Freie Universität Berlin

WSDL

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

Block Web Services

Freie Universität Berlin

Vorlesungs -termine	Vorlesung (4 + 1 + 1)	Übung – 2 Termine	Übungs-termine
11.06.	Web Services, RPCs vs. Messaging		
18.06.	SOAP im Detail	SOAP	18./19.06
25.06. (heute)	WSDL im Detail	WSDL	
02.07.	Web Services in der Praxis & Ausblick		02./03.07
09.07.	Rückblick		
16.07.	Klausur		

Heutige Vorlesung

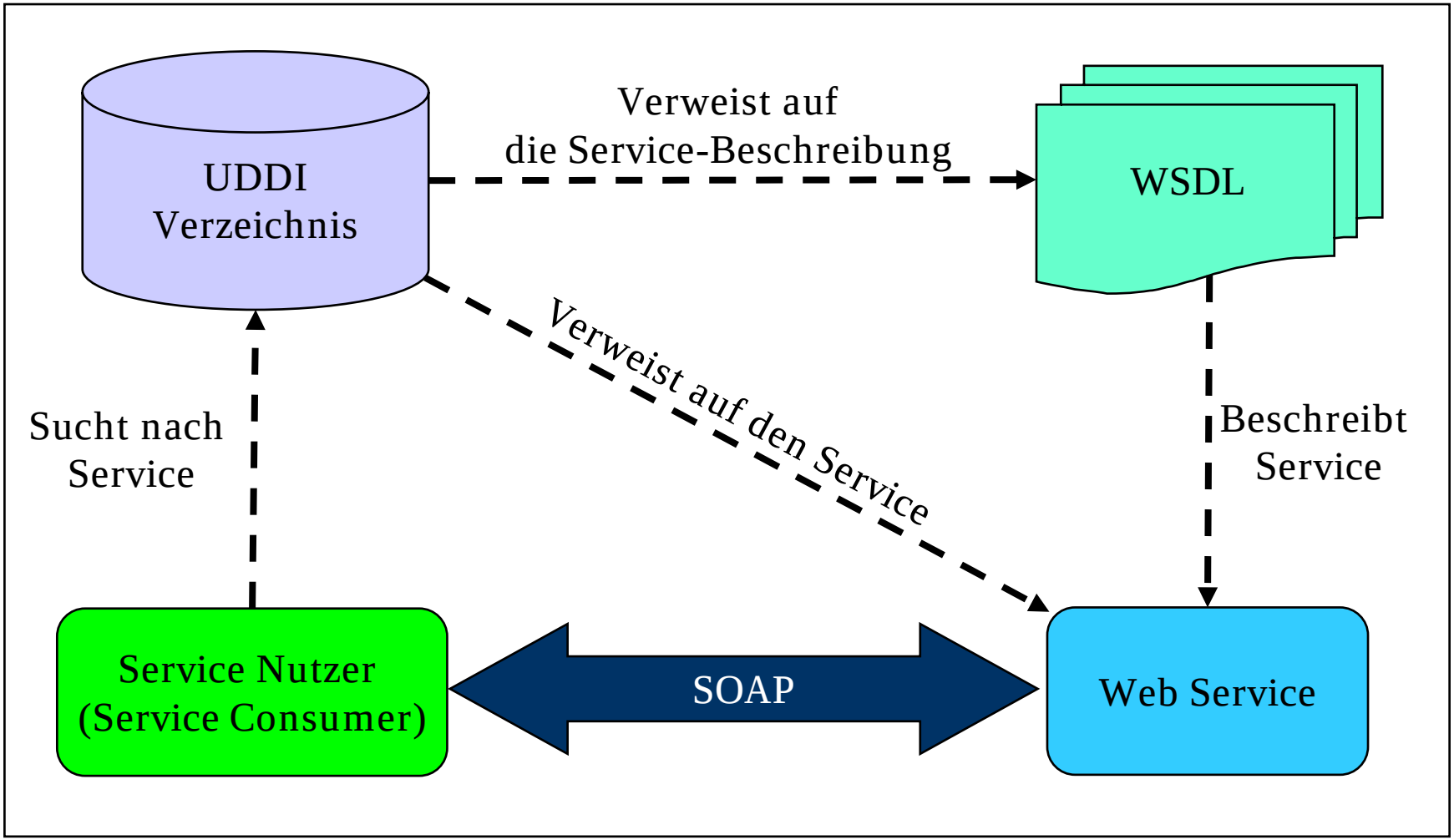
Freie Universität Berlin

letzte Woche

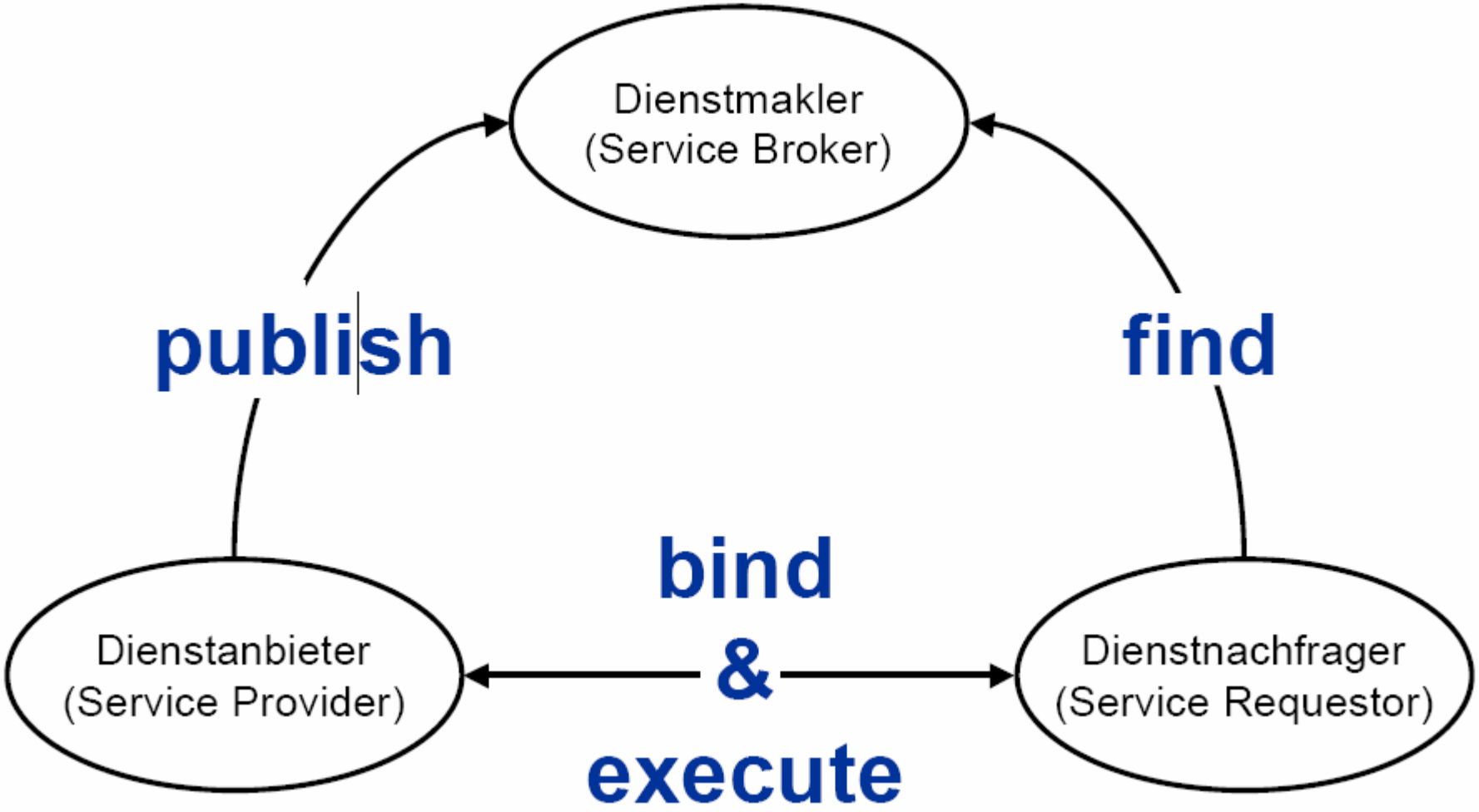
- ☑ prinzipieller Aufbau
- ☑ Kodierung von RPCs
- ☑ Verarbeitung & Übertragung
- ☑ Vor- und Nachteile

heutige Vorlesung → WSDL

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

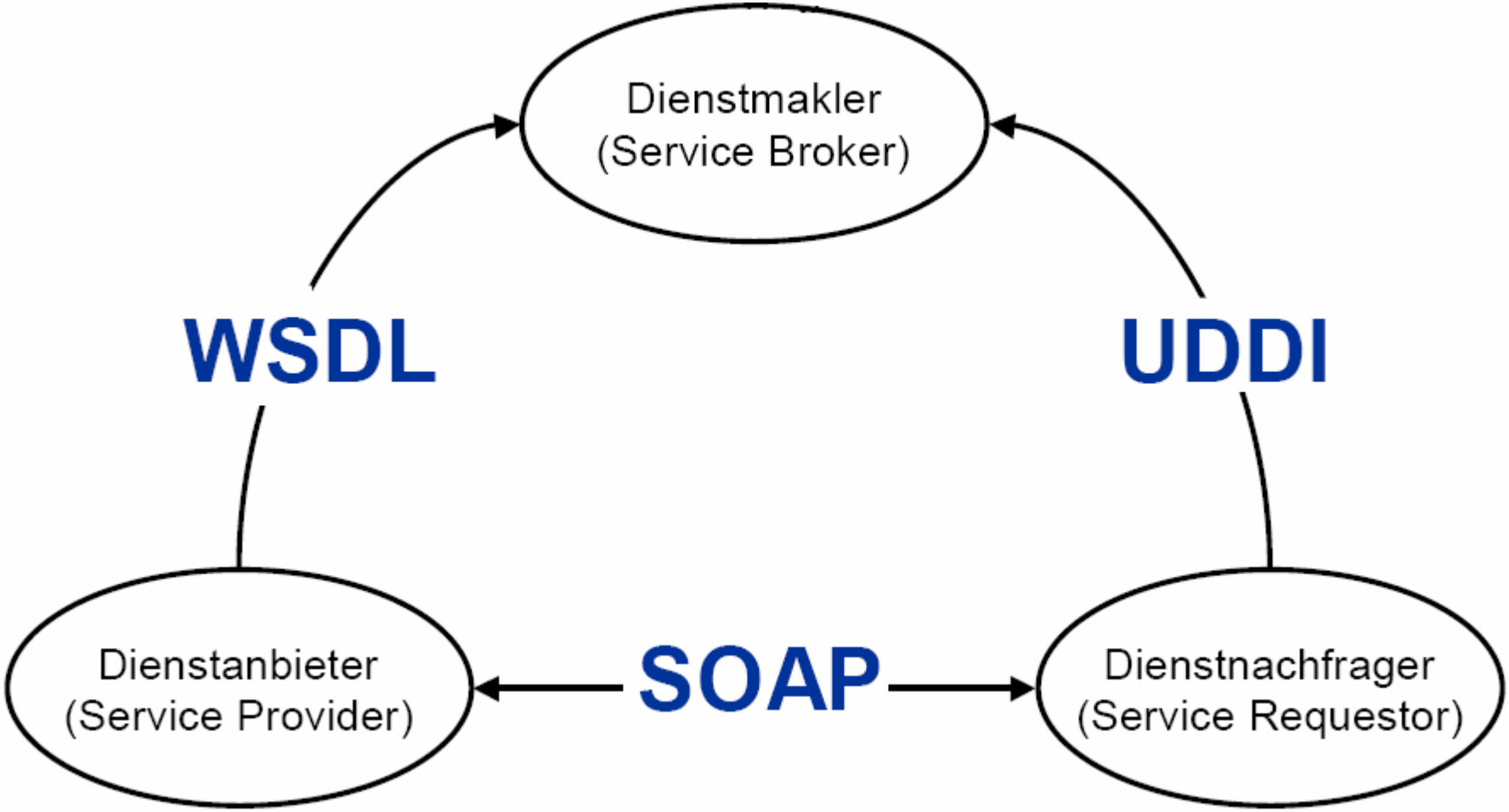

UDDI
Verzeichnis
Verweist auf
die Service-Beschreibung
WSDL
Verweist auf den Service
Sucht nach
Service
Beschreibt
Service
Service Nutzer
(Service Consumer)
SOAP
Web Service

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

Freie Universität Berlin

- 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

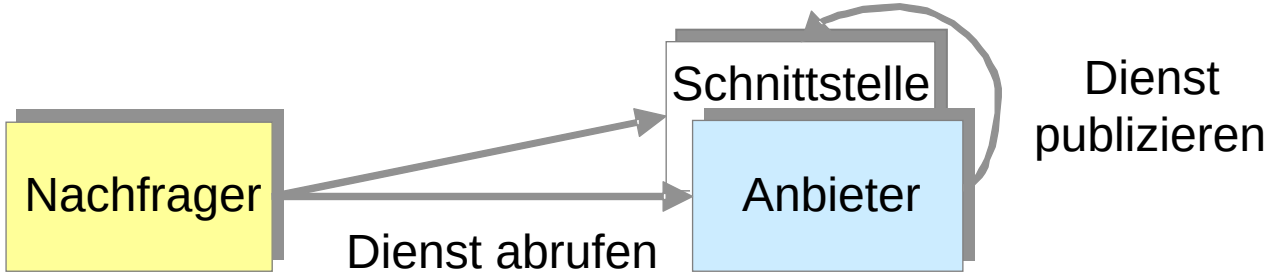
  → Syntax einer Schnittstelle kann bis ins kleinste Detail festgelegt werden

  → Beschreibung von Grundlegende Interaktionsmuster (wie Anfrage-Antwort)

Wozu dient WSDL?

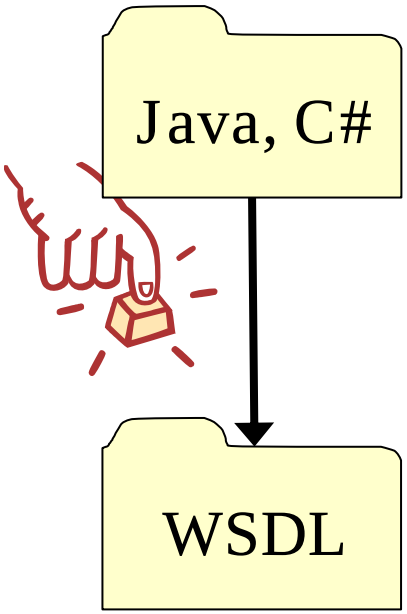
Freie Universität Berlin

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
- 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

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 Recommendation, Juni 2007
  - Part 0: Primer
  - Part 1: Core Language
  - Part 2: Adjuncts

http://www.hotcoding.com/xmls/webservice/33297.html

WSDL 2.0

- volle Unterstützung von HTTP & SOAP
- berücksichtigt Verbesserungen für WSDL 1.1 von WS-I Basic Profile
- eingebaute Vererbung, import-Funktionen, verbesserte Beschreibung von Fehlern
- solider, interoperabler Standard
- trifft die Anforderungen von Web-Applikation-Entwicklern

http://www.w3.org/2007/06/wsdl20-pressrelease

WSDL 2.0

http://www.w3.org/2007/06/wsdl20-pressrelease

W3C Completes Work on Critical Web Services Standard

WSDL 2.0 Builds on WS-I Basic Profile improvements; Expands with full HTTP and SOAP Support for Web Applications

Contact Americas, Australia --
Janet Daly, <janet@w3.org>, +1.617.253.5884 *or* +1.617.253.2613
Contact Europe, Africa and the Middle East--
Marie-Claire Forgue, <mcf@w3.org>, +33.492.38.75.94
Contact Asia --
Yasuyuki Hirakawa <chibao@w3.org>, +81.466.49.1170

(also available in French and Japanese; see also translations in other languages)

Press Release | Testimonials | W3C Press Release Archive

http://www.w3.org/ -- 27 June 2007 -- Today, W3C has finalized a Web services description language with full support of the primary protocol for the Web, HTTP, and the most frequently implemented Web services protocol, SOAP. Web Services Description Language (WSDL) 2.0 incorporates both the improvements for WSDL 1.1 found in the WS-I Basic Profile, and builds in inheritance, import functions, improved description of faults and errors, and full HTTP and SOAP support. Rigorous testing and interop sessions, including week-long programming marathons by the Web Services Description Working Group itself, have produced a solid interoperable standard that delivers on the goals of the initial WSDL 1.1 version, and also meets the needs of Web applications developers today.

"It's been a long time in development, but developers can see it's been worth the wait," explained Jonathan Marsh, co-chair of the W3C Web Services Description Working Group and Director of Mashup Technologies at WSO2. "In addition to the rigorous interoperability testing, we're pleased to have given developers the HTTP binding, which provides simple Web-friendly access to a service when the advanced features available in the SOAP stack, such as WS-Addressing, WS-Security, and WS-ReliableMessaging, aren't required."

WSDL 2.0 Development Began with Stringent Interoperability Requirements, Finished with Conversion Tools and Support in Emerging Web Standards

Wide interest in SOAP 1.1 and WSDL 1.1 marked the beginning of Web Services pursuits. The hope of interoperability in the initial, non-standardized versions was not immediately fulfilled in spite of the large number of developers who moved to that model for applications development. WS-I, for example, formed to create profiles based on non-standard versions of the popular Web Services technologies.

Against this backdrop, the W3C Membership urged the Consortium to develop an interoperable WSDL standard. Through significant issue tracking by the Working Group itself, and through cooperative work with the WS-I Basic Profile Working Group, it was possible to identify initial improvements and build from there for both SOAP 1.2 and WSDL 2.0.

Those who have built implementations that conform to the WS-I Basic Profile may use a converter to WSDL 2.0, thus easing the transition to the new standard and enabling use of new features.

Further, WSDL 2.0 support is built into critical emerging Web services standards, including Semantic Annotations for WSDL (SAWSDL) and WS-Policy 1.5, both anticipated in September 2007.

WSDL 2.0 Lets Developers Choose WS Application Development Models; HTTP or SOAP

Tools für WSDL 2.0

Freie Universität Berlin

- Apache Woden
  - WSDL 2.0 Parser & Validator
  - Subprojekt von Apache Web Services Project
  - Lesen, Bearbeiten, Erzeugen und Schreiben WSDL Dokumente
  - ursprünglich für WSDL 2.0 aber mit Ziel alles WSDL-Version zu unterstützen

    → http://ws.apache.org/woden/

- WSDL 1.1 to WSDL 2.0 Konverter
  - implementiert als XSLT 2.0 Stylesheet
  - keine Garantie für ein valides WSDL 2.0 Dokument

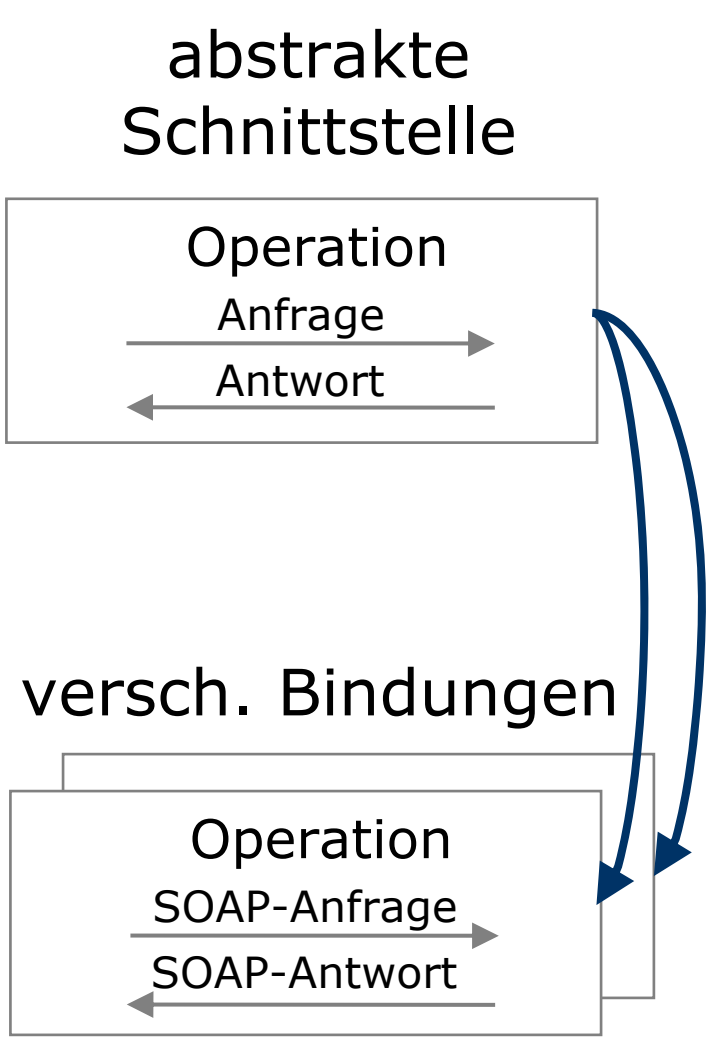
    → http://www.w3.org/2006/02/WSDLConvert.html

Prinzipieller Aufbau

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

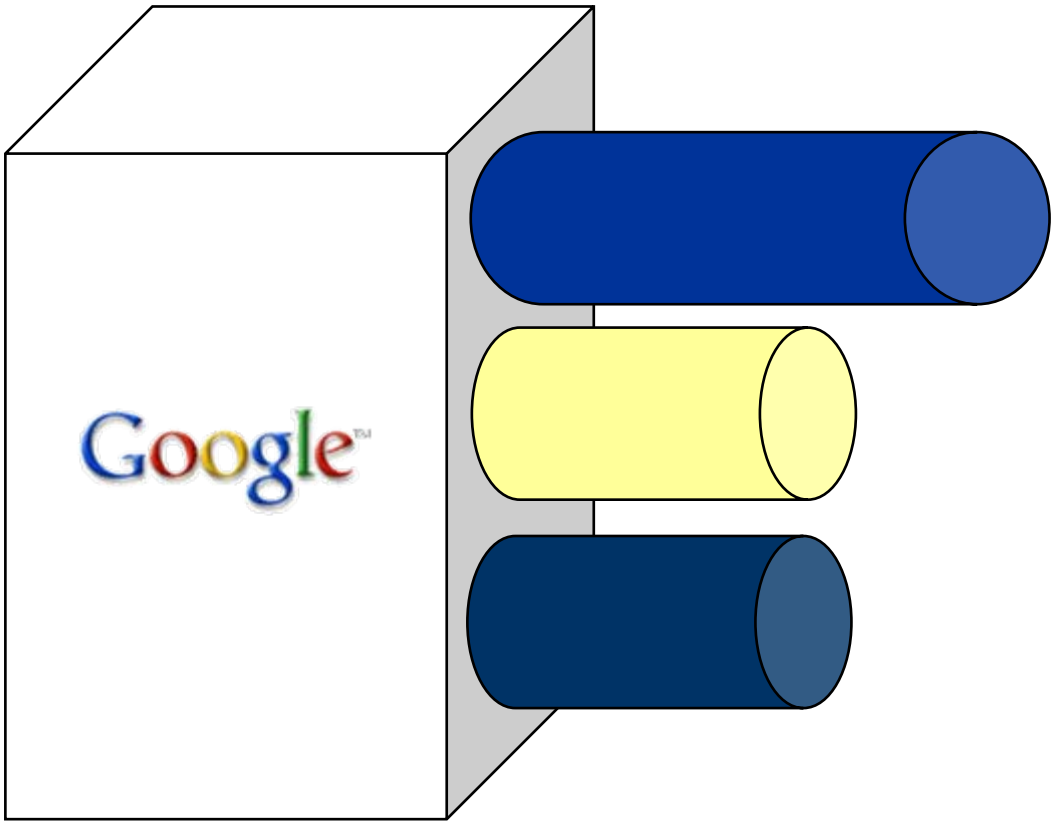
Beispiel Google (fiktiv)

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 Zugriffs-möglichkeiten (**Bindungen**):

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

Web Service von Google

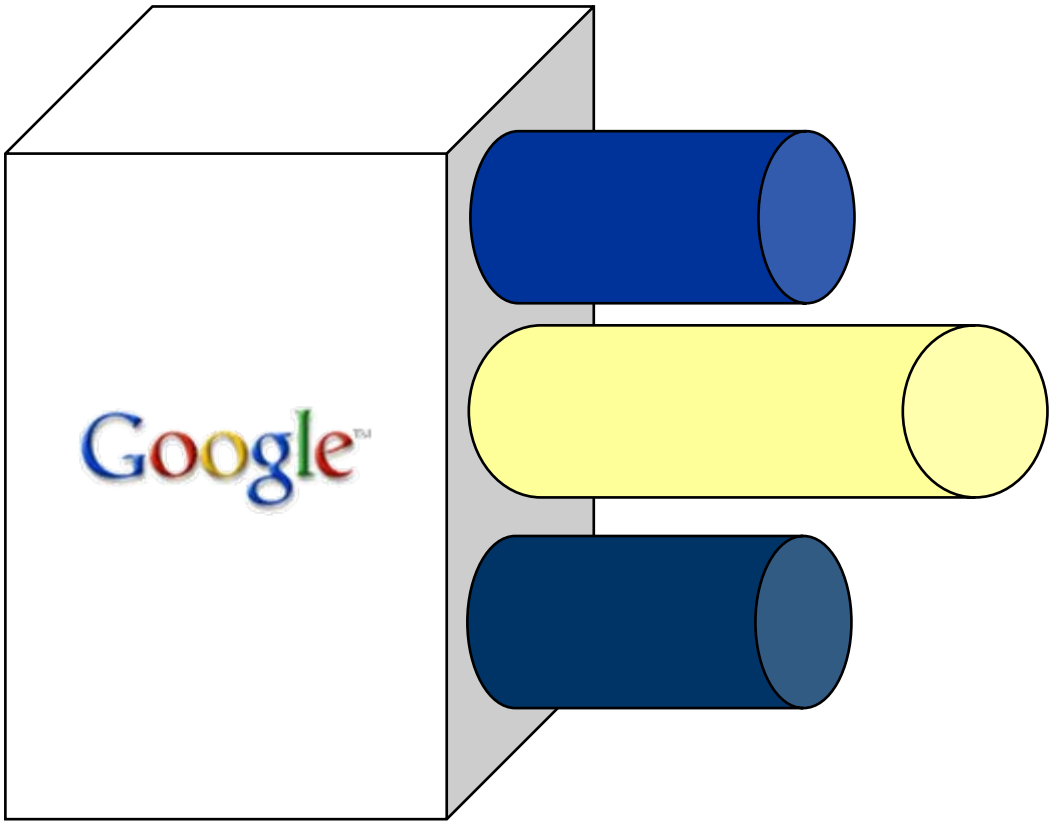


Dienst: Suche

Name der Operation:
doGoogleSearch

Rückgabe:
doGoogleSearchResponse

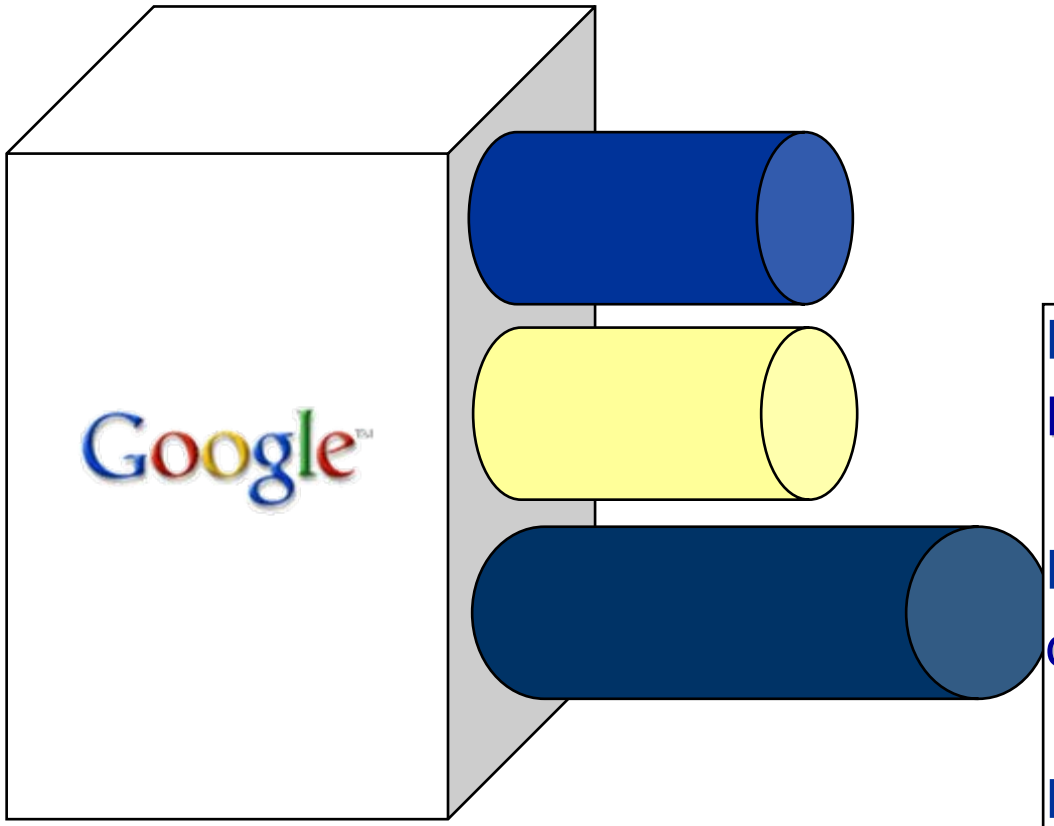
Web Service von Google



Dienst:
Zugriff auf Web-Cache

Name der Operation:
doGetCachedPage

Rückgabe:
doGetCachedPageResponse



Dienst:
Rechtschreibkorrektur

Name der Operation:
doSpellingSuggestion

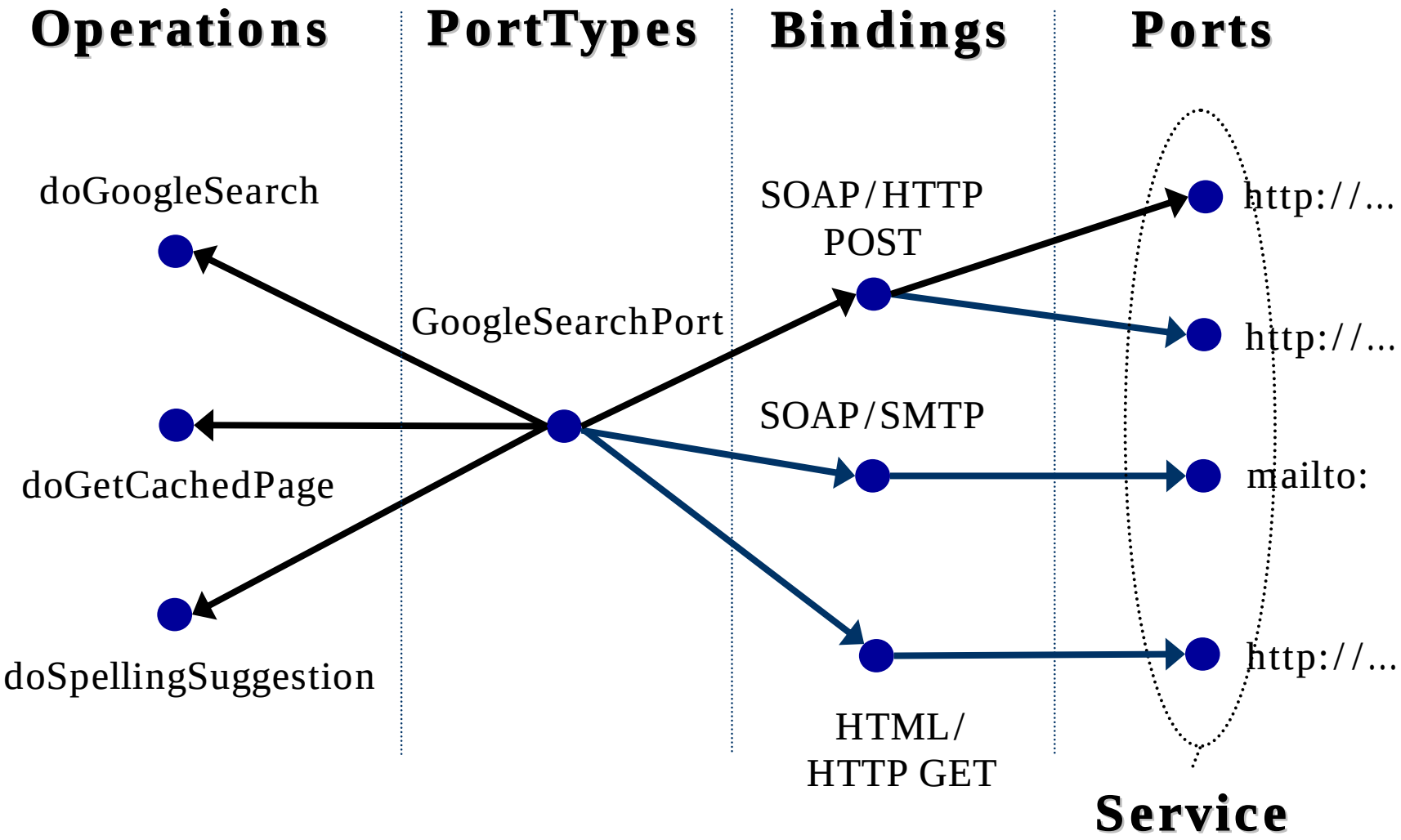
Rückgabe:
doSpellingSuggestionResponse

WSDL-Inhalt

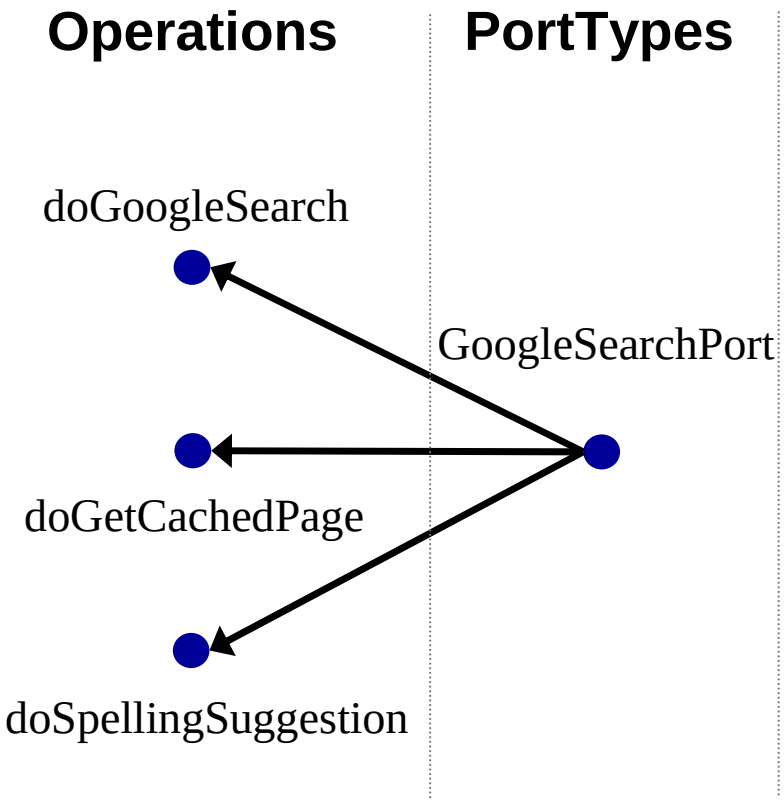
Freie Universität Berlin

- 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?

Grundstruktur

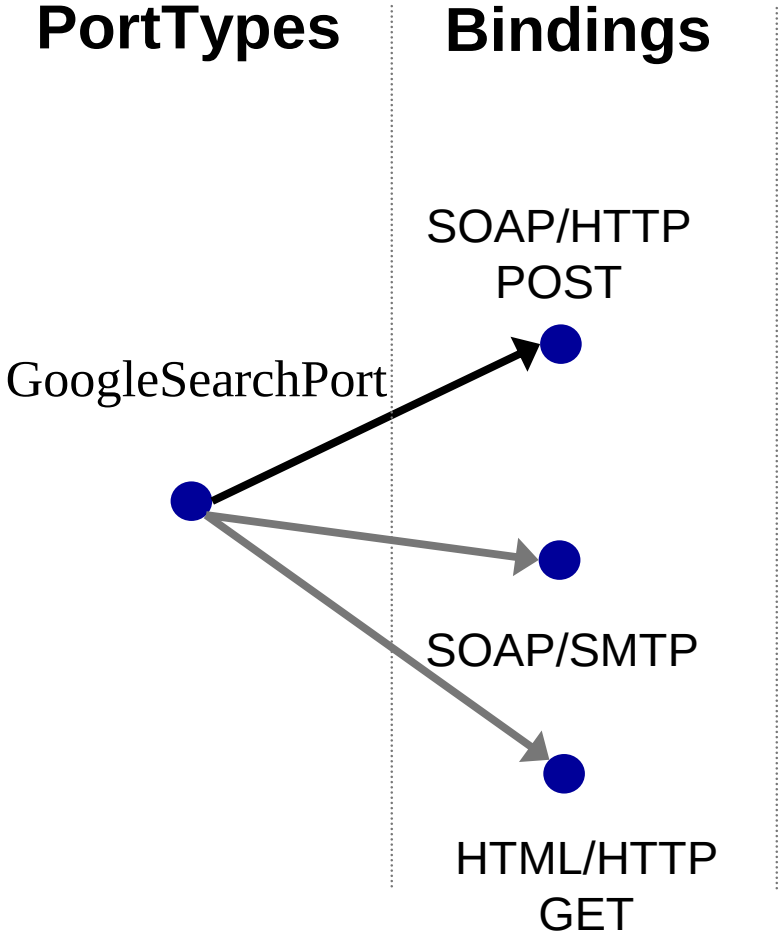


Abstrakte Schnittstellen



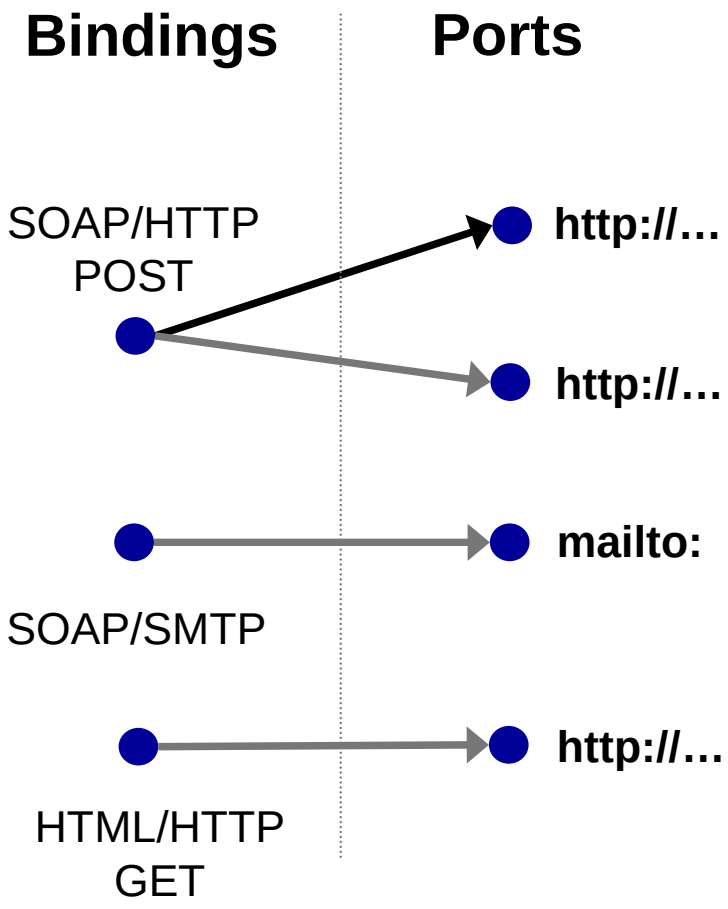
- **portType** (WSDL 1.1) **=** **interface** (WSDL 2.0)
- portType = Menge von abstrakten Operationen
- jede abstrakte Operation beschreibt Eingangs- und Ausgangsnachricht
- meist nur ein portType, aber in WSDL 1.1 auch mehrere möglich

Bindungen



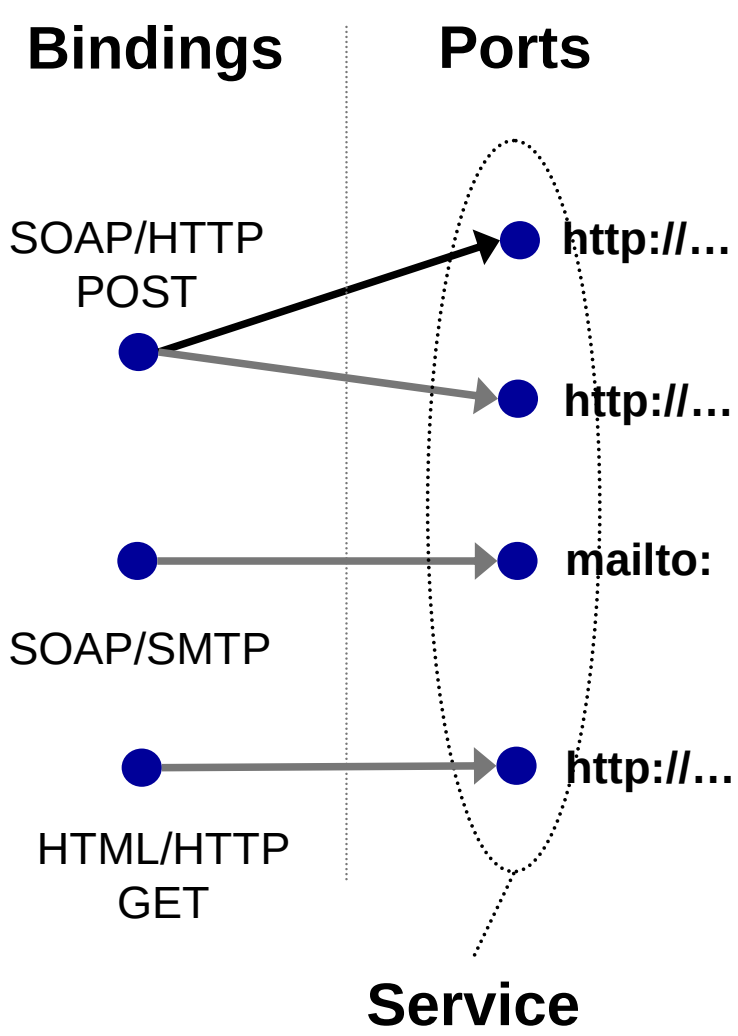
- in WSDL **binding** genannt
- für jede abstrakte Schnittstelle (**portType**) mindestens eine Bindung
- ein **portType** kann also mit unterschiedlichen Bindungen realisiert sein

Kommunikationsendpunkte



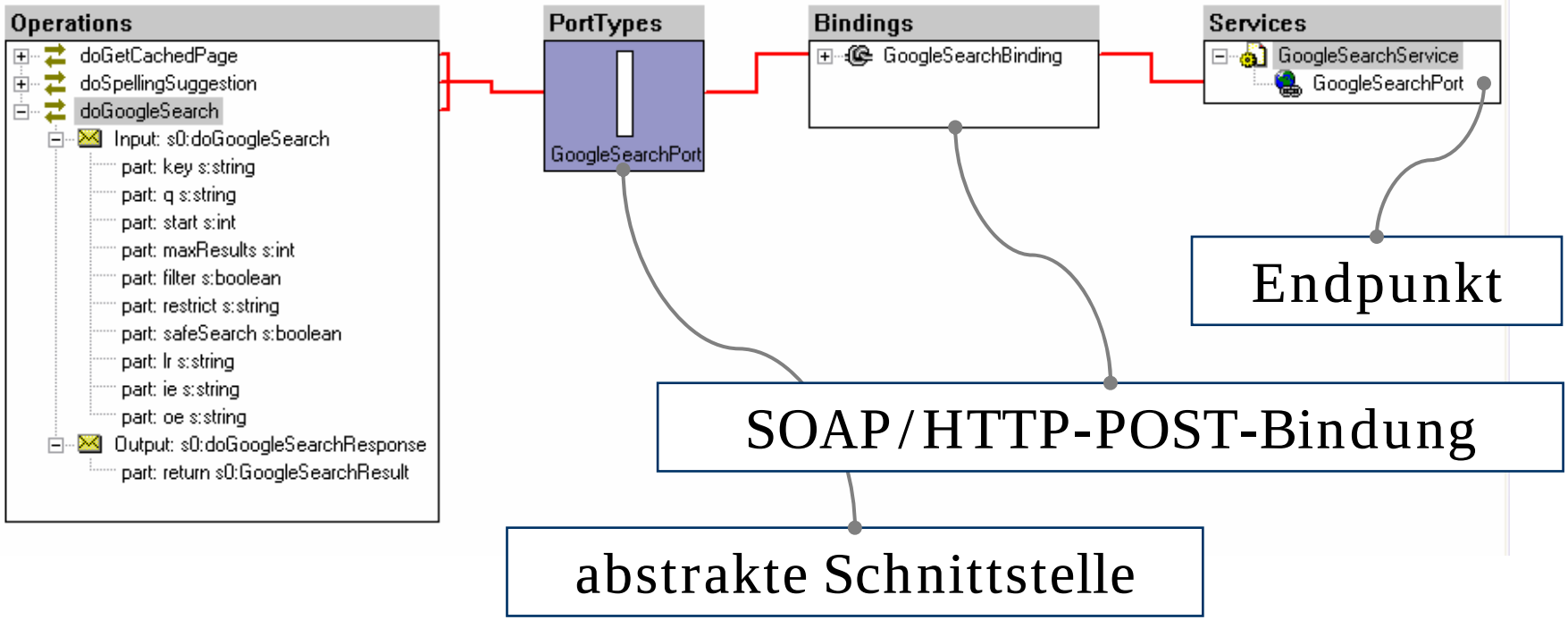
- **port** (WSDL 1.1) **=** **endpoint** (WSDL 2.0)
- **port** = Bindung + Web-Adresse
- für jede Bindung (**binding**) mindestens ein **port**
- ein **binding** kann also über unterschiedliche Web-Adressen zugänglich sein

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)

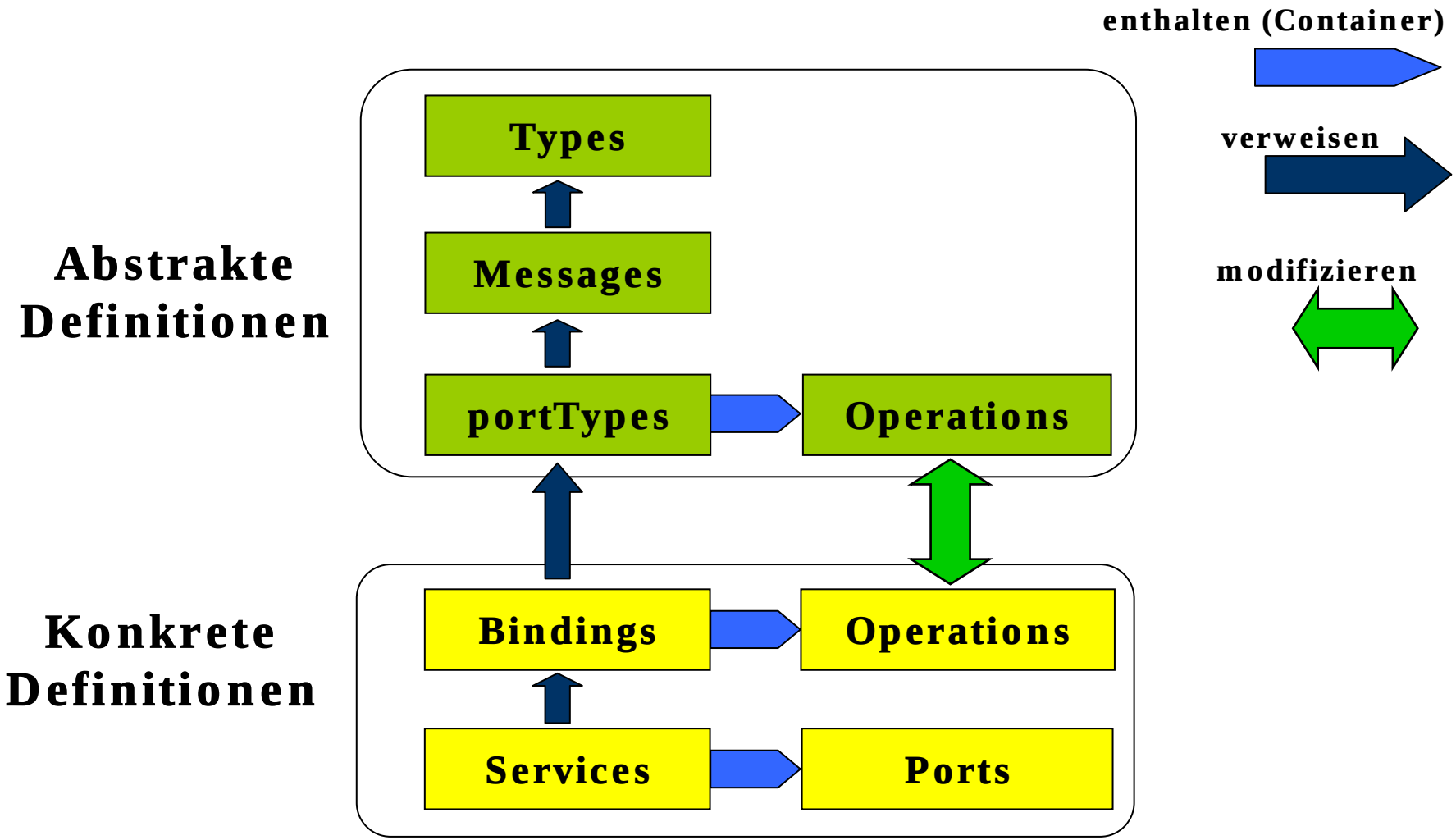
Freie Universität Berlin

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

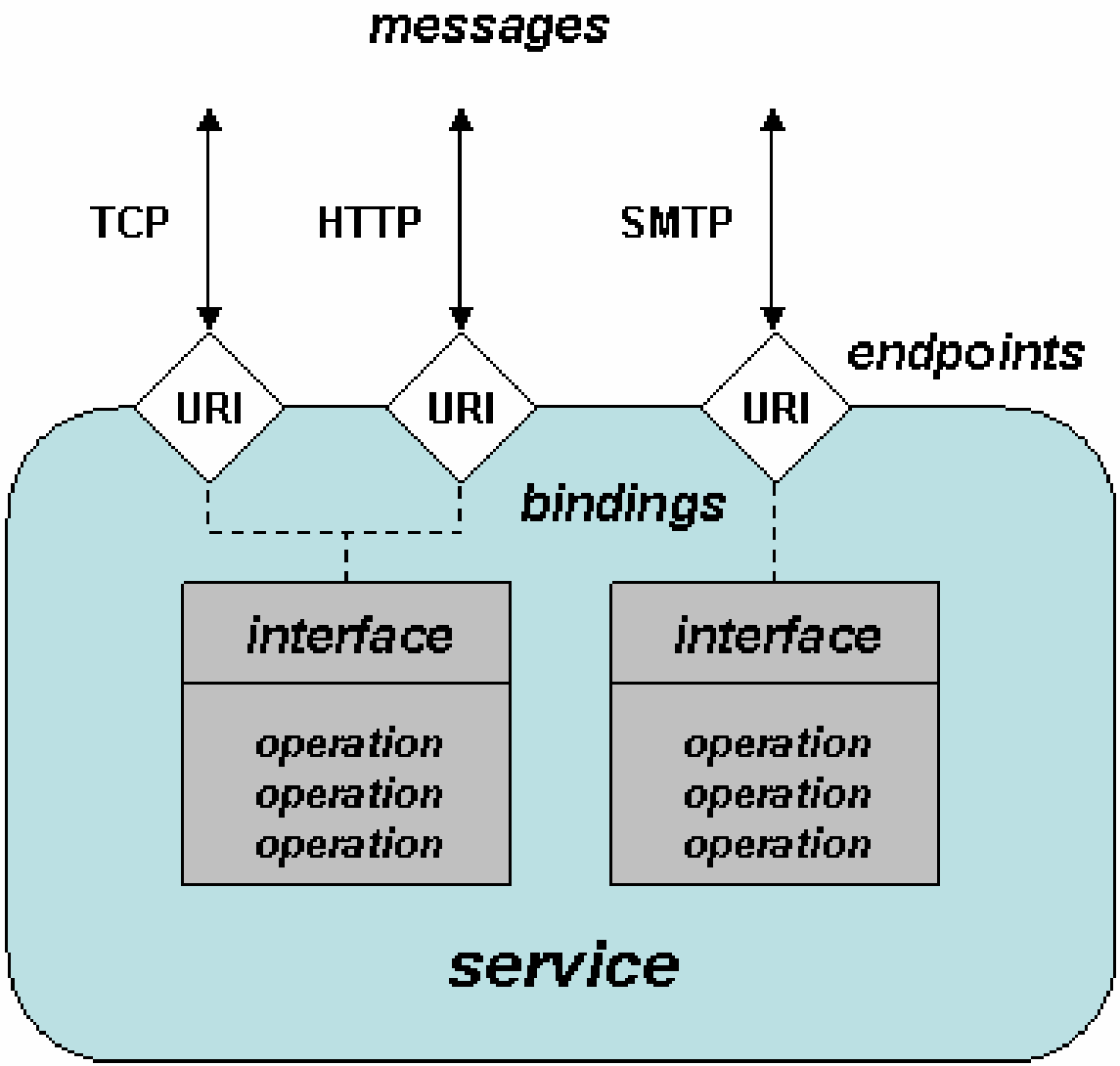
WSDL 1.1. – Elemente (2)

Element	Beschreibung
Konkrete Beschreibung	
`<binding>…</binding>`	- Kommunikationsprotokoll, 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 mehreren Ports)

Abstrakte vs. Konkrete Definition



WSDL – Schematische Darstellung



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

XML-Syntax von WSDL

Freie Universität Berlin

```
<?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

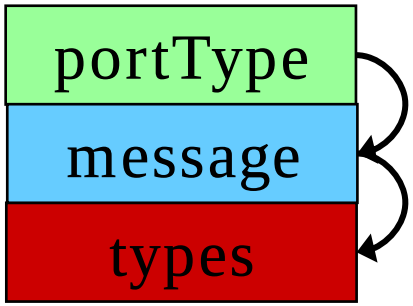
- **Wurzel-Element definitions** aus entsprechendem Namensraum
- **Namensraum** von **definitions** = Version
- WSDL-Beschreibung kann Ziel-Namensraum definieren

⇨ SOAP-Nachricht kann auf diesen Ziel-Namensraum verweisen

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">…</message>
   …
</definitions>
```



types

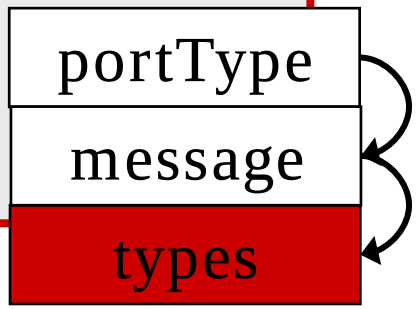
- Definition von Datentypen
- werden für Spezifikation von abstrakten Nachrichten verwendet

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

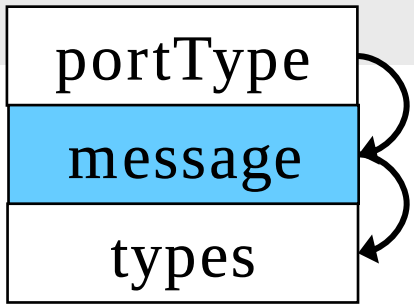
```
<types>
  <xsd:schema  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               xmlns:tns="urn:GoogleSearch"
               targetNamespace="urn:GoogleSearch">
    <xsd:complexType name="GoogleSearchResult">
      <xsd:all>
       <xsd:element name="estimatedTotalResultsCount"  type="xsd:int"/>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

Freie Universität Berlin

```
<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

Freie Universität Berlin

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

Freie Universität Berlin

zwei unterschiedliche Modellierungen

1. mehrere parts:

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

2. ein part mit komplexen Date

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

tns:complexType könnte z.B

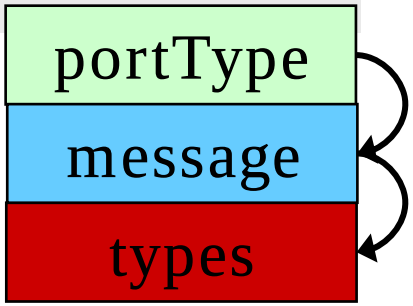
Unterschiede

- 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>
```



Definition der abstrakten Schnittstelle

Freie Universität Berlin

```
<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

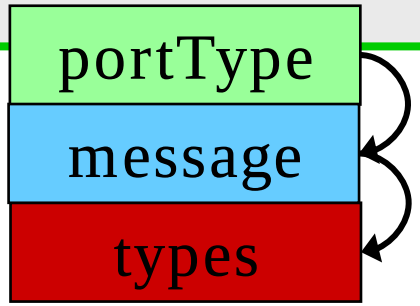
Freie Universität Berlin

```
<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

Freie Universität Berlin

```
<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

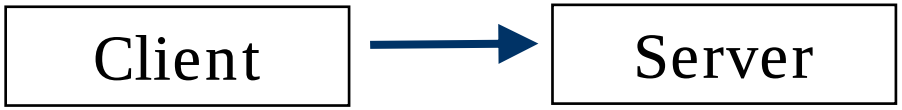
```
<portType>
   <operation name="doGoogleSearch">
      <input message="tns:doGoogleSearch"/>
      <output message="tns:doGoogleSearchResponse"/>
   </operation>
  …
</portType>
```

Definition einer abstrakten Schnittstelle

portType
message
types

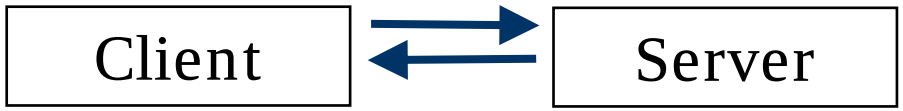
Interaktionsmuster (2)

Einweg (oneway)




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

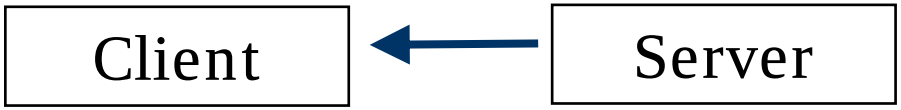
Anfrage-Antwort (request-response)




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

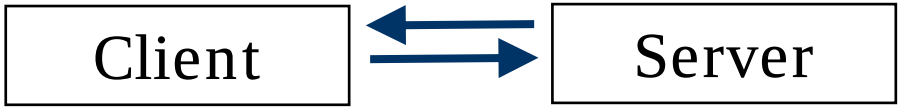
Freie Universität Berlin

Benachrichtigung (notification)




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

Benachrichtigung-Antwort (solicit-response)




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

Abstrakte (!) Interaktionsmuster

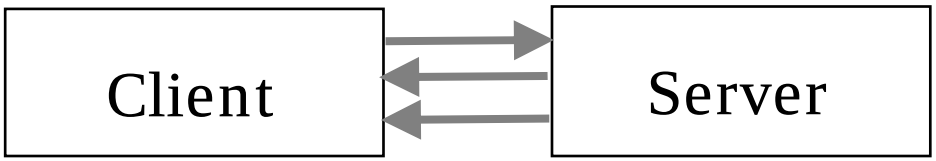
Freie Universität Berlin

- 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

Freie Universität Berlin

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




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

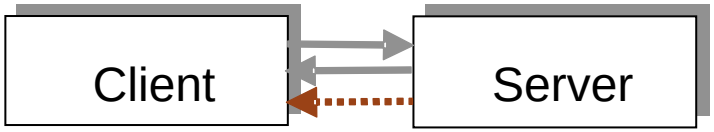
In WSDL nicht erlaubt!

Fehlermeldungen

Freie Universität Berlin

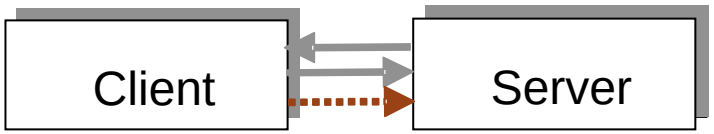
Anfrage-Antwort

```
<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

Freie Universität Berlin

- 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

```
<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

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

```
<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

Bindung allgemein: soap:binding

```
<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>
```

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

Freie Universität Berlin

```
<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>
```

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.
- <u>wichtig</u>: 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

```
<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

Prinzip

- Gruppierung einzelner Komponente nach verschiedenen Gesichtspunkten
- Einbinden von Dokumentteilen
  - 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
  - wird 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**

  → 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

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