25.06
24.06. (heute)	WSDL im Detail	WSDL	01./02.07
01.07.	Web Services in der Praxis & Ausblick		
08.07.	Rückblick		
15.07.	Klausur		

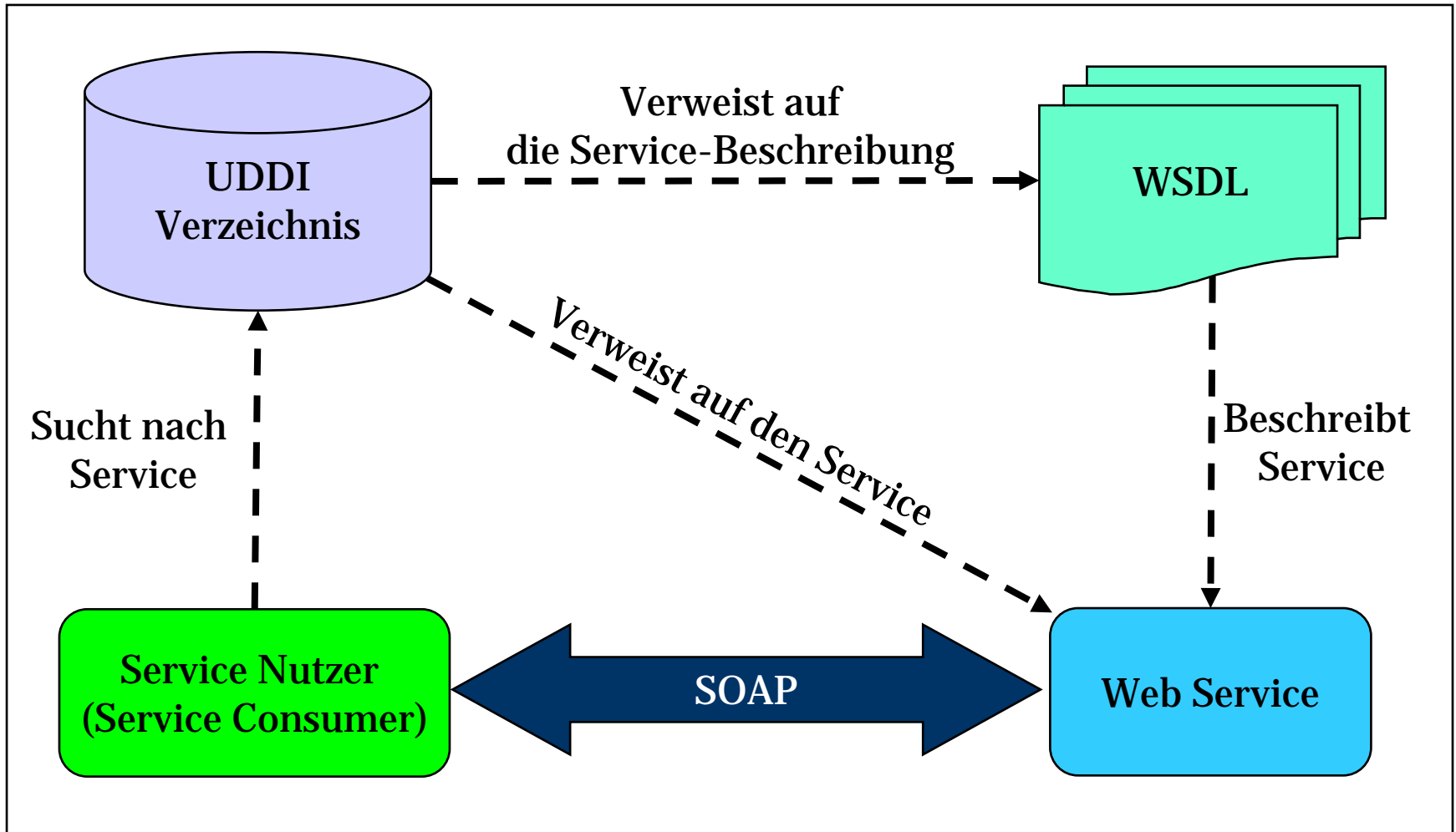
Heutige Vorlesung

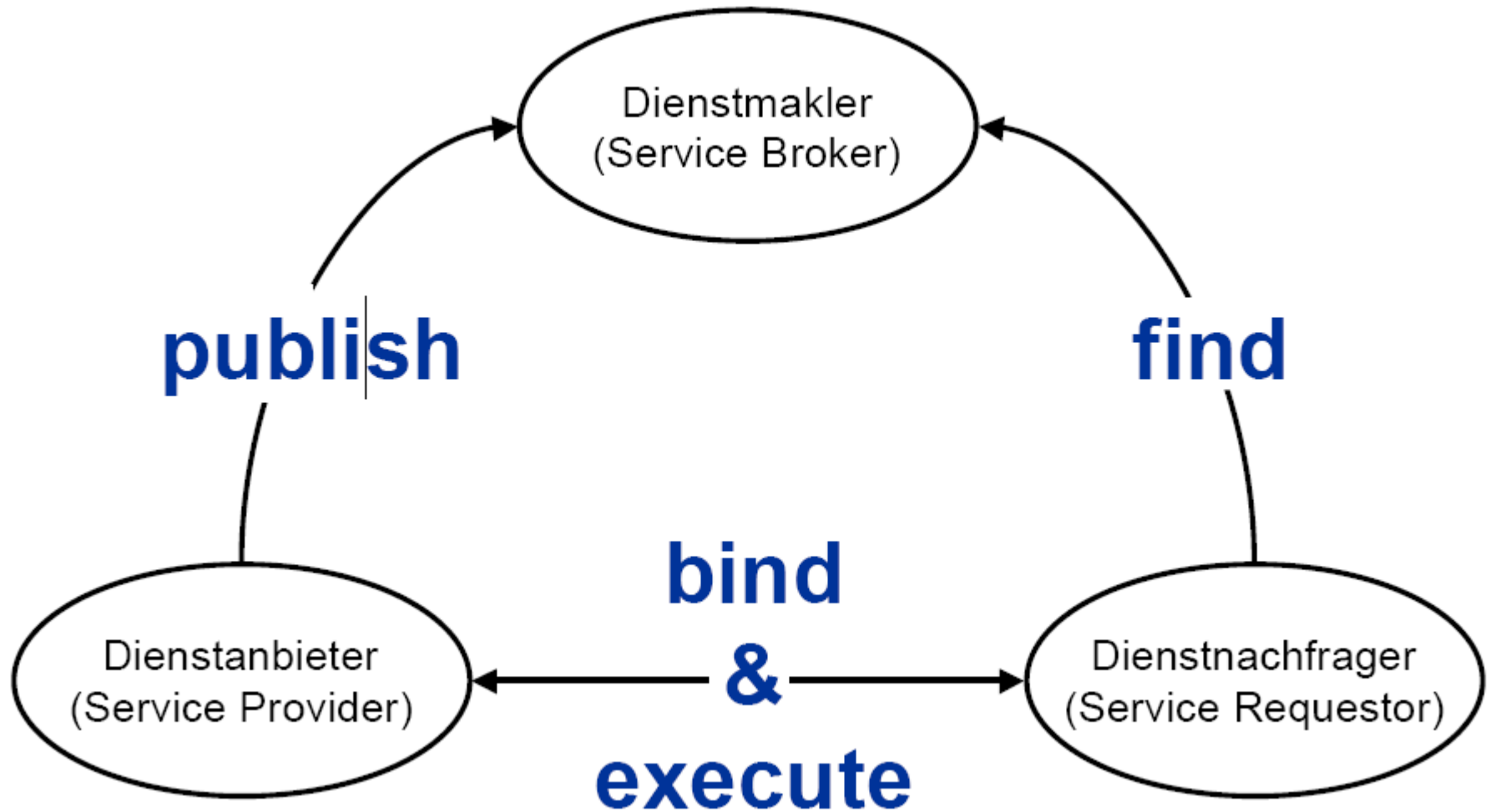
letzte Woche

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

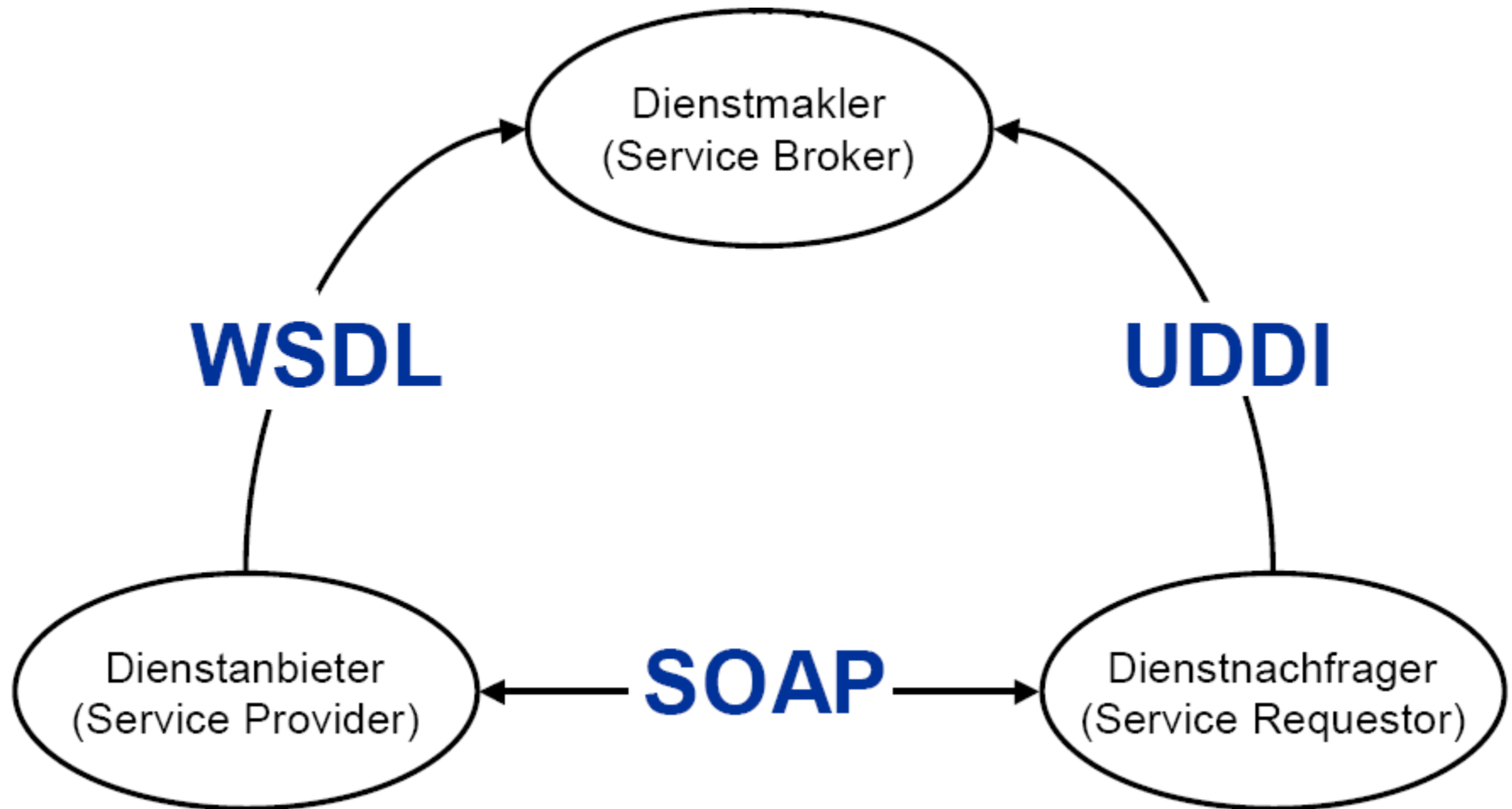
heutige Vorlesung → WSDL

- Wozu WSDL-Syntax verstehen?
- prinzipieller Aufbau
 - Datenschemata
 - Funktionalität
 - Protokollbindung
 - Service-Aufbau
- standardisierte Bindungen (SOAP & HTTP)
- Vor- und Nachteile





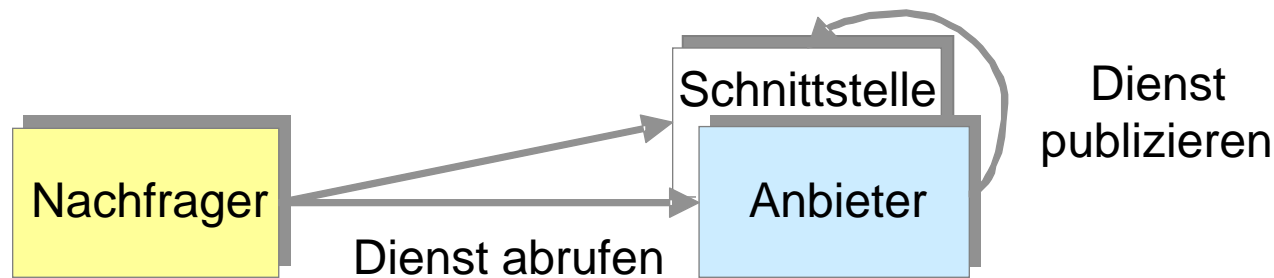
Quelle: <http://www.jeckle.de/files/WSDL2002.pdf>



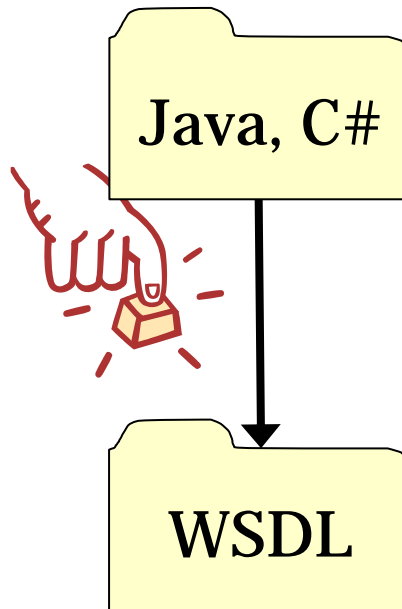
Quelle: <http://www.jeckle.de/files/WSDL2002.pdf>

- 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)

Formale Beschreibung der Schnittstelle von Services



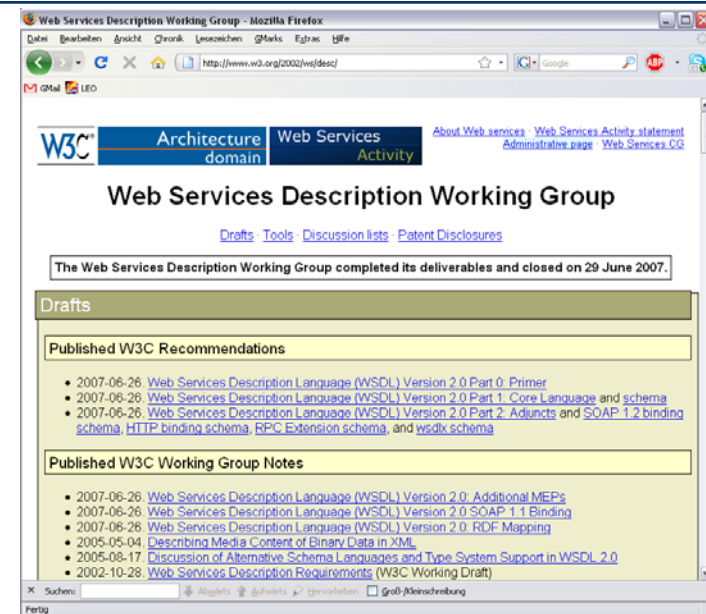
- Client möchte bestimmten Web Service nutzen
- Client benötigt hierfür:
 - Struktur des Aufrufes: Name, Parameter, Ergebnis, Fehlermeldungen
 - Übertragungsprotokoll und Web-Adresse
- genau dies wird mit WSDL beschrieben



- 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

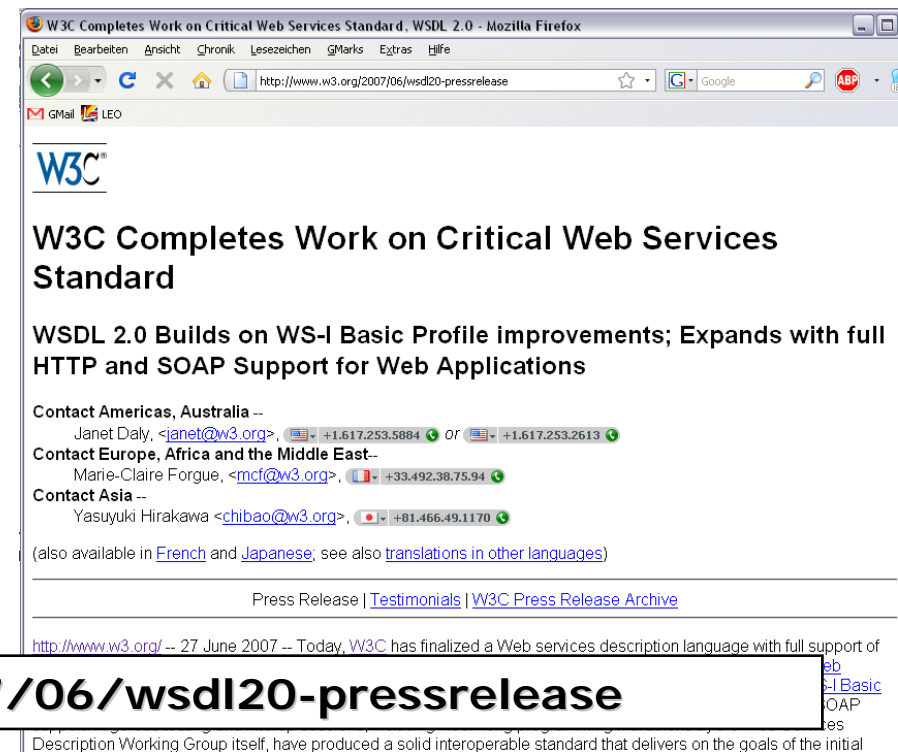
WSDL bei W3C

- 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, März 2003
 - Part 1: Core Language
 - Part 2: Message Patterns
 - Part 3: Bindings
- WSDL Version 2.0 → W3C Recommendation, Juni 2007
 - Part 0: Primer
 - Part 1: Core Language
 - Part 2: Adjuncts



**Keine Kompatibilität
zwischen WSDL 1.1. und WSDL 2.0**

- berücksichtigt Verbesserungen für WSDL1.1 von WS-I Basic Profile
- eingebaute Vererbung, import-Funktionen, verbesserte Beschreibung von Fehlern
- solider, interoperabler Standard
- trifft die Anforderungen von Web-Applikation-Entwicklern



- + Einfachere und flexiblere Beschreibung von Services and Service Wiederverwendung
- + Reicherer Syntax für Beschreibung von Aspekte der „Quality of Service“
- + Interface-Vererbung
- mehr Komplexität
- Keine Rückwärtskompatibilität
- (noch) nicht weit verbreitet

- **Apache Woden**

- WSDL 2.0 Parser & Validator
- Subprojekt von Apache Web Services Project
- Lesen, Bearbeiten, Erzeugen und Schreiben von WSDL-Dokumenten
- ursprünglich für WSDL 2.0 aber mit dem Ziel alle WSDL-Version zu unterstützen
- <http://ws.apache.org/woden/>



<http://ws.apache.org/>

Apache <Web Services /> Project

- **WSDL 1.1 to WSDL 2.0 Konverter**

- implementiert als XSLT 2.0 Stylesheet
- keine Garantie für ein valides WSDL 2.0 Dokument
- <http://www.w3.org/2006/02/WSDLConvert.html>



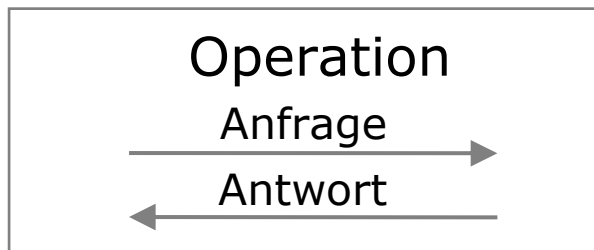


Prinzipieller Aufbau – allgemein

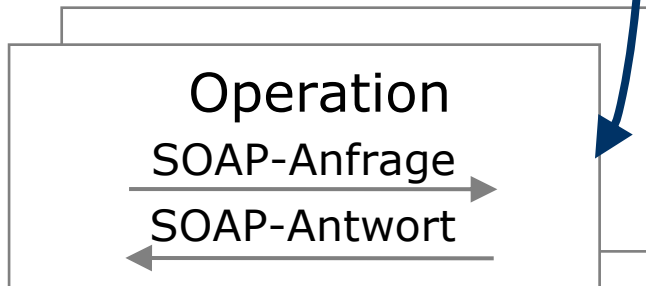


- beschreibt Netzwerkdienste als Kommunikationsendpunkte (Ports), die bestimmte Nachrichten über bestimmte Protokolle austauschen

abstrakte Schnittstelle



versch. Bindungen



abstrakte Schnittstelle

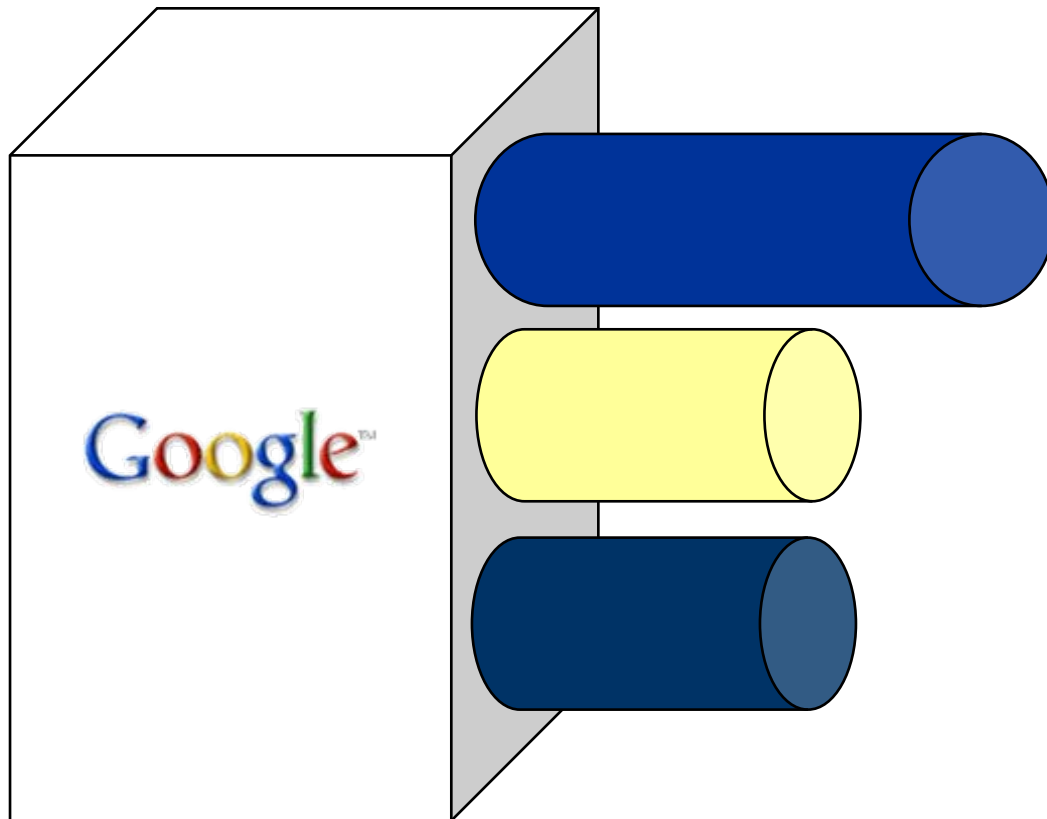
- Beschreibung der Schnittstelle unabhängig von
 - Nachrichtenformaten wie SOAP
 - Übertragungsprotokollen wie HTTP

Bindung

- Realisierung einer abstrakten Schnittstelle mit bestimmtem Nachrichtenformat und Übertragungsprotokoll

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

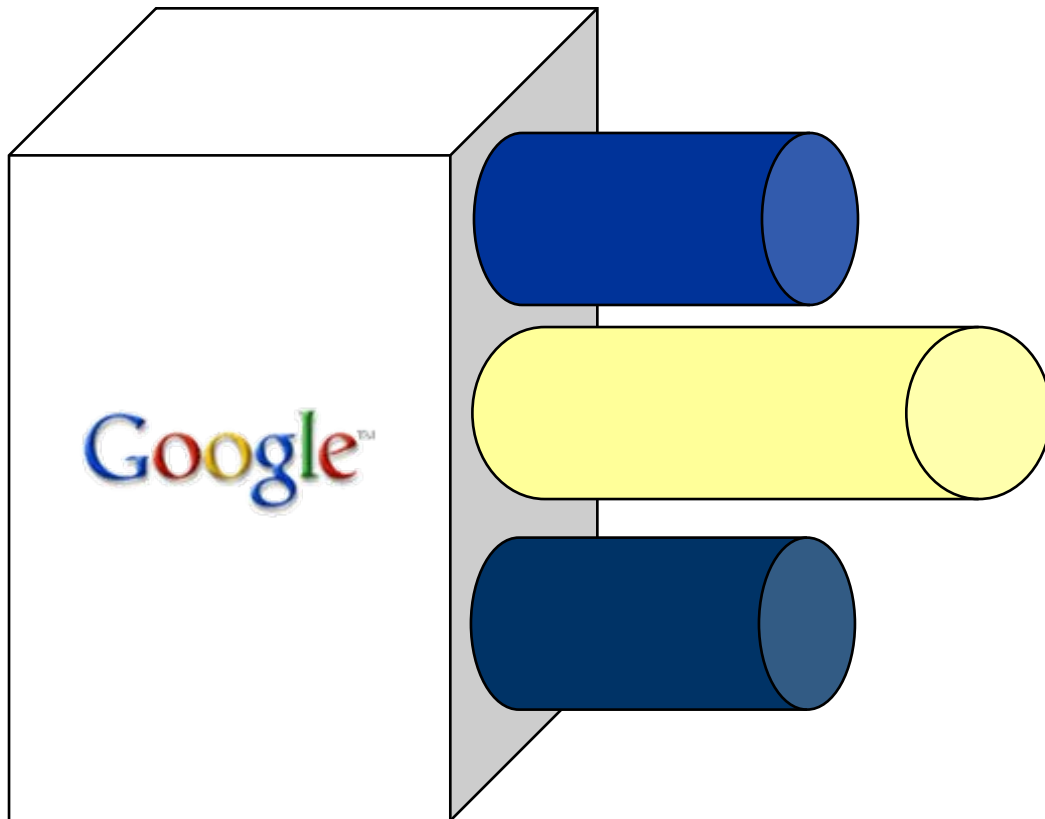
- eine Beschreibung (**WSDL**), aber 4 Zugriffsmöglichkeiten (**Bindungen**):
 1. SOAP/HTTP-POST
 2. SOAP/HTTP-GET (Rest)
 3. SOAP/SMTP (asynchron)
 4. HTML/HTTP-GET (Browser)



Dienst: Suche

Name der Operation:
doGoogleSearch

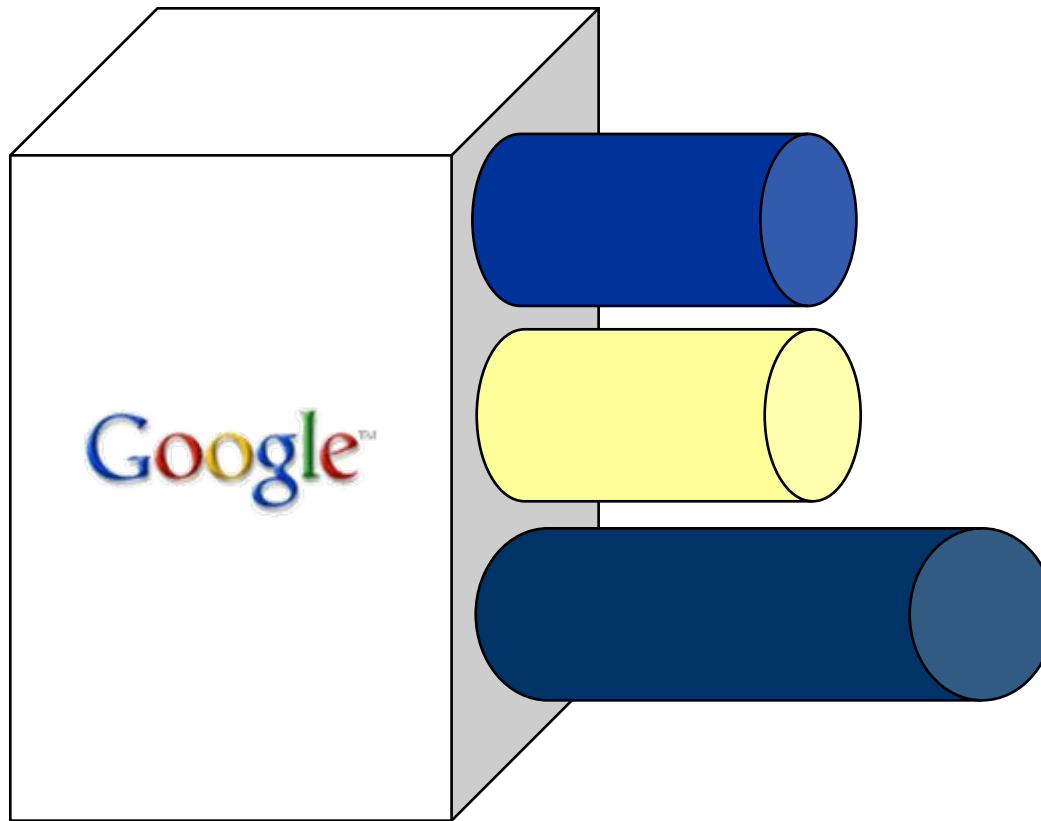
Rückgabe:
doGoogleSearchResponse



Dienst:
Zugriff auf Web-Cache

Name der Operation:
doGetCachedPage

Rückgabe:
doGetCachedPageResponse



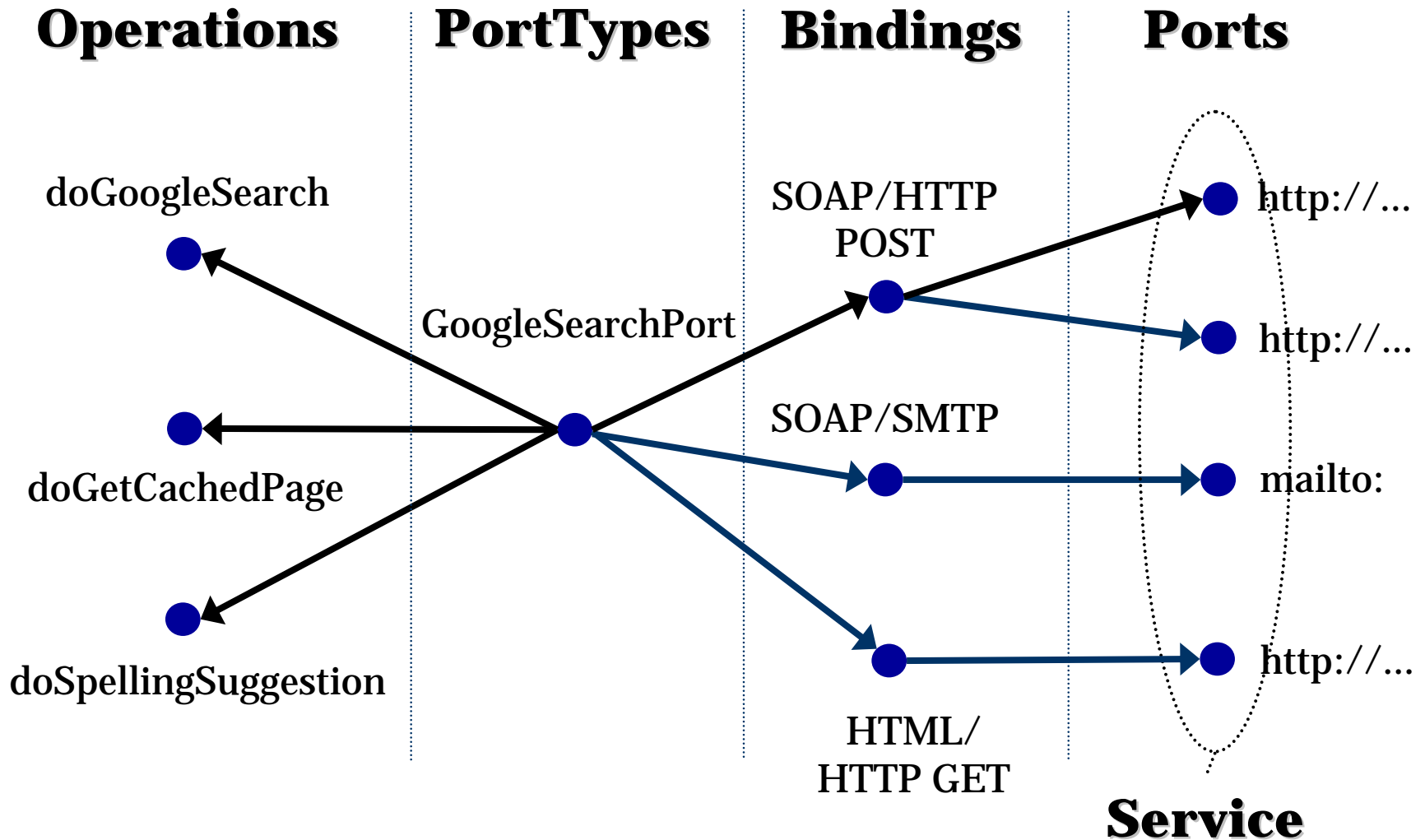
Dienst:
Rechtschreibkorrektur

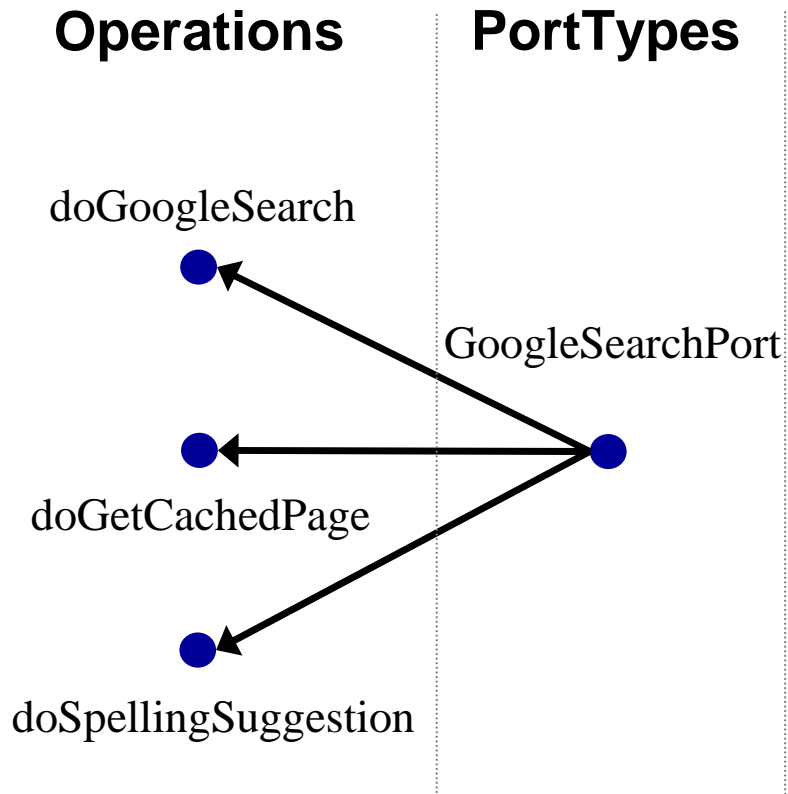
Name der Operation:
doSpellingSuggestion

Rückgabe:
doSpellingSuggestionResponse

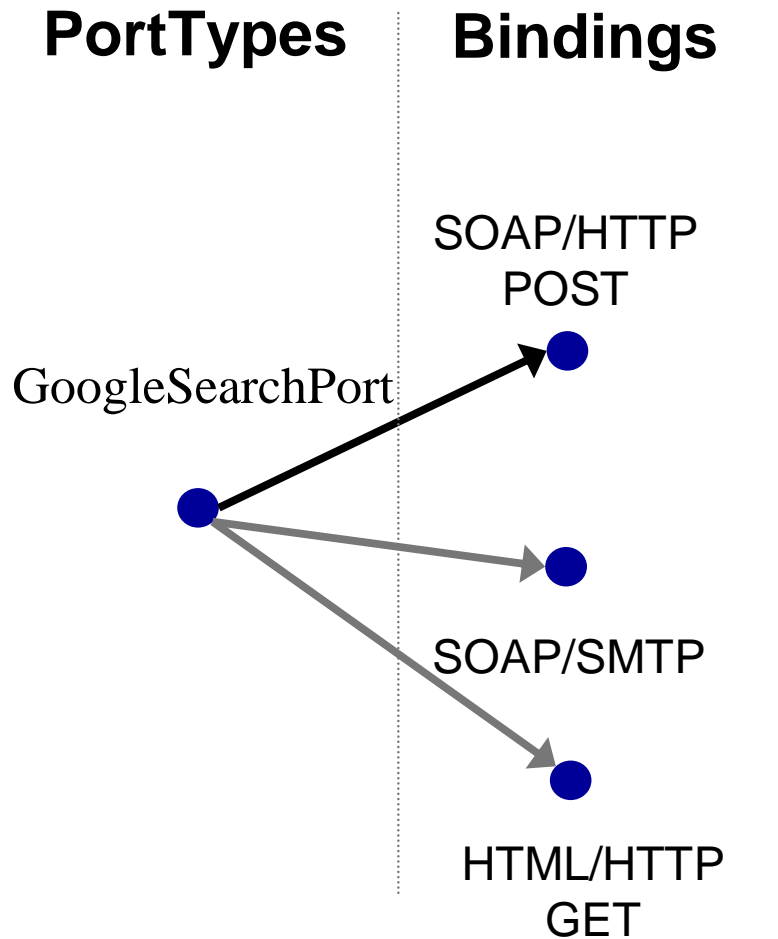
WSDL-Inhalt

- Was? → Typen, Messages, PortTypes (Interfaces)
 - Deklaration des 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?

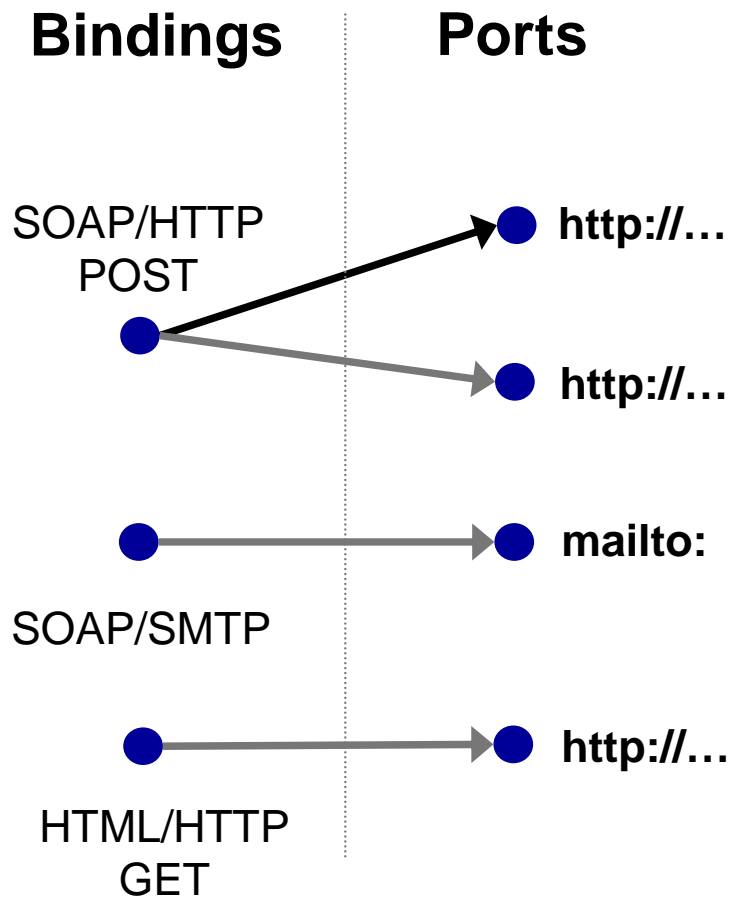




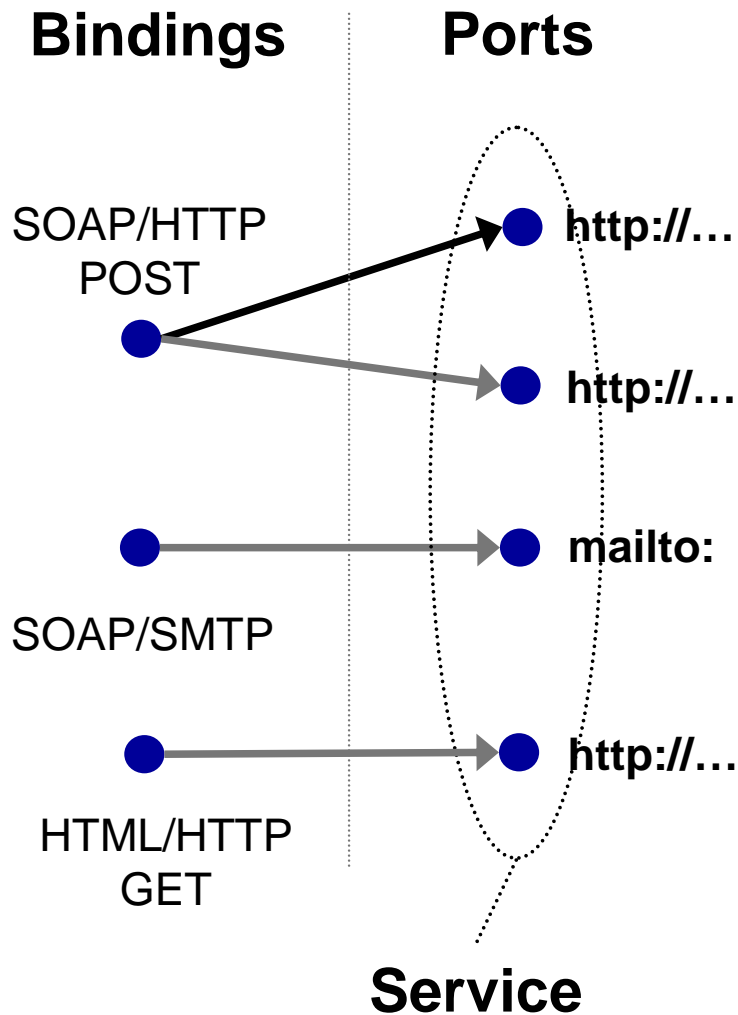
- **portType** (WSDL 1.1) = **interface** (WSDL 2.0)
- portType = Menge von abstrakten Operationen
- jede abstrakte Operation beschreibt Eingangs- und Ausgangsnachricht
- meist nur ein portType, aber in WSDL 1.1 auch mehrere möglich



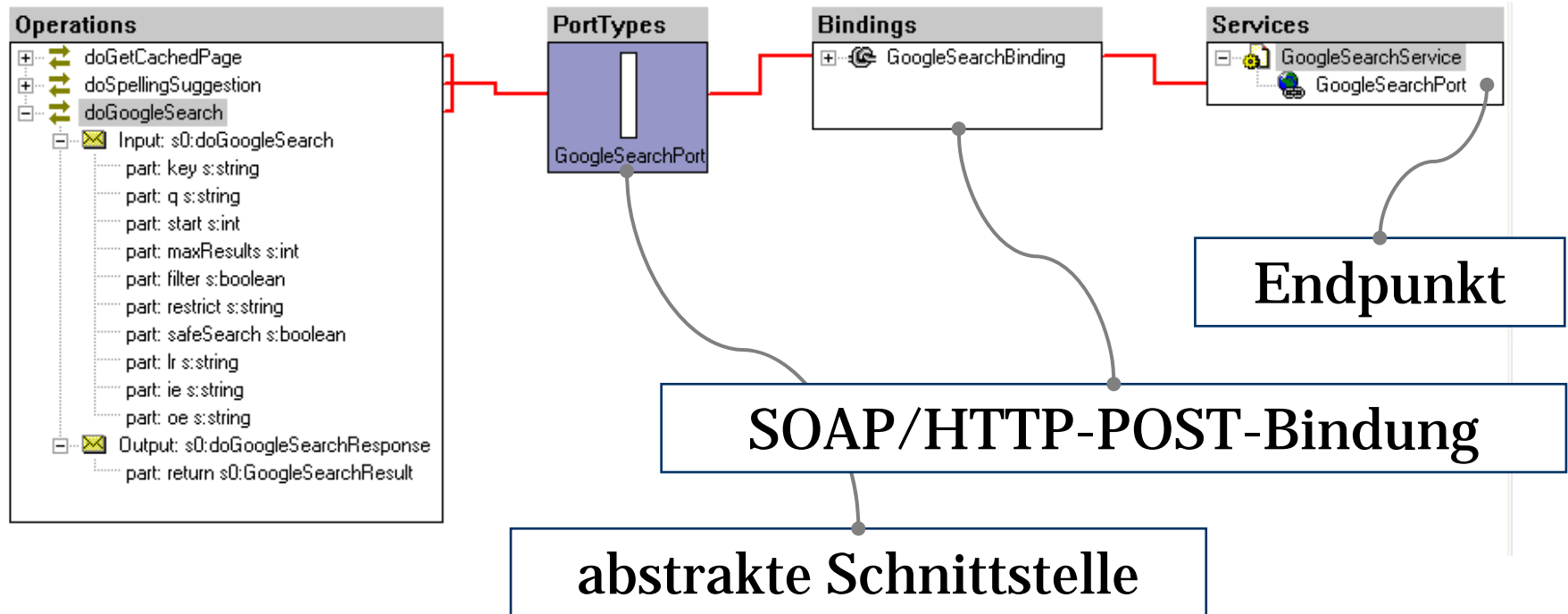
- in WSDL **binding** genannt
- für jede abstrakte Schnittstelle (**portType**) mindestens eine Bindung
- ein **portType** kann also mit **unterschiedlichen Bindungen** realisiert sein



- **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

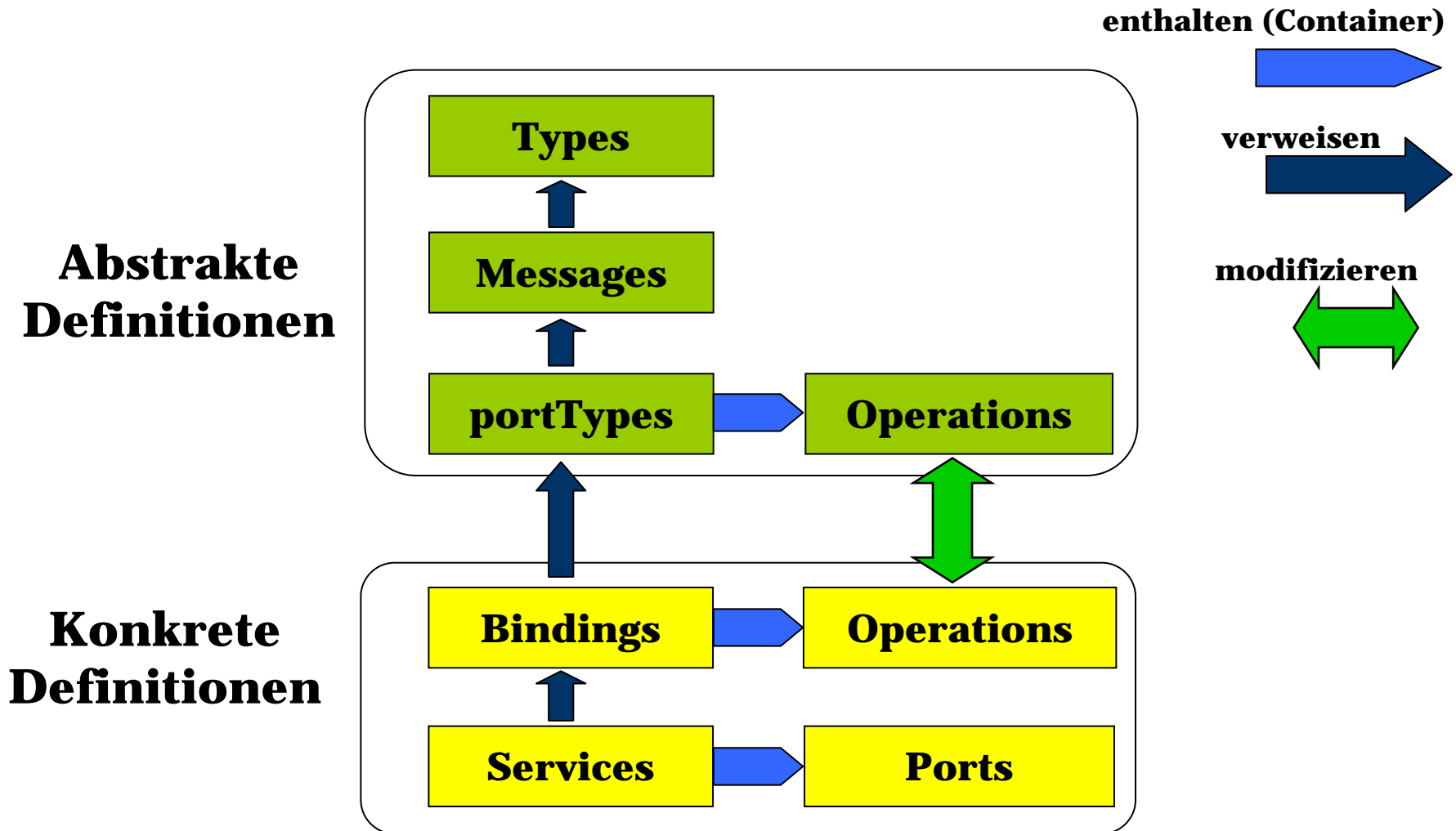


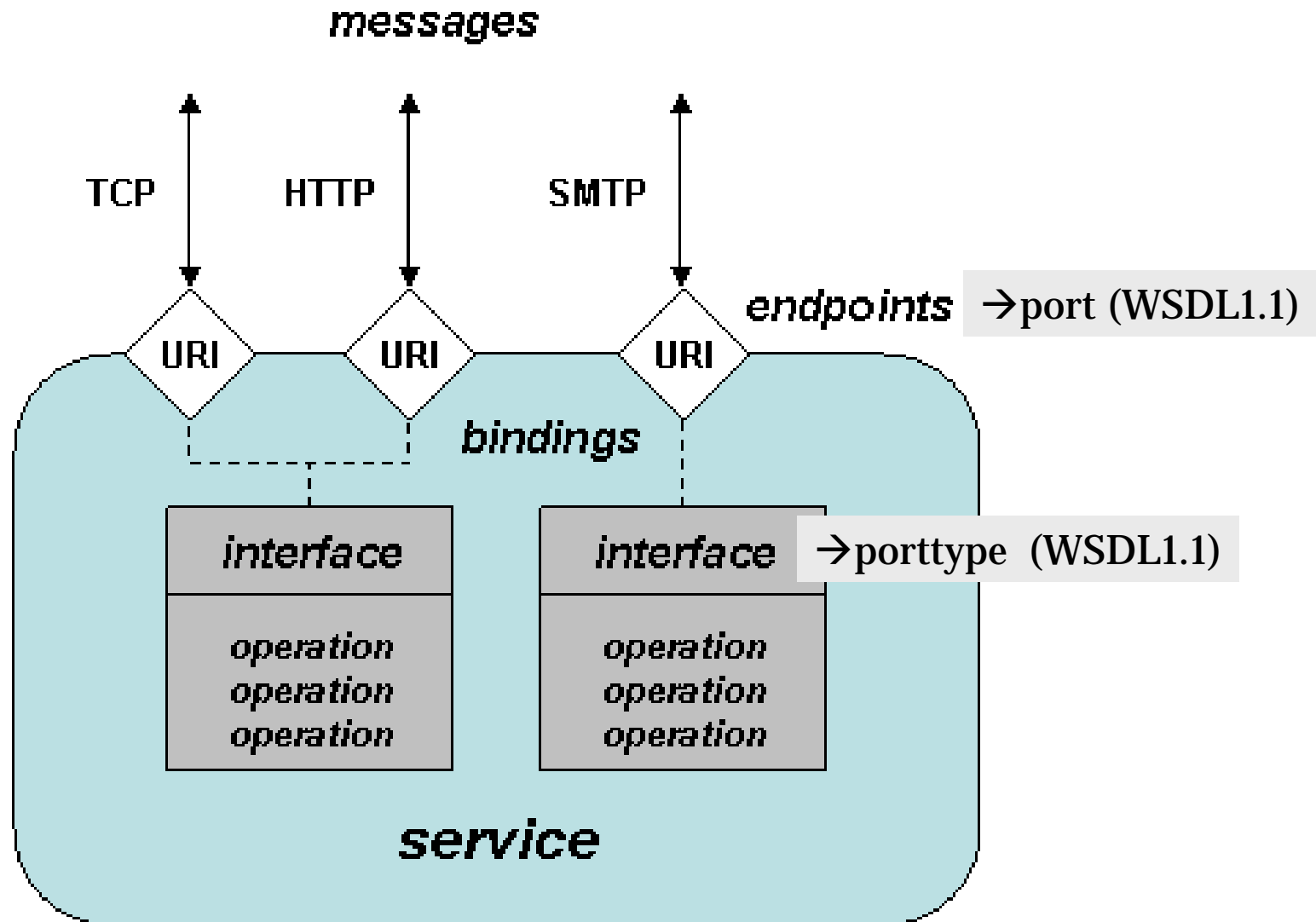
- Menge von **ports** bilden zusammen einen **Service**
- **ports** können in verschiedene Services gruppiert werden
- **ports** eines Service = semantisch äquivalente Alternativen



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

Element	Beschreibung
Konkrete Beschreibung	
<code><binding>...</binding></code>	<ul style="list-style-type: none">- Kommunikationsprotokoll, der beim Web Service benutzt wird- Gibt die Bindung(en) der einzelnen Operationen im portType-Abschnitt an
<code><service>...</service></code>	- gibt die Anschlussadresse(n) der einzelnen Bindungen an (Sammlung von einem oder mehreren Ports)





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

XML-Syntax von WSDL

```
<?xml version="1.0"?>
```

XML-Deklaration

```
<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

- **Wurzel-Element** `definitions` aus entsprechendem Namensraum
- **Namensraum** von `definitions` = Version
- WSDL-Beschreibung kann Ziel-Namensraum definieren
- ⇒ SOAP-Nachricht kann auf diesen Ziel-Namensraum verweisen

- Prefixe für Namenräume - Konvention

prefix	namespace URI	definition
wSDL	http://schemas.xmlsoap.org/wSDL/	WSDL namespace for WSDL framework.
soap	http://schemas.xmlsoap.org/wSDL/soap/	WSDL namespace for WSDL SOAP binding.
http	http://schemas.xmlsoap.org/wSDL/http/	WSDL namespace for WSDL HTTP GET & POST binding.
soapenc	http://schemas.xmlsoap.org/soap/encoding/	Encoding namespace as defined by SOAP 1.1 [8].
soapenv	http://schemas.xmlsoap.org/soap/envelope/	Envelope namespace as defined by SOAP 1.1 [8].
xsi	http://www.w3.org/2000/10/XMLSchema-instance	Instance namespace as defined by XSD [10].
xsd	http://www.w3.org/2000/10/XMLSchema	Schema namespace as defined by XSD [10].

...

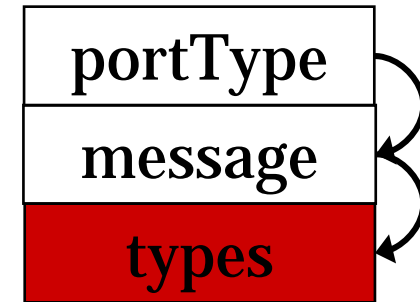
Quelle: <http://www.w3.org/TR/wSDL>



Prinzipieller Aufbau (1/4): Datenschema

<types>

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



types

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

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

portType

message

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.

<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



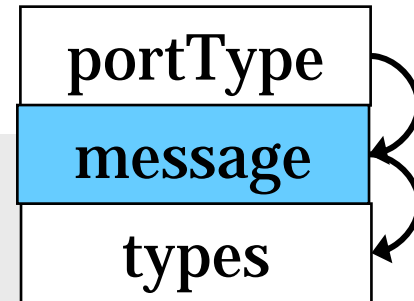
Prinzipieller Aufbau (2/4): Funktionalität

<message>

```

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

```



message

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

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

- **name** muss innerhalb der WSDL eindeutig sein
- setzen sich aus logischen Bestandteilen (**parts**) zusammen: $\#parts \geq 1$
- **part** kann z.B. ein Parameter eines RPCs sein
- jedes **part** hat eindeutigen Namen
- Reihenfolge der logischen Bestandteile unerheblich

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

zwei unterschiedliche Modellierungen

1. mehrere **parts**:

```
<message name="doGoogleSe  
  <part name="param1" elem  
  <part name="param2" elem  
</message>
```

2. ein **part** mit komplexen Daten

```
<message name="doGoogleSe  
  <part name="return" type=  
</message>
```

tns:complexType könnte z.B.

Unterschiede

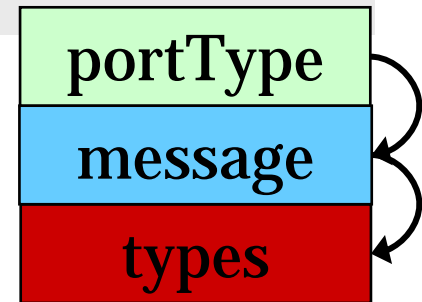
- **parts** immer reihenfolgeunabhängig
- parts können in Bindung unterschiedlich behandelt werden, z.B.:
ein part in Body der SOAP-Nachricht, ein anderes part in den Header

<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>

```



```
<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 (**operations**)

```
<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

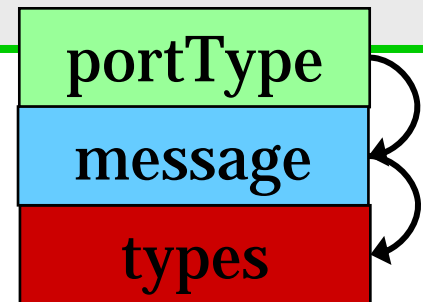
Abstrakte Nachricht vs. Datentyp

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

Datentyp

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

abstrakte
Nachricht



Datentyp / Nachricht / Porttyp

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

Definition des Datentyps

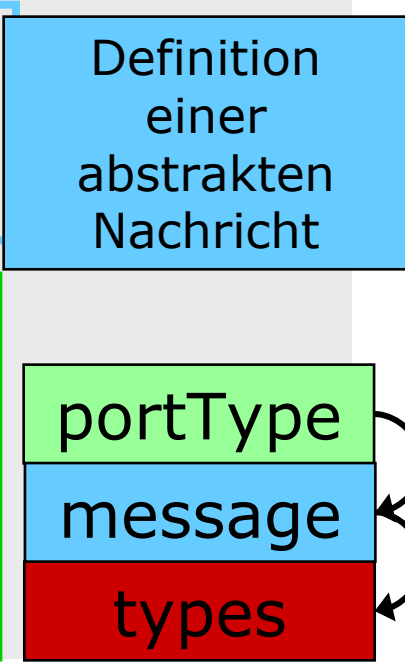
```
<message name="doGoogleSearch">...</message>
<message name="doGoogleSearchResponse">
  <part name="return" type="tns:GoogleSearchResult"/>
</message>
```

Definition einer abstrakten Nachricht

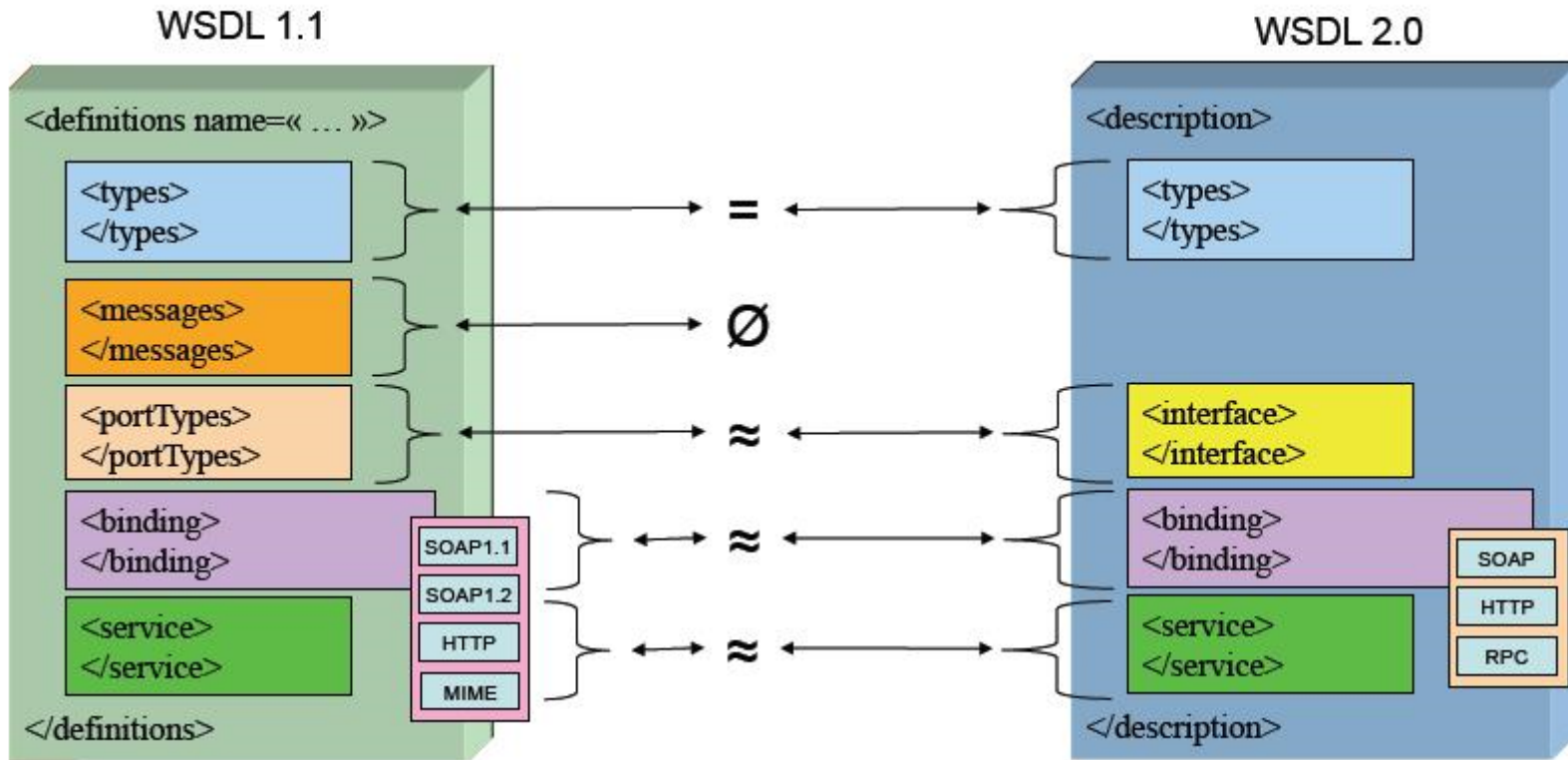
```
<portType>
  <operation name="doGoogleSearch">
    <input message="tns:doGoogleSearch"/>
    <output message="tns:doGoogleSearchResponse"/>
  </operation>
  ...
</portType>
```

Definition einer abstrakten Schnittstelle

portType
message
types



Struktur in WSDL 1.1 & 2.0



Quelle: <http://ssagara.blogspot.com/2009/01/converting-wsdl11-to-wsdl20-using-woden.html>

Einweg (oneway)



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

Anfrage-Antwort (request-response)



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

Benachrichtigung (notification)



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

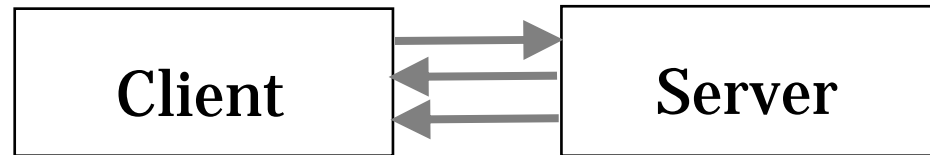
Benachrichtigung-Antwort (solicit-response)



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

- 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

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

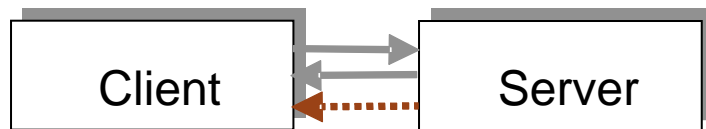


```
<operation name="...">  
  <input message="..."/>  
  <output message="..."/>  
  <output message="..."/>  
</operation>
```

In WSDL 1.1
nicht erlaubt!

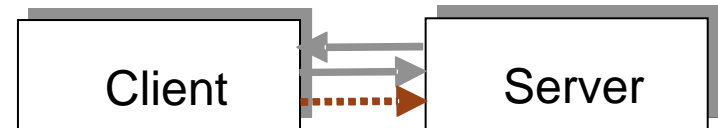
Anfrage-Antwort

```
<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

```
<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>
```

```
<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

```
<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-Aufbau

<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 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



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

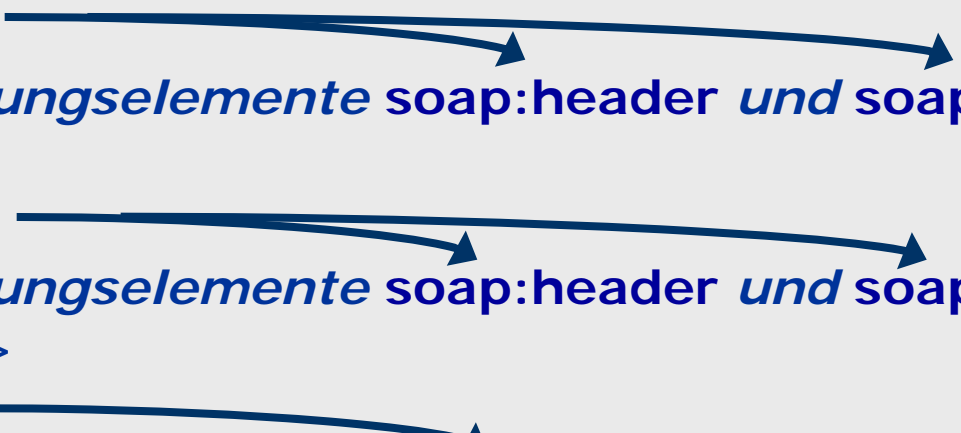
- spezifiziert MIME types (text/xml, multipart/related, ...)



Bindung: SOAP-Bindung



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



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

```
<binding name="GoogleSearchBinding" type="tns:GoogleSearchPort">
```

```
Erweiterungselement soap:binding
```

```
<operation name="doGoogleSearch">
```

```
Erweiterungselement soap:operation
```

```
<input>
```

```
Erweiterungselemente soap:header und soap:body
```

```
</input>
```

```
<output>
```

```
Erweiterungselemente soap:header und soap:body
```

```
</output>
```

```
<fault>
```

```
Erweiterungselement soap:fault
```

```
</fault>
```

```
</operation>
```

```
</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"**




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

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

```
<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)

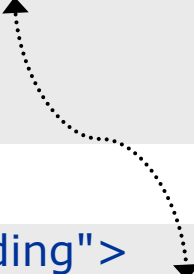
```
<operation name="doGoogleSearch">
  ...
  <input>
    <soap:header message="tns:doGoogleSearch" part="key"
      use="literal"/>
    <soap:body parts="q start maxResults ..." use="encoded" .../>
  </input>
  <output>...</output>
</operation>
```

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

```
<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

- 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

```
<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!

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



Vor- und Nachteile von WSDL

Vorteile von WSDL

sollen unterschiedliche Probleme lösen:

- **Interoperabilität**
 - Interoperabilität zwischen unterschiedlichen Implementierungsplattformen
 - gemeinsame Technologie für verschiedene Anwendungsgebiete
- **Kosten** → 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

schaffen 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 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?

WSDL

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

Übung diese Woche

- SOAP

Übung nächste Woche (letzte Übung)

- WSDL

Vorlesung nächste Woche

- Web Services in der Praxis & Ausblick