Freie Universität Berlin

SOAP & WSDL

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

Heutige Vorlesung

Freie Universität Berlin

letzte Woche

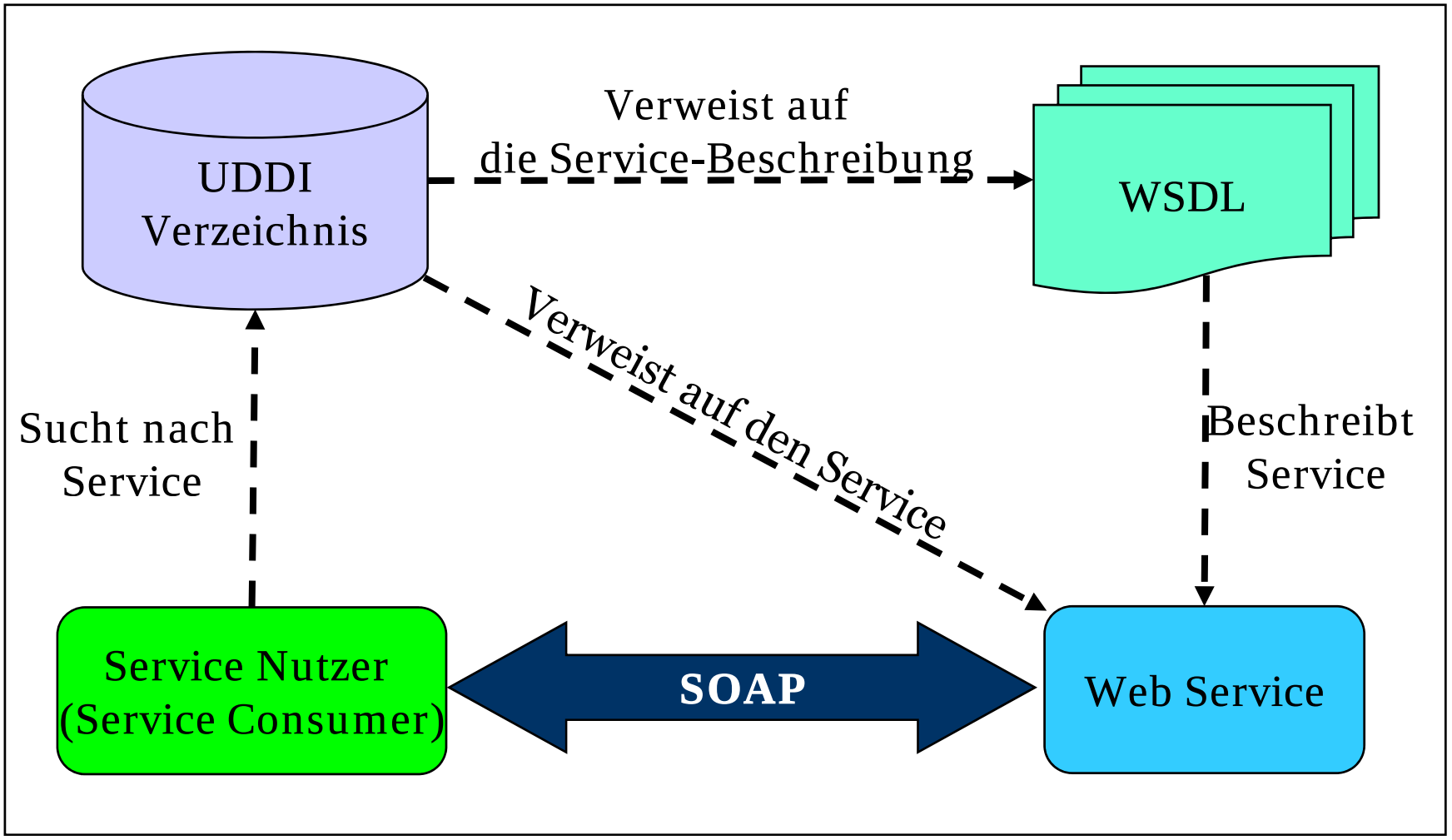
- ☑ Was sind Web Services?
- ☑ Was ist SOAP? / Was ist WSDL?
- ☑ Anwendungen
- ☑ RPC vs. Messaging

heutige Vorlesung → SOAP

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

heutige Vorlesung → WSDL

- prinzipieller Aufbau (Datenschemata, Funktionalität, Protokollbindung, Service-Aufbau)
- standardisierte Bindungen (SOAP & HTTP)
- 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

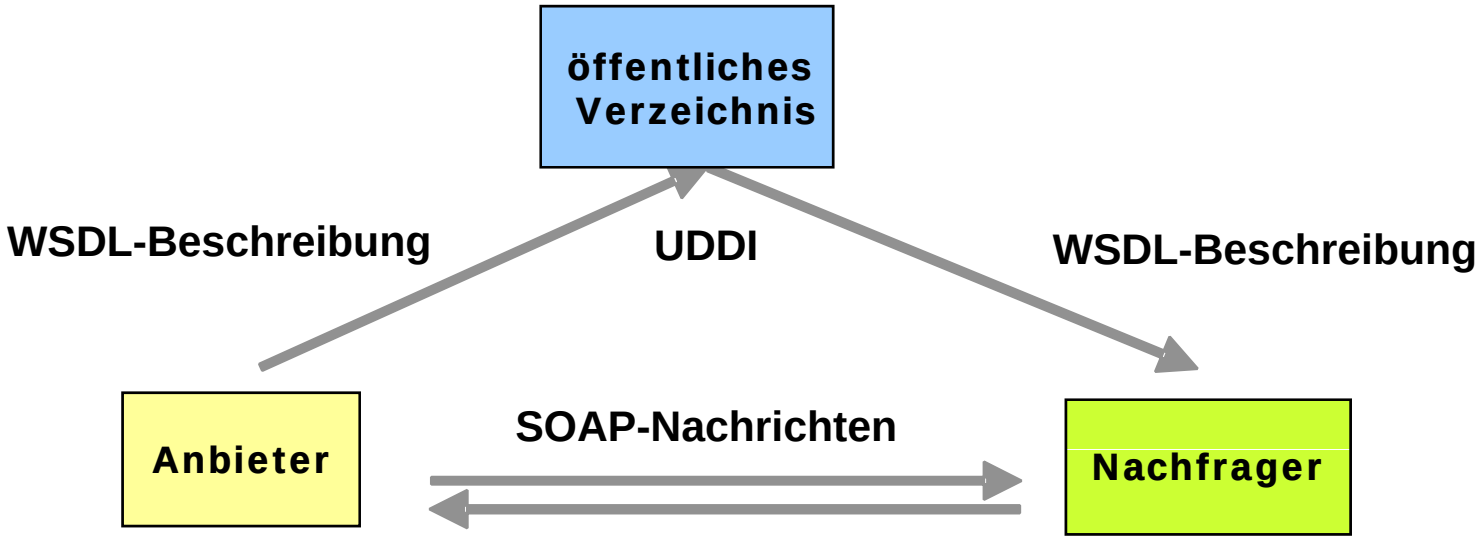
SOAP ist…

Freie Universität Berlin

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

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

Wozu SOAP?



- SOAP: Format zum Austausch von Daten
- Warum spezielles Format und nicht einfach beliebige XML-Syntax zulassen?

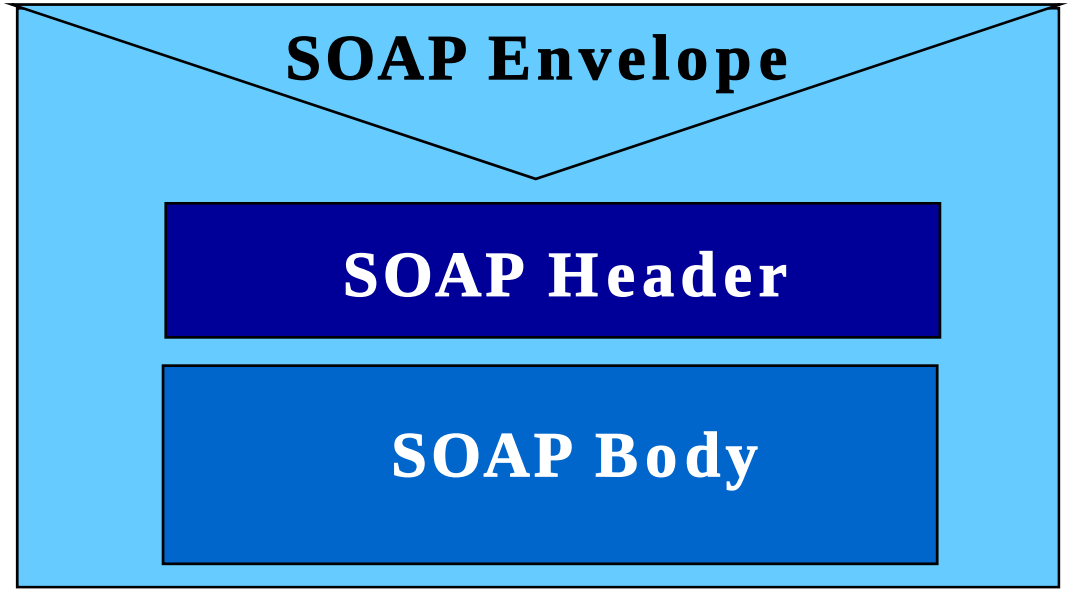
Wozu SOAP?

Freie Universität Berlin

- Es muss auf jeden Fall festgelegt werden wie:
  - Aufruf proc(param-1, …, param-n) kodiert wird
  - Fehlermeldungen kodiert werden
  - Arrays type[] und Matrizen type[][] kodiert werden
- Und genau dies leistet SOAP!
- zusätzlich bietet SOAP noch ein Konzept, um Datenformate einfach zu erweitern

Prinzipieller Aufbau

Aufbau einer SOAP-Nachricht




```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope/">
      <!- SOAP Header -->
      <!- SOAP Body -->
</env:Envelope>
```

SOAP Namensraum

SOAP Block

Freie Universität Berlin

SOAP Nachricht → ein XML Dokument das beinhaltet:

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

SOAP Syntax Regel

Freie Universität Berlin

- SOAP Nachricht **MUSS in XML kodiert** werden
- SOAP Nachricht **MUSS** einen der beiden **SOAP Envelope Namespace** benutzen

- SOAP Nachricht MUSS NICHT Verweis auf DTD beinhalten
- SOAP Nachricht MUSS NICHT XML Processing Anweisungen beinhalten

SOAP Envelope Element

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

- Im Beispiel: W3C-Namensraum → SOAP 1.2

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

Nachrichteninhalt (Body)

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

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

Briefkopf (Header)

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

Header Blöcke

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

Erweiterung von SOAP-Nachrichten

Freie Universität Berlin

- Nachrichtenformat kann durch Header Blocks erweitert werden, ohne ursprüngliches Format (Body) zu modifizieren.
- einzelne Erweiterungen unabhängig voneinander

⇨ mächtiges Konzept für Versionierung

Hypothetisches Beispiel

Freie Universität Berlin

```
<env:Body>
  <DoGoogleSearch>
     …
</DoGoogleSearch>
</env:Body>
```

eigentliche Nachricht (Body) bleibt unverändert

```
<env:Header>
    <Public-Key>
      rg8658hgkkg557j
    </Public-Key>
      …
</env:Header>
```

…

```
<env:Header>
      …
    <NotifyNewPage>
      mymail@inf.fu-berlin.de
    </NotifyNewPage>
</env:Header>
```

unabhängige Erweiterungen

Kodierung von RPCs

Kodierung für RPCs

Freie Universität Berlin

- SOAP auch Nachrichtenformat für entfernte Prozeduraufrufe (RPCs)
- eigentlichen RPCs werden aber von Middleware realisiert
- SOAP selbst unterstützt nur Einweg-Kommunikation
- Anfrage-Antwort-Muster:

1. SOAP mit HTTP übertragen
   - ⇨auf Ebene des Transportprotokolls
2. eindeutige Referenz im SOAP-Briefkopf
   - ⇨auf Ebene von SOAP, dadurch unabhängig vom Transportprotokoll

Mechanismen

Freie Universität Berlin

- Anfrage (Request)
  - → Methodenaufruf
  - → Parameterübergabe
- Antwort (Answer)
  - → fehlerfreie Bearbeitung
  - → Ergebnisübergabe
- Fehlerfall (Fault)
  - → Fehlerübergabe

RPC-Aufruf

Procedure(Parameter-1="val-1",…,Parameter-n ="val-n")

```
<env:Envelope …>
  <env:Body>
    <m:Procedure xmlns:m="URI">
      <m:Parameter-1>val-1</m:Parameter-1>
      …
      <m:Parameter-n>val-n</m:Parameter-n>
    </m:Procedure>
  </env:Body>
</env:Envelope>
```

- Name der Prozedur: Kind-Element von Body
- Eingangsparameter: Kind-Elemente der Prozedur
- Beachte: **Reihenfolge der Parameter relevant**!
- Beachte: grundsätzlich Call-by-Value!

Aufruf-Adresse

Freie Universität Berlin

Wo soll die Prozedur aufgerufen werden (URI)?

- entweder außerhalb von SOAP im Transportprotokoll (z.B. HTTP) spezifiziert
- besser im SOAP-Briefkopf angeben:

```
<env:Header>
  <wsa:EndpointReference
        xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing>
     <wsa:Address>http://api.google.com/search/beta2</wsa:Address>
     <wsa:PortType>ns1:GoogleSearchPort</wsa:PortType>
   </wsa:EndpointReference>
</env:Header>
```

RPC-Ergebnis

Freie Universität Berlin

public Parameter-i Procedure(…,Parameter-j,…)

```
<env:Envelope …>
  <env:Body>
     <m:ProcedureResponse xmlns:m="URI"
           xmlns:rpc="http://www.w3.org/2003/05/soap-rpc">
       <rpc:result>m:Parameter-i</rpc:result>
       <m:Parameter-i>…</m:Parameter-i>
        …
       <m:Parameter-j>…</m:Parameter-j>
     </m:ProcedureResponse>
  </env:Body>
</env:Envelope>
```

- *Wahl von ProcedureResponse* beliebig
- Kind-Elemente von *ProcedureResponse*: Rückgabewerte
- In-Out-Parameter = erscheinen im Aufruf & Antwort

RPC-Ergebnis (2)

public Parameter-i Procedure(…,Parameter-j,…)

```
<env:Envelope …>
  <env:Body>
    <m:ProcedureResponse xmlns:m="URI"
          xmlns:rpc="http://www.w3.org/2003/05/soap-rpc">
      <rpc:result>m:Parameter-i</rpc:result>
      <m:Parameter-i>…</m:Parameter-i>
        …
      <m:Parameter-j>…</m:Parameter-j>
    </m:ProcedureResponse>
  </env:Body>
</env:Envelope>
```

- rpc:result: ausgezeichnetes Ergebnis (optional)
- Namensraum …/soap-rpc Teil der SOAP-Spezifikation

Kodierung von Fehlermeldungen

Im Fall eines Fehlers in der Kommunikation wird ein SOAP Fault Block als einziges Element des SOAP Body übertragen.

```
<env:Envelope …>
  <env:Body>
    <env:Fault>
      <env:Code>
        <env:Value>env:Sender</env:Value>
      </env:Code>
      <env:Reason>
        <env:Text xml:lang="en-US">Processing error</env:Text>
        <env:Text xml:lang="de">Verarbeitungsfehler</env:Text>
      </env:Reason>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

- Code und Reason obligatorisch
- **Code**: für maschinelle Verarbeitung
- **Reason**: zusätzliche Information, nicht für maschinelle Verarbeitung

Beispiel: Timeout-Fehlermeldung

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
              xmlns:m="http://www.example.org/timeouts"
              xmlns:xml="http://www.w3.org/XML/1998/namespace">
  <env:Body>
    <env:Fault>
      <env:Code>
        <env:Value>env:Sender</env:Value>
        <env:Subcode>
          <env:Value>m:MessageTimeout</env:Value>
        </env:Subcode>
      </env:Code>
      <env:Reason>
        <env:Text xml:lang="en">Sender Timeout</env:Text>
      </env:Reason>
      <env:Detail>
        <m:MaxTime>P5M</m:MaxTime>
      </env:Detail>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

Elemente des SOAP Fault Blocks

Freie Universität Berlin

Element-name	**Status**	**Beschreibung**
Code	obligatorisch	von der SOAP Spezifikation festgelegte **Codierung der Fehlerquelle**
Reason	obligatorisch	**textuelle Beschreibung** des aufgetretenen Fehlers
Node	optional	gibt an, **an welcher Stelle** der SOAP Kommunikation der **Fehler** aufgetreten ist
Role	optional	beschreibt die **Rolle des Knotens**, bei dem der Fehler aufgetreten ist
Details	optional	enthält **weitere Infos** zum aufgetretenen Fehler (der Inhalt kann von den Anwendungen frei festgelegt werden)

Datentypen

Kodierung von Arrays

- Beispiel: Als Parameter soll int[3] übergeben werden.
- Wie soll dieses Array dargestellt werden?

so?

```
<array>
  <number
  xsi:type="xsd:int">108</number>
  <number
  xsi:type="xsd:int">99</number>
  <number
  xsi:type="xsd:int">205</
</array>
```

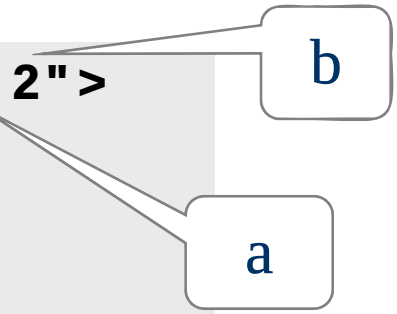
oder so?

```
<array
  elementType="xsd:int">
  108 99 205
</array>
```

Mehrdimensionale SOAP-Arrays

```
<numbers enc:itemType="xsd:int" enc:arraySize="3 2">
  <number>1</number>
  <number>2</number>
  <number>3</number>
  <number>4</number>
  <number>5</number>
  <number>6</number>
</numbers>
```

➔ a1 b1
➔ a2 b1
➔ a3 b1
➔ a1 b2
➔ a2 b2
➔ a3 b2



- 3x2-Matrix mit Elementen vom Typ xsd:int.
- enc:arraySize="* 2": n x 2-Matrix
- Beachte: * nur an erster Stelle erlaubt

⇨ eindeutig auflösbar

SOAP-Arrays

```
<numbers xmlns:enc="http://www.w3.org/2003/05/soap-encoding"
         enc:itemType="xsd:int" enc:arraySize="3">
  <number>1</number>
  <number>2</number>
  <number>2</number>
</numbers>
```

- entspricht int[3] bei SOAP 1.1
- Element-Namen (hier numbers und number) beliebig, entscheidend sind Attribute enc:itemType und enc:arraySize
- Namensraum **.../soap-encoding** Teil der SOAP-Spezifikation
- enc:arraySize="*": entspricht int[]

Beispiel enc:arraySize="* 2"

```
<numbers enc:itemType="xsd:int" enc:arraySize="* 2">
  <number>1</number>
  <number>2</number>
  <number>3</number>
  <number>4</number>
  <number>5</number>
  <number>6</number>
</numbers>
```

b

a

➔ a1 b1
➔ a2 b1
➔ a3 b1
➔ a1 b2
➔ a2 b2
➔ a3 b2

- #Elemente = 6 = n x 2

⇨ eindeutige Lösung: n = 3

- für #Elemente = 6 = n x m gäbe es keine eindeutige Lösung

⇨ enc:arraySize="* *" nicht erlaubt

env:encodingStyle

Freie Universität Berlin

- die vorgestellte Kodierung für RPCs und Arrays muss nicht verwendet werden
- wird sie verwendet, dann in SOAP-Nachricht folg. Kodierungsschema angeben:

```
env:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
```

- Beachte: für SOAP 1.1 hier andere URL!

- Kodierungsschema auch anwendungsspezifisch:

```
env:encodingStyle="http://www.ibm.com/soap-encoding"  (fiktiv)
```

⇨ Empfänger muss entspr. Kodierungsschema kennen

Freie Universität Berlin

Verarbeitung von SOAP-Nachrichten

Freie Universität Berlin

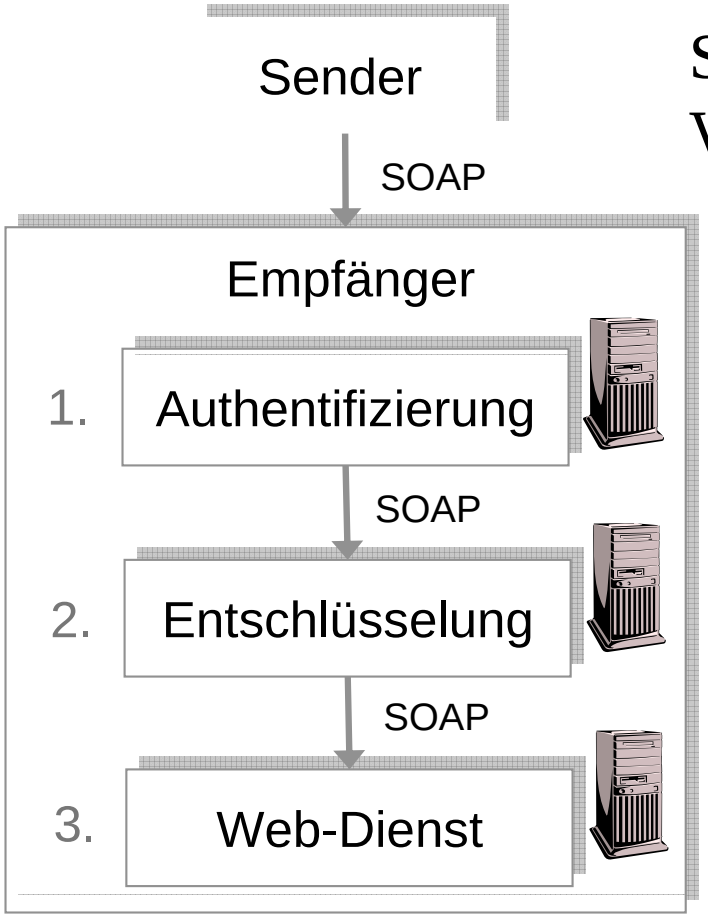
Empfänger **muss verarbeiten**:

- Body
- Header Blocks mit mustUnderstand="true"

Empfänger **darf ignorieren**:

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

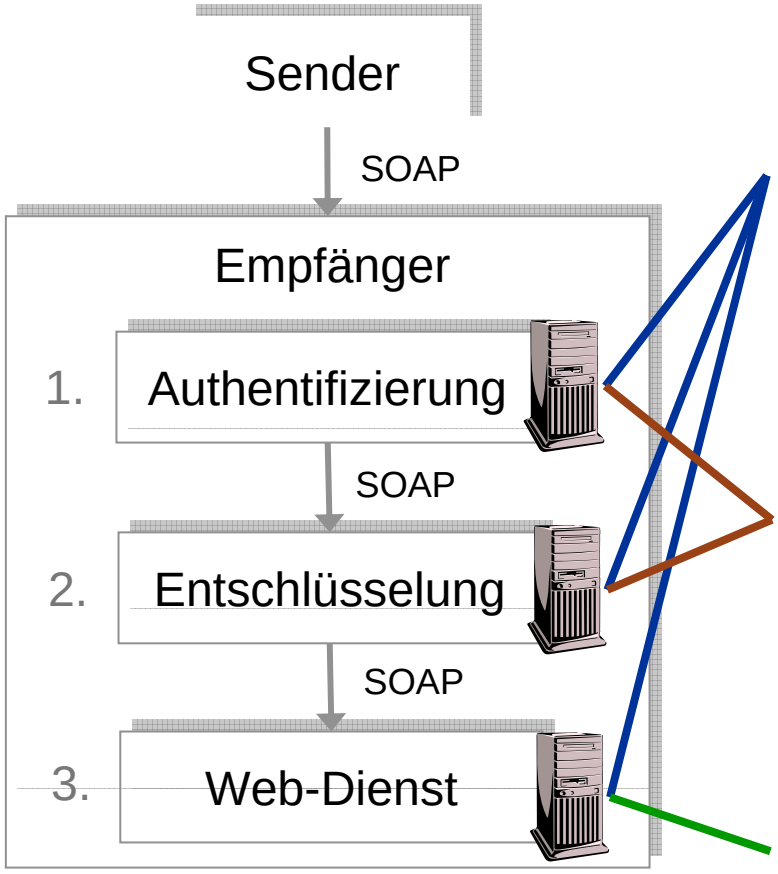
Schrittweise Verarbeitung



SOAP unterstützt schrittweise Verarbeitung von Nachrichten, z.B.:

1. Authentifizierung: Verifizierung einer digitalen Signatur in einem Header Block
2. Entschlüsselung des Body
3. Aufruf des eigentlichen Web Services

SOAP-Knoten



SOAP-Knoten: Rechner, die Teil einer SOAP-Nachricht verarbeiten

Zwischenknoten (intermediary)

Endknoten (ultimate receiver)

Typen von SOAP Nodes

Freie Universität Berlin

- Initial SOAP sender
  - erzeugt die SOAP Nachricht
  - ist der Startpunkt des sog. SOAP Nachrichtenpfades (Message Path)
- SOAP Intermediary
  - empfängt eine SOAP Nachricht, verarbeitet Teile der Nachricht, modifiziert evtl. Teile der Nachricht und sendet die Nachricht weiter
- Ultimate SOAP Receiver
  - endgültiger Bestimmungsort der SOAP Nachricht
  - Endpunkt des SOAP Nachrichtenpfades
  - Nachricht muß diesen nicht in jedem Fall erreichen

Identifizierung von SOAP-Knoten

Freie Universität Berlin

- SOAP-Knoten werden mit URIs identifiziert

1. anwendungspezifische URI
   - z.B. www.example.org/Log
   - muss vom Empfänger interpretiert werden können

2. standardisierte URI
   - http://www.w3.org/2003/05/envelope/role/next

   = aktueller Empfänger (Zwischen- oder Endknoten)

   - http://www.w3.org/2003/05/envelope/role/ultimateReceiver

   = Endknoten

Festlegung der Zuständigkeiten

Freie Universität Berlin

```
<env:Header>
   <FirstBlock env:role="www.example.org/Log">
      ...
   </FirstBlock>
   <SecondBlock
    env:role="http://www.w3.org/2003/05/envelope/role/next">
      ...
   </SecondBlock>
   <ThirdBlock>
         ...
   </ThirdBlock>
</env:Header>
```

➔ "Log"-Knoten zuständig

➔ aktueller Empfänger zuständig

➔ Endknoten zuständig

- role: zuständiger SOAP-Knoten (URI)
- Beachte: fehlt role-Attribut, dann ist Endknoten zuständig

role Attribut

Freie Universität Berlin

- Spezifiziert den Empfänger oder Zwischenstation, die die das Header verarbeiten darf

http://www.w3.org/2003/05/soap-envelope/role/next

→ dieser Teil des Headers ist für die nächste Anwendung bestimmt, die die Nachricht verarbeiten wird.

http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver

→ dieser Teil des Headers ist nur für den letzten „Stop" bestimmt

http://www.w3.org/2003/05/soap-envelope/role/none

→ schaltet den Header-Teil aus

Aufgabe eines Zwischenknotens

Freie Universität Berlin

- verarbeitet Header Blocks mit
  - role="http://www.w3.org/2003/05/envelope/role/**next**"
  - role="*URI*", wobei "*URI*" den betreffenden Zwischenknoten bezeichnet

- Alle anderen Header Blocks und Body werden nicht verarbeitet.
- löscht alle verarbeiteten Header Blocks !
- fügt evtl. neue Header Blocks hinzu
- entscheidet, welcher SOAP-Knoten nächster Knoten in Verarbeitungsskette sein soll
- leitet modifizierte SOAP-Nachricht an diesen SOAP-Knoten weiter

Tiefe der Verarbeitung

Freie Universität Berlin

- Tiefe der Verarbeitung durch mustUnderstand-Attribut bestimmt:

mustUnderstand="true"

- Empfänger muss Header Block verstehen, ansonsten Fehlermeldung
- Löschen von unbekannten Header Blocks nicht erlaubt

mustUnderstand="false"

- Empfänger kann Header Block ignorieren
- Löschen von unbekannten Header Blocks erlaubt

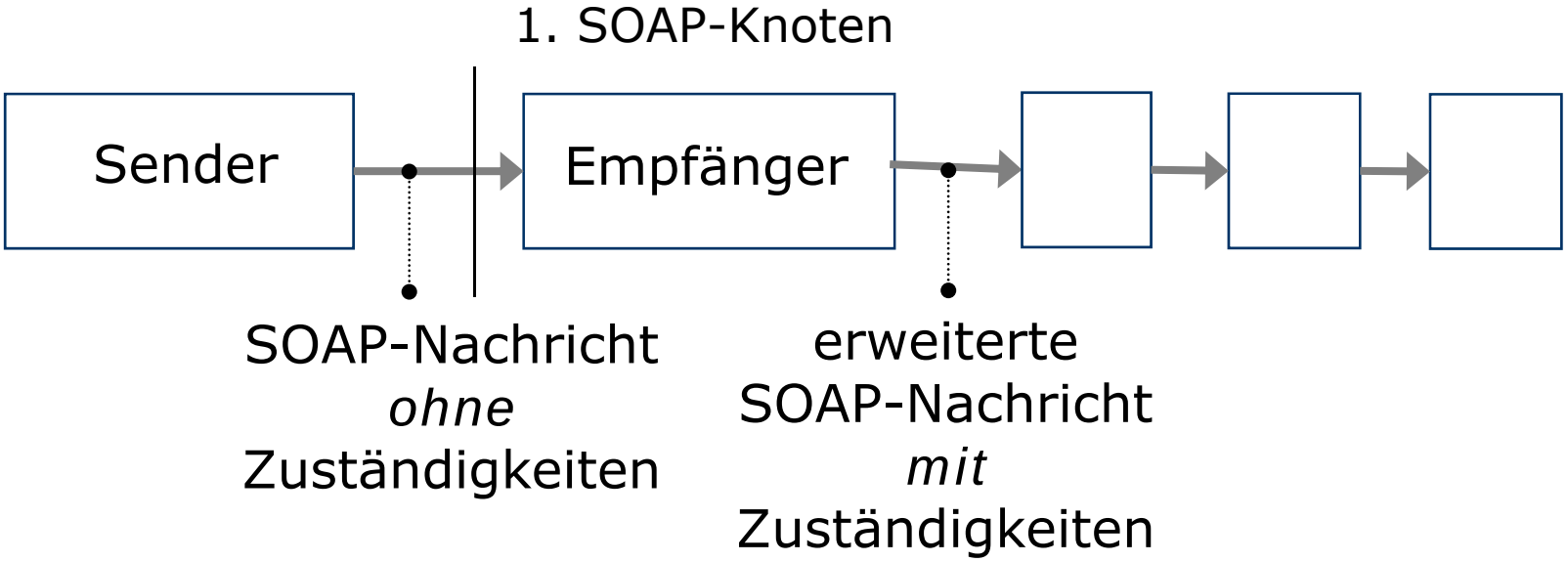
Aufgabe des Endknoten

Freie Universität Berlin

1. verarbeitet Header Blocks
   - mit role="http://www.w3.org/2003/05/envelope/role/**ultimateReceiver"**
   - mit role="http://www.w3.org/2003/05/envelope/role/**next**"
   - ohne role-Attribut
2. versteht und verarbeitet Body

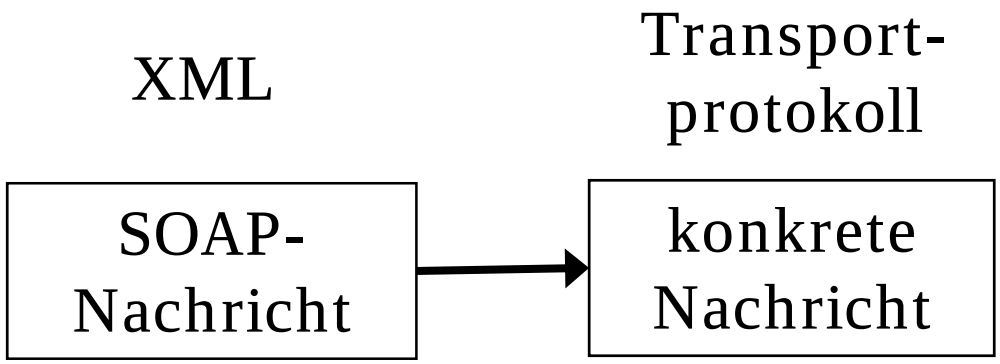
Festlegung der Zuständigkeiten?

- Vorteile der schrittweisen Verarbeitung → Aufgabenverteilung auf spezialisierte Server



Übertragung von SOAP-Nachrichten

Protokoll-Bindung (Binding)



- Wie werden SOAP-Nachrichten mit bestimmten Transportprotokoll übertragen?
- Wie SOAP-Nachrichten serialisieren?
- SOAP-Spezifikation schreibt nicht vor, **wie** Protokoll-Bindung spezifiziert wird

Anforderungen an Protokoll-Bindung

Freie Universität Berlin

- konkrete Nachricht meist XML, kann aber auch beliebig anderes Format sein:

  z.B. komprimiertes Binärformat
- einzige Bedingung:

  Serialisierung ohne Informationsverlust

- Serialisierung s muss also symmetrisch sein:

  $s^{-1}(s(N)) = N$, für alle SOAP-Nachrichten N

Standardisierte Protokoll-Bindungen

Freie Universität Berlin

- HTTP-Binding bisher als einzige Protokoll-Bindung für SOAP standardisiert

- zwei unterschiedliche HTTP-Bindungen:
  - HTTP-POST
  - HTTP-GET

HTTP-GET vs. HTTP-POST

Freie Universität Berlin

HTTP GET

- URL ➔ Antwort
- Parameter können in URL kodiert werden, z.B.:

http://google.com/doGoogleSearch?q=Beginning+XML

= Aufruf doGoogleSearch(q="Beginning XML")

HTTP POST

- URL + Datenanhang ➔ Antwort

SOAP 1.1 über HTTP POST: Anfrage

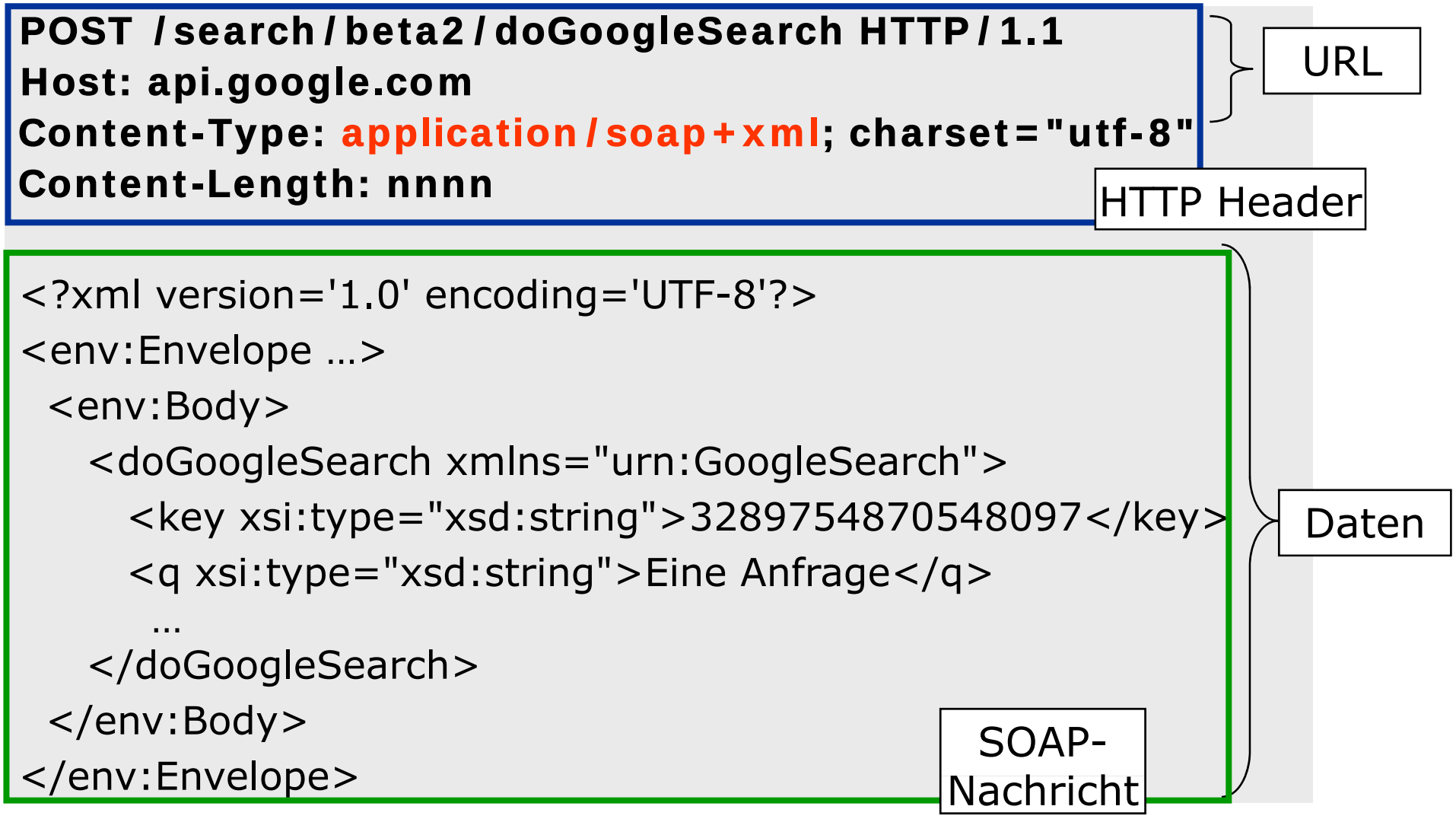
Freie Universität Berlin

```
POST /search/beta2/doGoogleSearch HTTP/1.1
Host: api.google.com
Content-Type: text/xml; charset="utf-8"
SOAPAction: ""
Content-Length: nnnn
```

URL

HTTP Header

SOAP 1.2 über HTTP POST: Anfrage



SOAP über HTTP POST: Antwort

Freie Universität Berlin

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

HTTP Header

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

SOAP-Response

SOAP über HTTP GET

```
GET /search/beta2/doGoogleSearch?q=Beginning+XML
    HTTP/1.1
Host: api.google.com
Accept: application/soap+xml
```

- ruft doGoogleSearch(q="Beginning XML") auf
- gesamte SOAP-Nachricht als URL kodiert
- Antwort wie bei HTTP POST
- Amazon bietet HTTP-GET-Schnittstelle an, Google jedoch nicht
- sehr beliebt weil leichtgewichtig

REST oder nicht?

Freie Universität Berlin

HTTP GET

⇨ entspricht REST-Grundsatz

- würde z.B.
  travel.com/Reservations/itinerary?reservationCode=FT35ZBQ
  anfragen

⇨ URL identifiziert eindeutig gebuchte Reise

HTTP POST

⇨ widerspricht REST-Grundsatz

- würde z.B.
  travel.com/Reservations/
  anfragen mit SOAP-RPC als Datenhang:
  itinerary(reservationCode="FT35ZBQ")

⇨ URL identifiziert nicht gebuchte Reise

HTTP POST vs. HTTP GET

Freie Universität Berlin

- SOAP-Spezifikation empfiehlt HTTP GET, wenn Parameter Web-Ressourcen identifizieren
  - ⇨ Übereinstimmung mit REST-Grundsatz
- Problem:
  - Wie komplexe Parameter als URI kodieren?
- Beispiel:
  - reservationCode könnte aus Bezeichner + Datum bestehen
  - so kodieren?

```
travel.com/Reservations/itinerary?reservationCode=FT35ZBQ+22/6/2005
```

  - oder so?

```
travel.com/Reservations/itinerary?reservationId=FT35ZBQ&reservationDate=22/6/2005
```

Vor- und Nachteile von SOAP

SOAP - Vorteile

Freie Universität Berlin

+ etablierter Standard (u.a. in .Net verwendet)
+ unabhängig von Übertragungsprotokollen
+ sowohl für RPCs als auch für Messaging geeignet
+ einfach erweiterbar
+ Erweiterungen unabhängig voneinander
+ Plattformunabhängig
+ Programmiersprachenunabhängig

SOAP - Nachteile

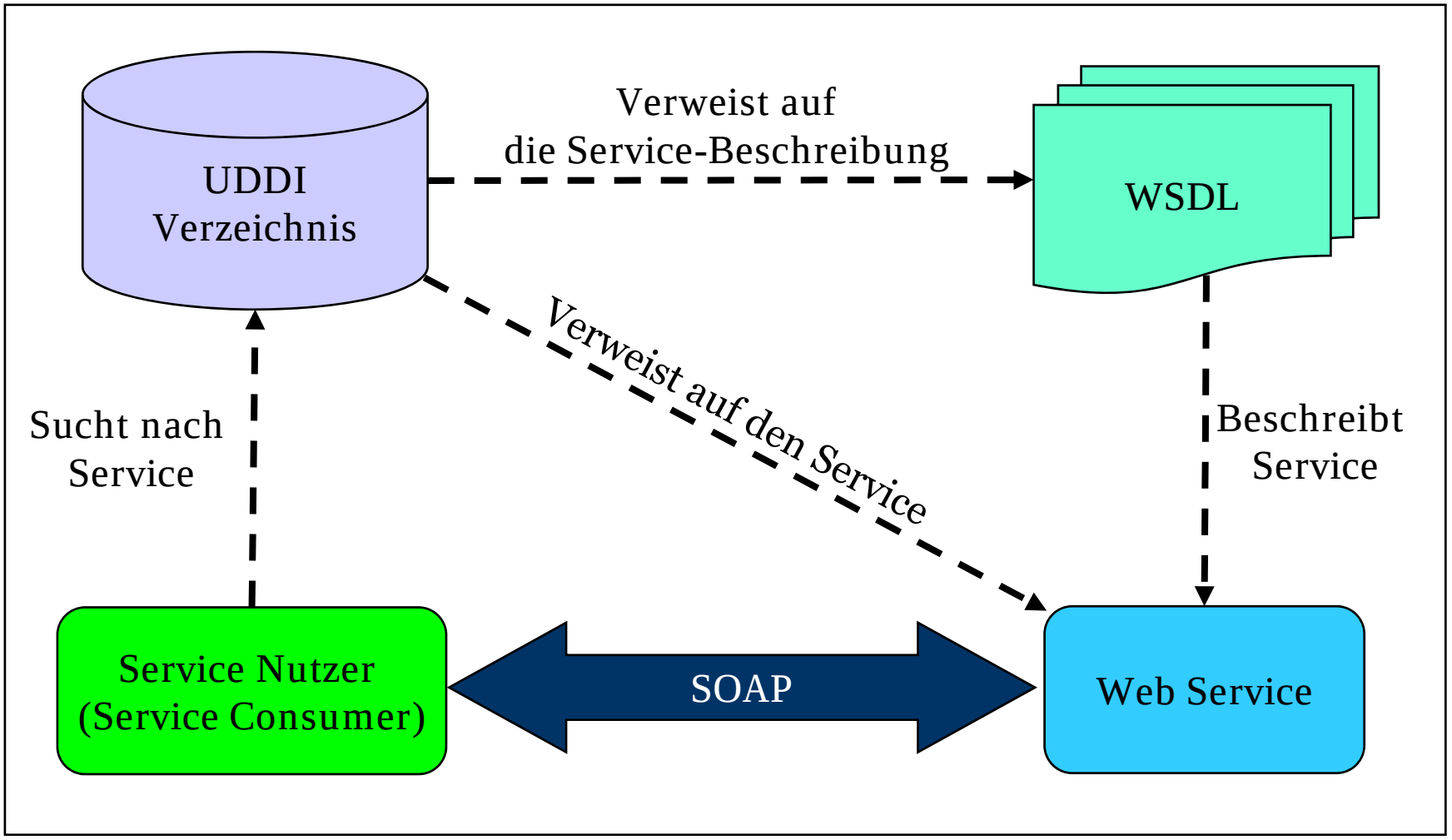
- zusätzlicher Verarbeitungsaufwand
- nicht so einfach zu erlernen
- für viele notwendige Erweiterungen noch kein etablierter Standard

  Beispiel: wsu:identifier vs. wsa:MessageID
- heutzutage noch nicht vollständig interoperabel

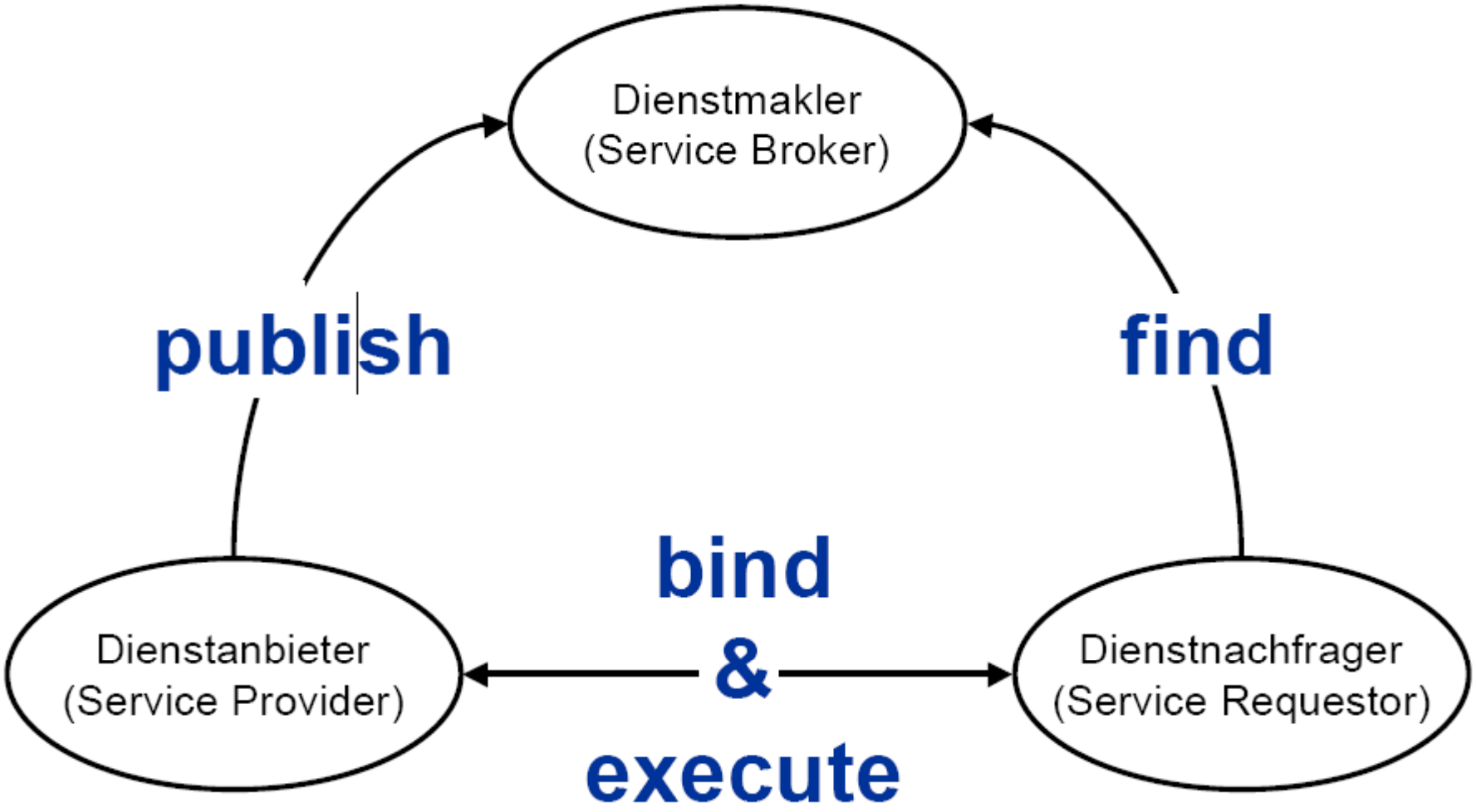
Freie Universität Berlin

WSDL

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

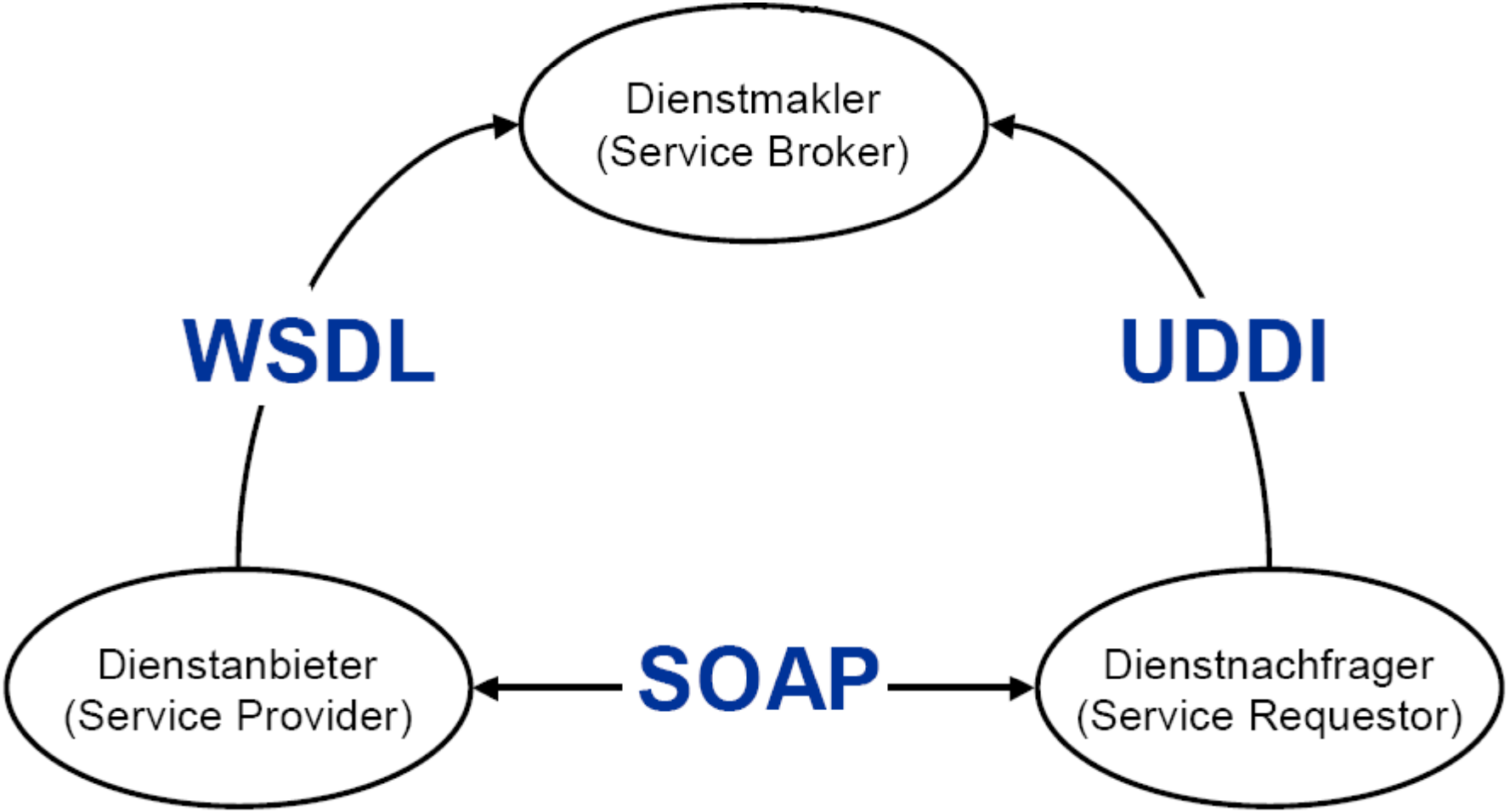

UDDI
Verzeichnis
Verweist auf
die Service-Beschreibung
WSDL
Verweist auf den Service
Sucht nach
Service
Beschreibt
Service
Service Nutzer
(Service Consumer)
SOAP
Web Service

Web Services (2)



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

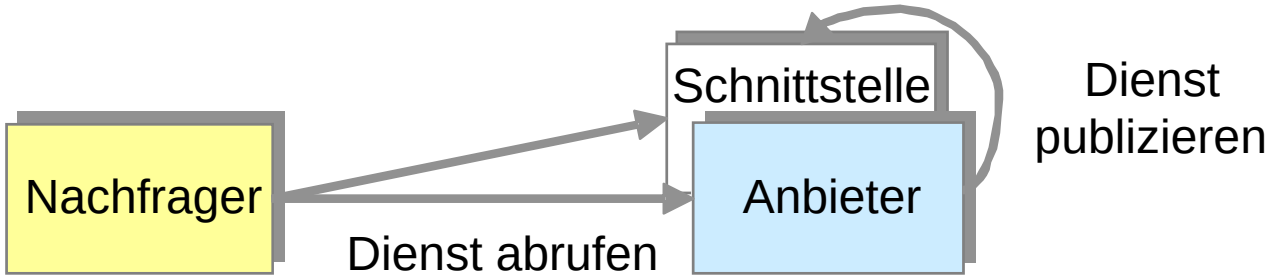
Web Services (3)



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

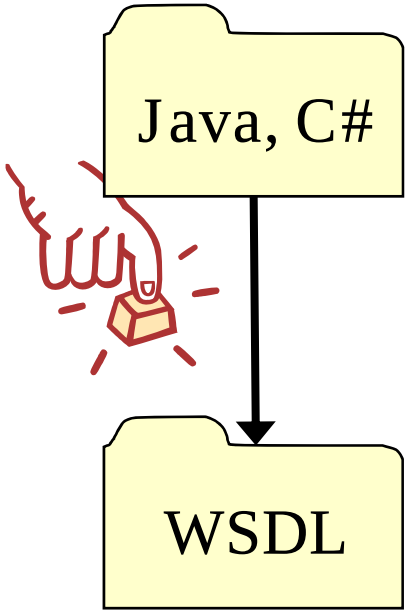
Wozu dient WSDL?

Formale Beschreibung der Schnittstelle von Services



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

Wozu WSDL-Syntax verstehen?



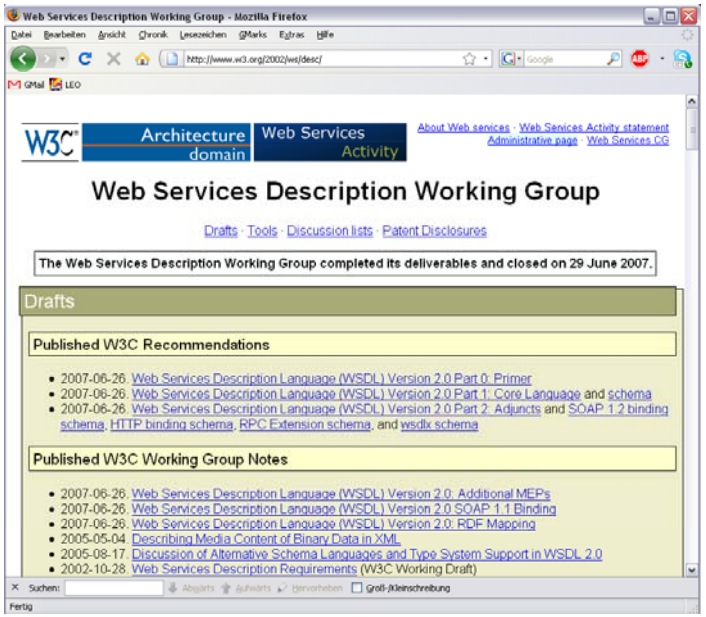
- WSDL = zu veröffentlichende Schnittstellenbeschreibung (Vertrag)
- Nutzer des Web Service kennt nur WSDL, nicht Programm-Code

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

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

WSDL 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

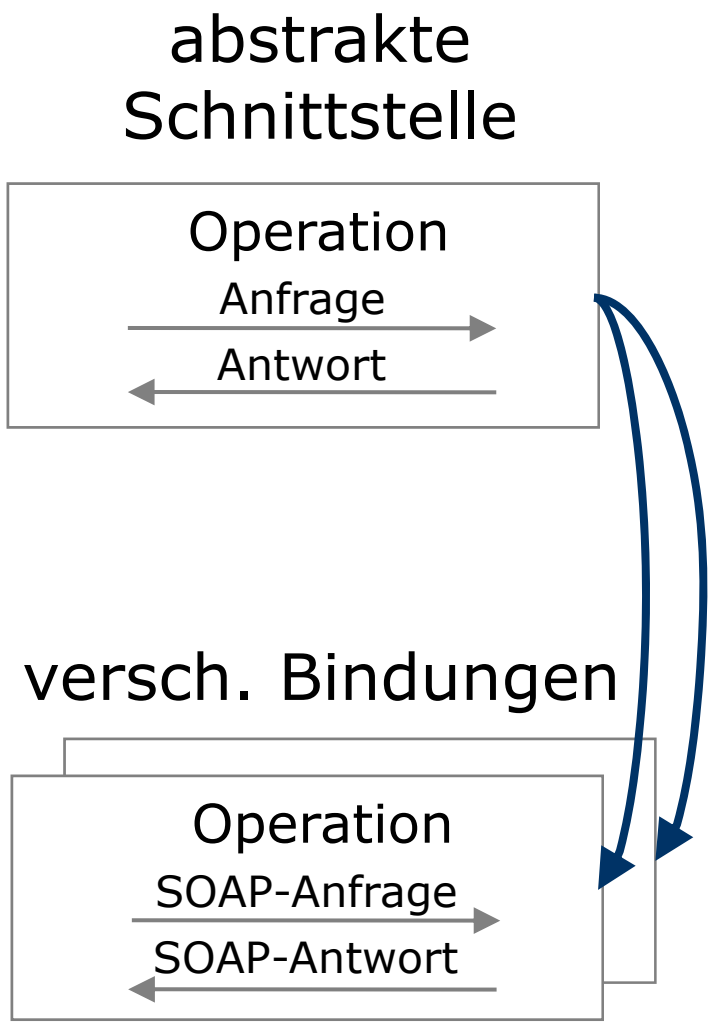
A. Dhesiaseelan „What's New in WSDL 2.0", 2004, http://www.hotcoding.com/xmls/webservice/33297.html

Prinzipieller Aufbau – allgemein

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

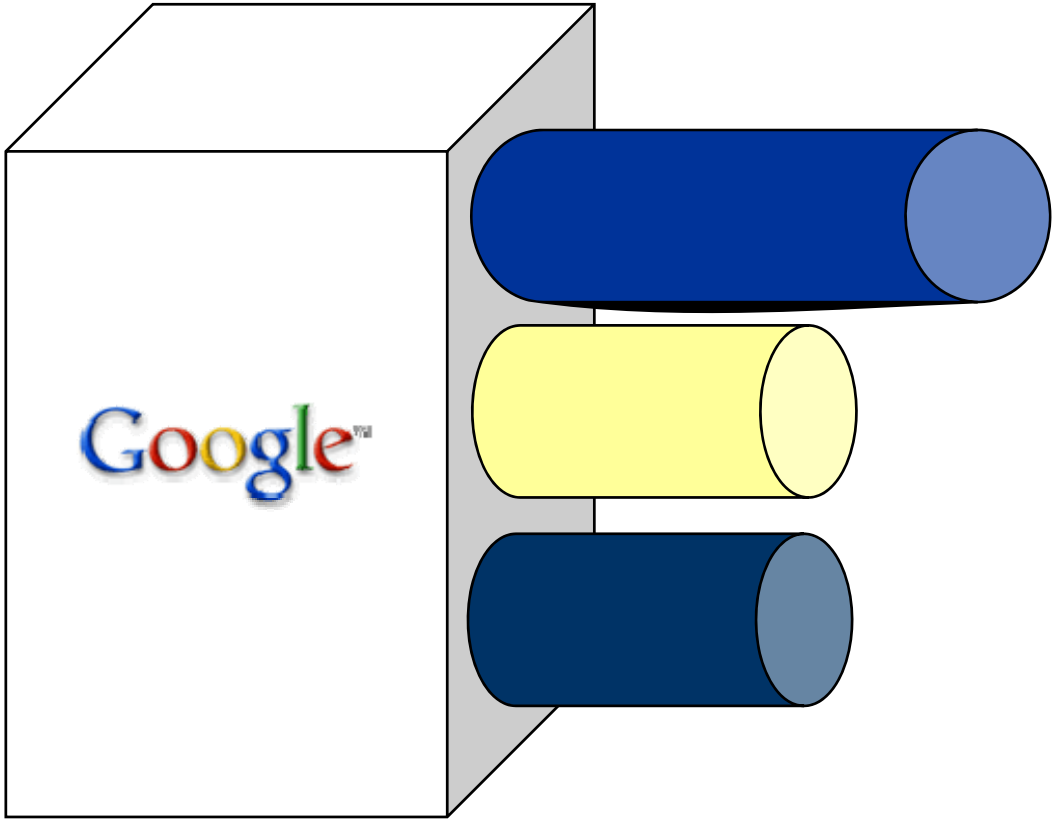
- Realisierung einer abstrakten Schnittstelle mit bestimmtem Nachrichtenformat und Übertragungsprotokoll

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)

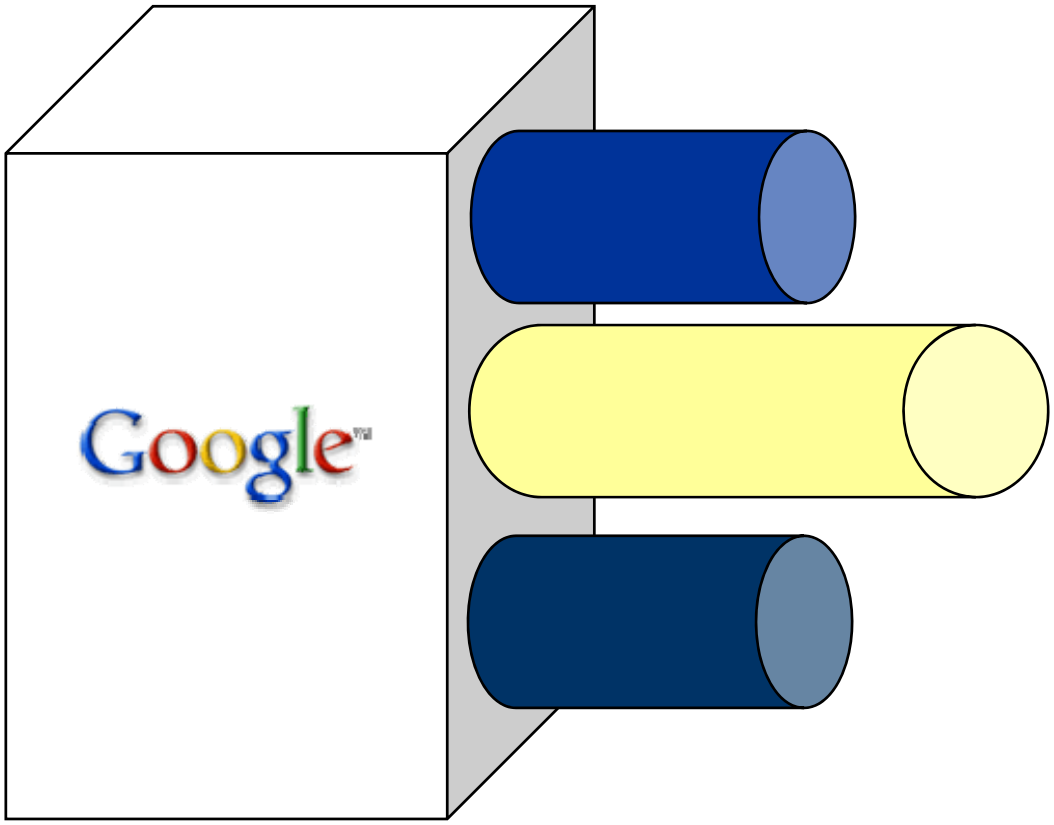
- eine Beschreibung (**WSDL**), aber 4 Zugriffs-möglichkeiten (**Bindungen**):
  1. SOAP/HTTP-POST
  2. SOAP/HTTP-GET (Rest)
  3. SOAP/SMTP (asynchron)
  4. HTML/HTTP-GET (Browser)



Dienst: Suche

Name der Operation:
doGoogleSearch

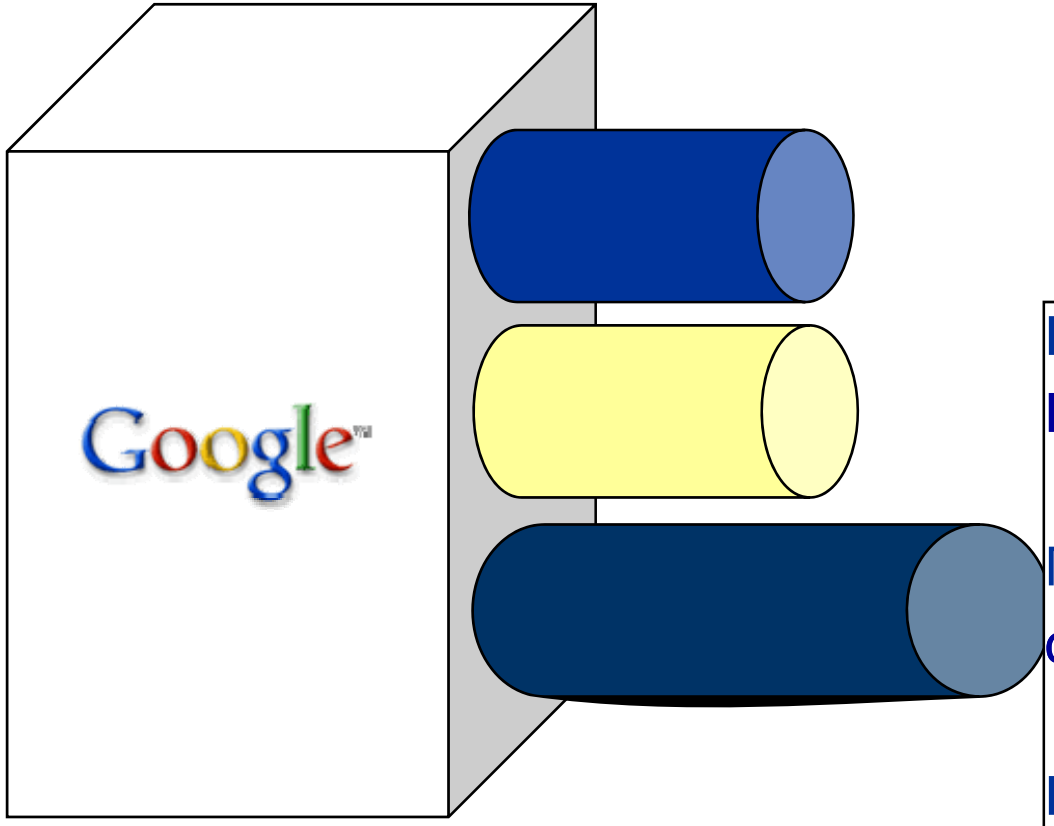
Rückgabe:
doGoogleSearchResponse



Dienst:
Zugriff auf Web-Cache

Name der Operation:
doGetCachedPage

Rückgabe:
doGetCachedPageResponse



Dienst:
Rechtschreibkorrektur

Name der Operation:
doSpellingSuggestion

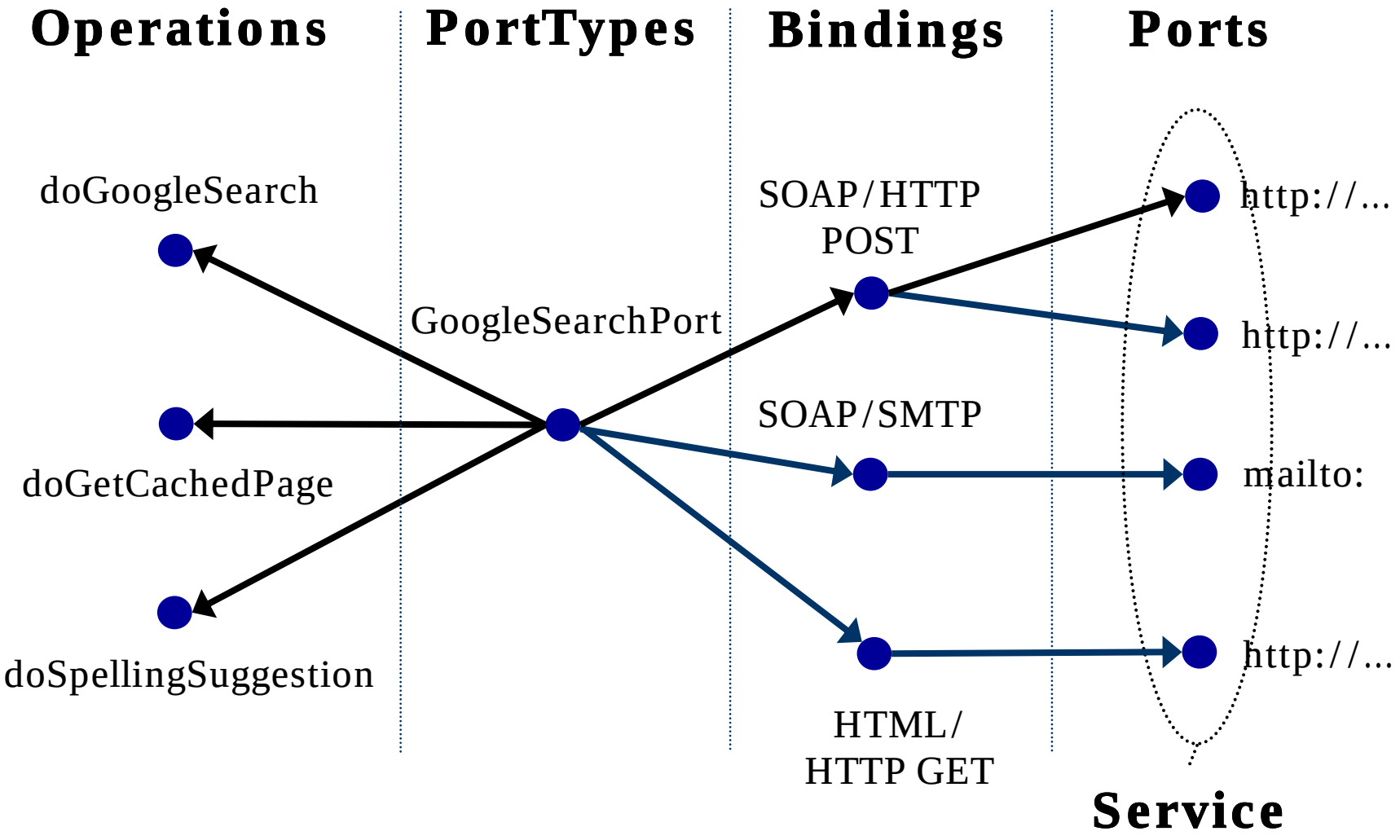
Rückgabe:
doSpellingSuggestionResponse

WSDL-Inhalt

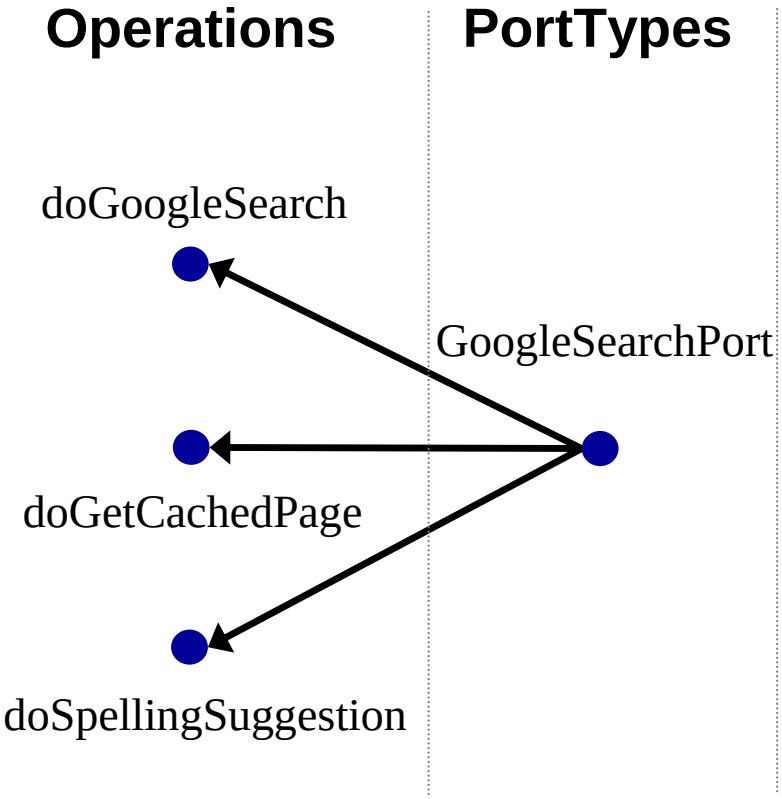
Freie Universität Berlin

- 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

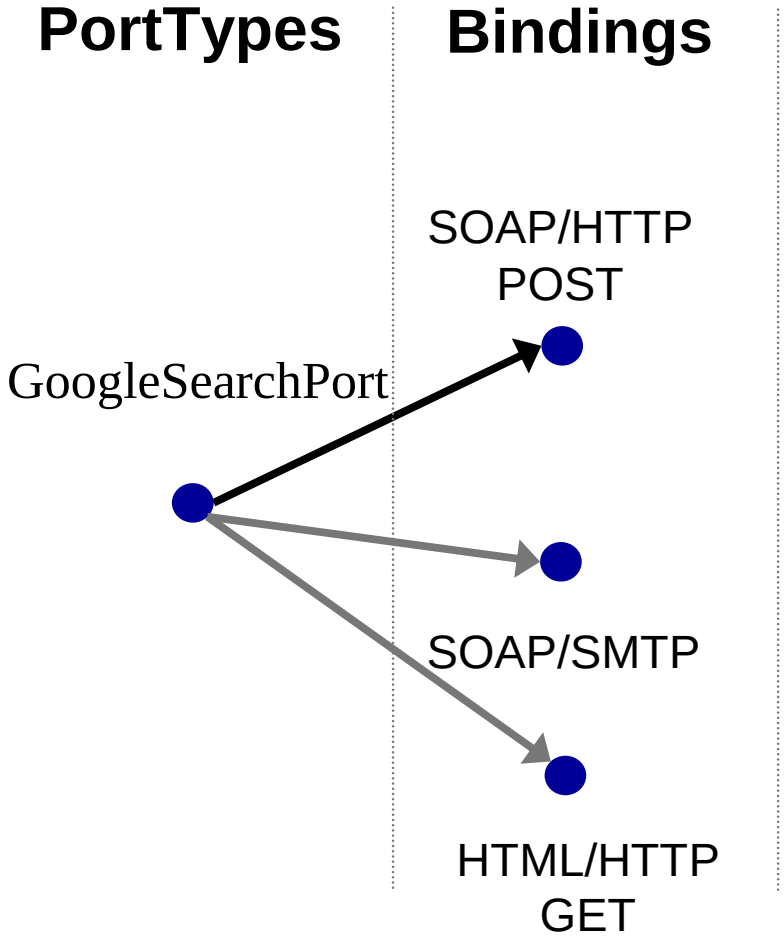


Abstrakte Schnittstellen



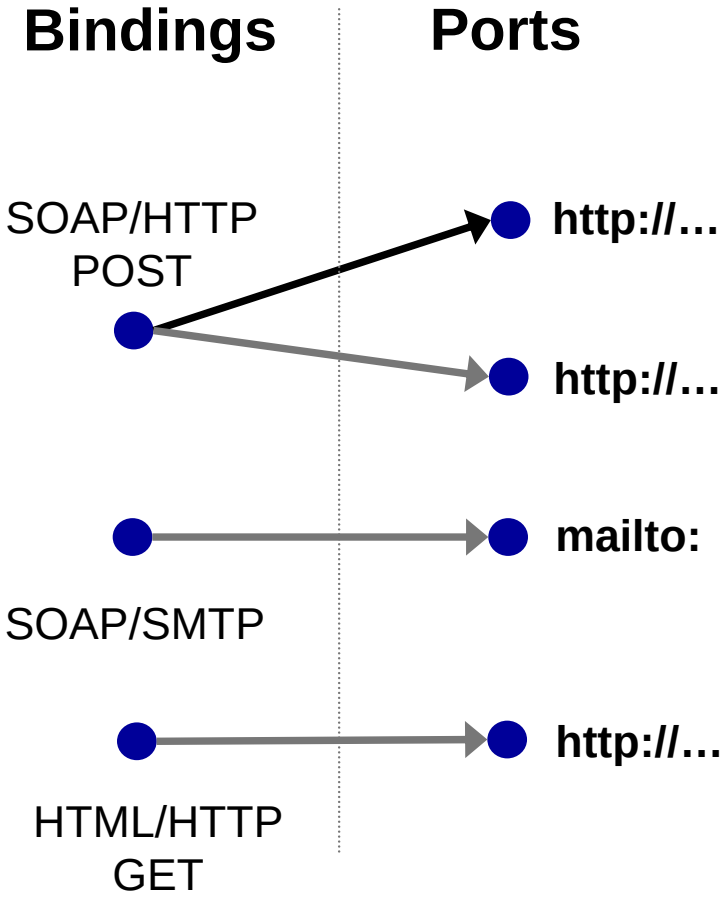
- **portType** (WSDL 1.1) **= interface** (WSDL 2.0)
- portType = Menge von abstrakten Operationen
- jede abstrakte Operation beschreibt Eingangs- und Ausgangsnachricht
- meist nur ein portType, aber in WSDL 1.1 auch mehrere möglich

Bindungen



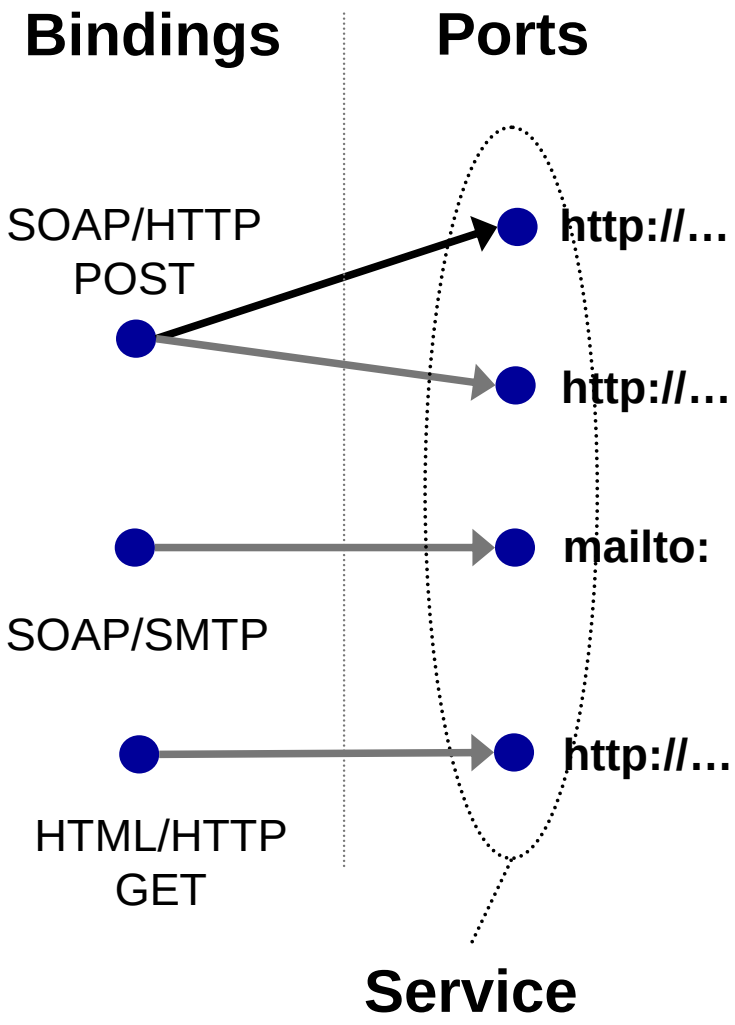
- in WSDL **binding** genannt
- für jede abstrakte Schnittstelle (**portType**) mindestens eine Bindung
- ein **portType** kann also mit **unterschiedlichen Bindungen** realisiert sein

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

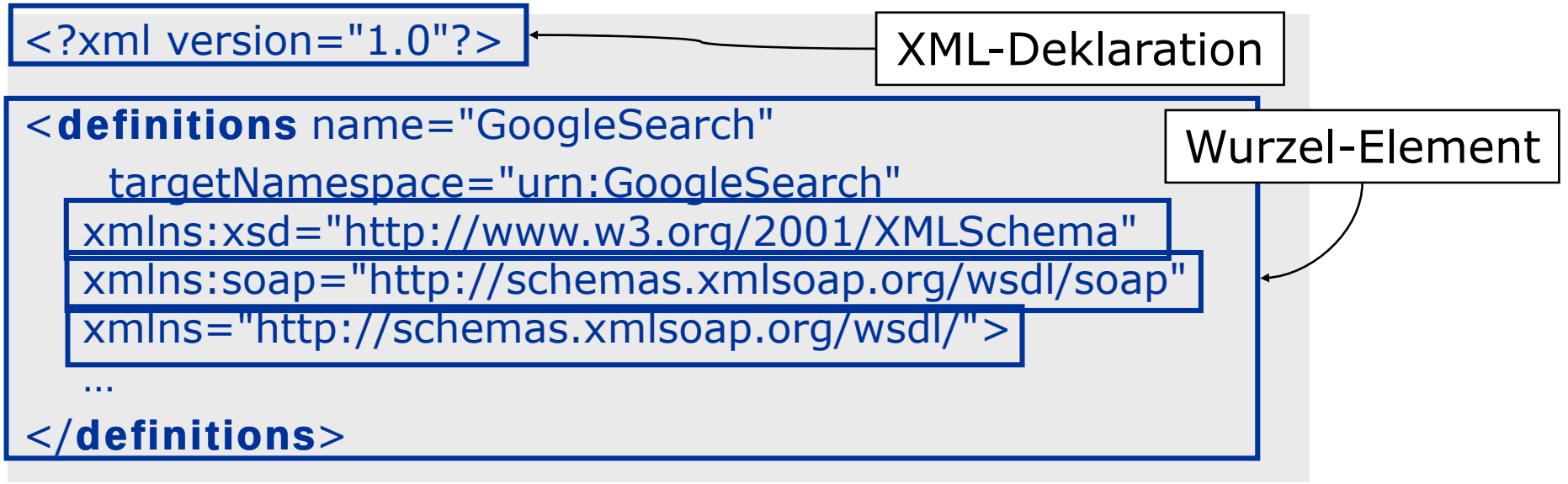
WSDL 1.1. – Elemente (1)

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

Freie Universität Berlin

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

XML-Syntax von WSDL



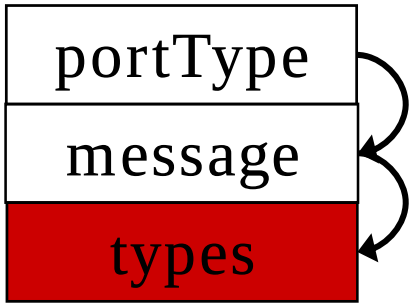
- **Wurzel-Element definitions** aus entsprechendem Namensraum
- **Namensraum** von **definitions** = Version
- WSDL-Beschreibung kann Ziel-Namensraum definieren

⇨ 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>
    ...
</definitions>
```

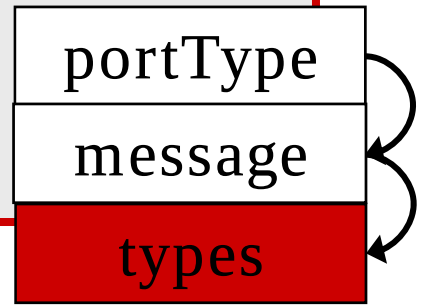


types

- Definition von Datentypen
- werden für Spezifikation von abstrakten Nachrichten 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

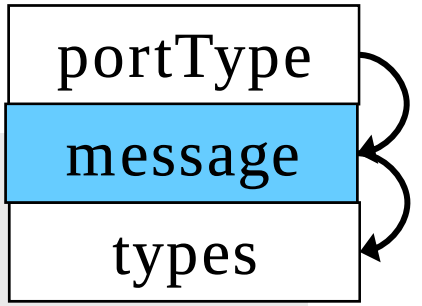
Freie Universität Berlin

```
<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>
```

message

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

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

Freie Universität Berlin

zwei unterschiedliche Modellierungen

1. mehrere **part**s:

```
<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 **part**s:

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

2. ein **part** mit komplexen Dat

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

tns:complexType könnte z.B

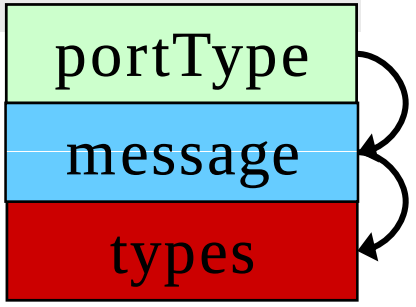
Unterschiede

- **part**s 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>

Freie Universität Berlin

```
<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

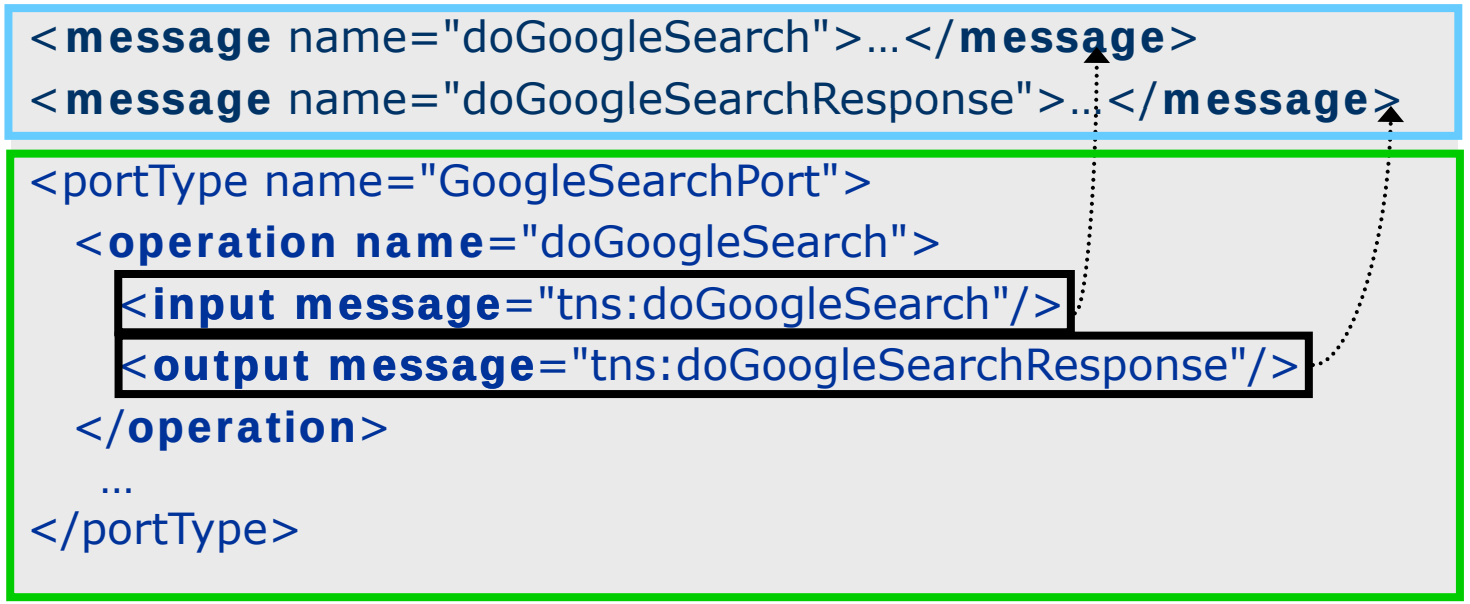
Freie Universität Berlin

```
<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 (**operations**)

Abstrakte Schnittstelle: operation



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

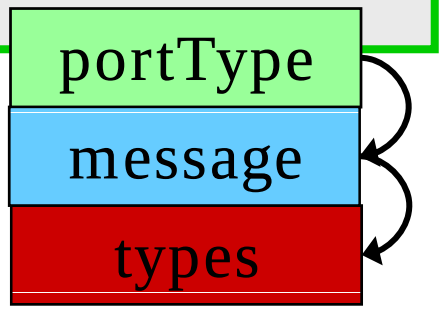
Abstrakte Nachricht vs. Datentyp

```
<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

Freie Universität Berlin

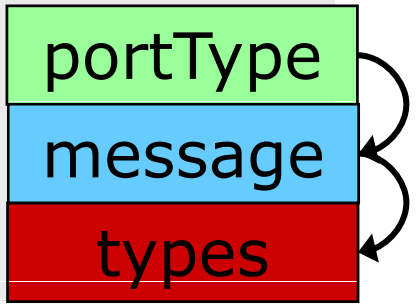
```
<types>
   <xsd:schema  xmlns:xsd="…"  xmlns:tns="…" targetNamespace="…">
     <xsd:complexType name="GoogleSearchResult">
        …
     </xsd:complexType>
   </schema>
 </types>
```

Definition des Datentyps

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

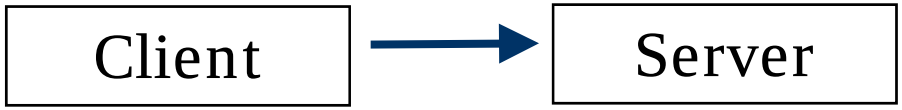
Definition einer abstrakten Nachricht

Definition einer abstrakten Schnittstelle



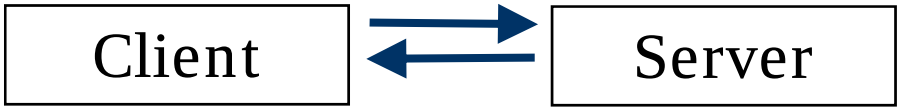
Freie Universität Berlin

Einweg (oneway)




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

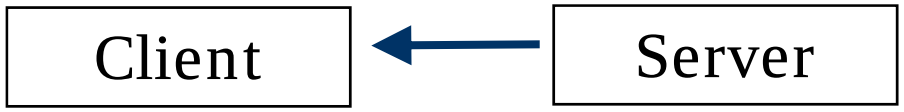
Anfrage-Antwort (request-response)




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

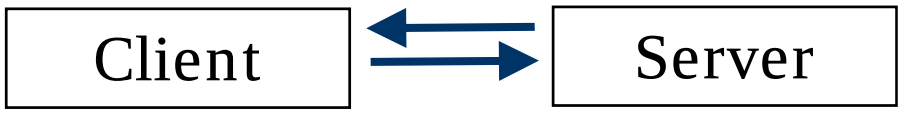
Freie Universität Berlin

Benachrichtigung (notification)




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

Benachrichtigung-Antwort (solicit-response)



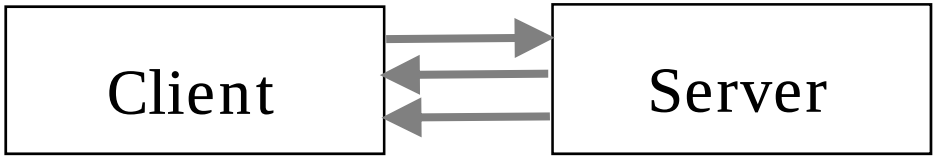

```
<operation name="...">
    <output message="..."/>
    <input message="..."/>
</operation>
```

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

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




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

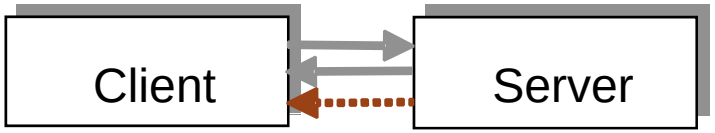
In WSDL 1.1
nicht erlaubt!

Fehlermeldungen

Freie Universität Berlin

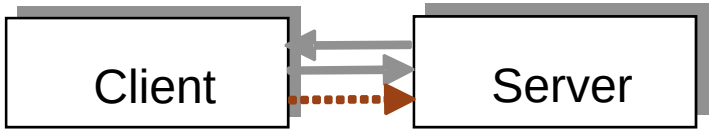
Anfrage-Antwort

```
<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>

Freie Universität Berlin

```
<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

Bindungen

Bindungen

1. SOAP

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

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

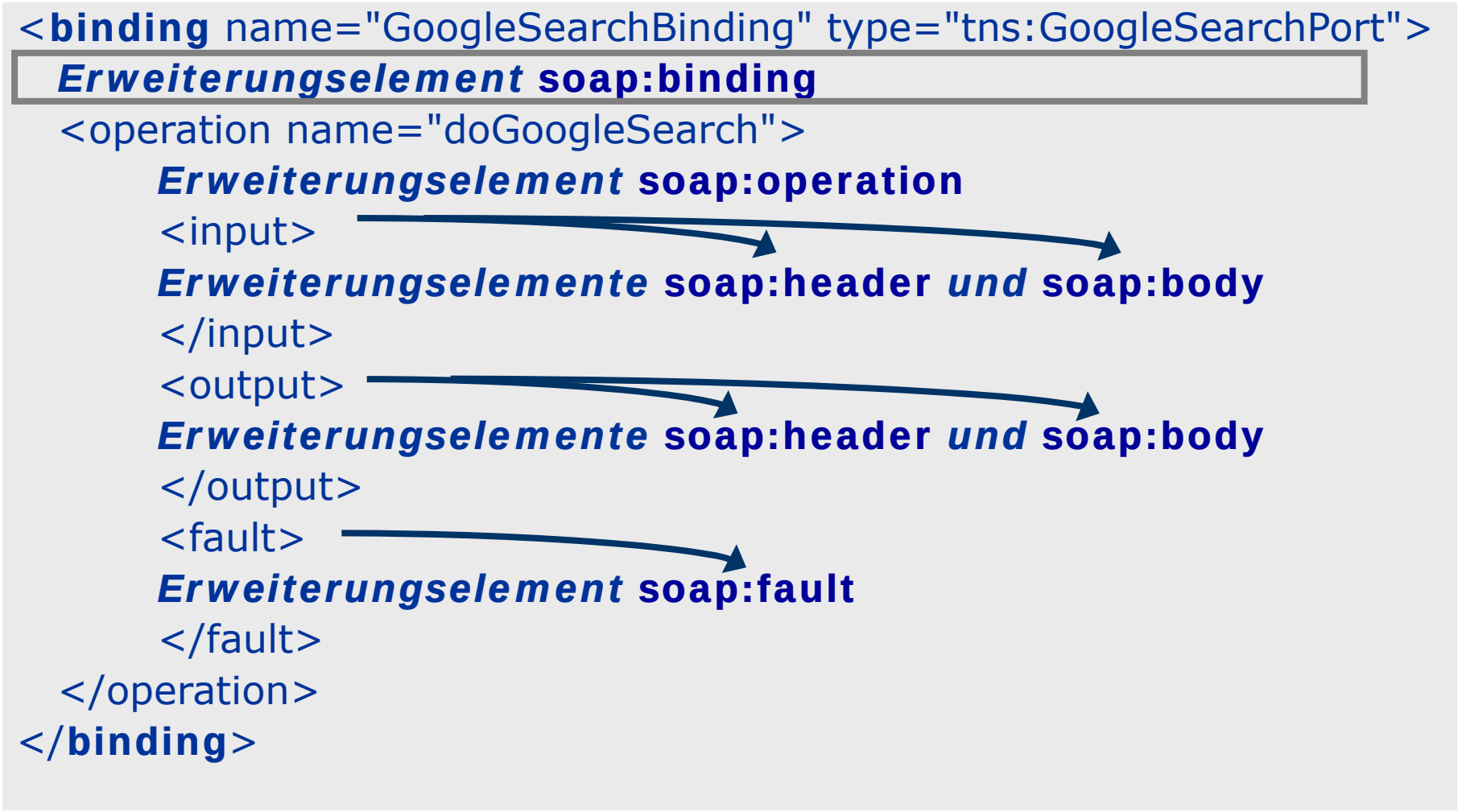
Bindung: SOAP-Bindung

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



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-Style: Etwas irreführend!

Freie Universität Berlin

style="rpc"

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

style="document"

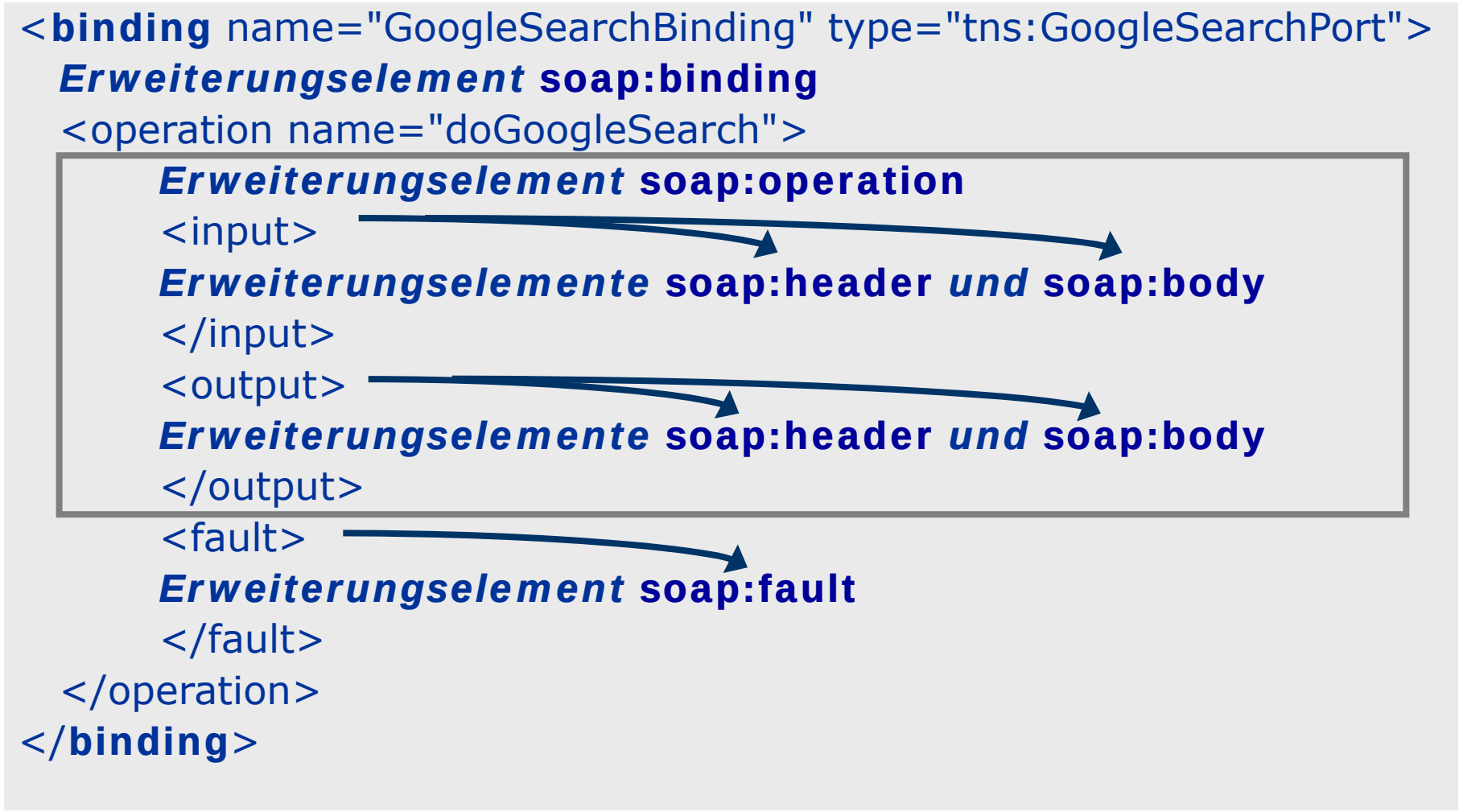
```
<body>

     <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>
```

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)
- 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?

Freie Universität Berlin

SOAP & WSDL

- ☑ Prinzipieller Aufbau
- ☑ Kodierung von RPCs
- ☑ Verarbeitung & Übertragung
- ☑ SOAP- und HTTP-Bindungen
- ☑ Vor- und Nachteile

Vorlesung morgen

- Überblick über die Projektarbeit
- Einführung Projektmanagement