Freie Universität Berlin

SOAP

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. (heute)	SOAP im Detail	SOAP	18./19.06
25.06.	WSDL im Detail	WSDL	
02.07.	Web Services in der Praxis & Ausblick		02./03.07
09.07.	Rückblick		
16.07.	Klausur		

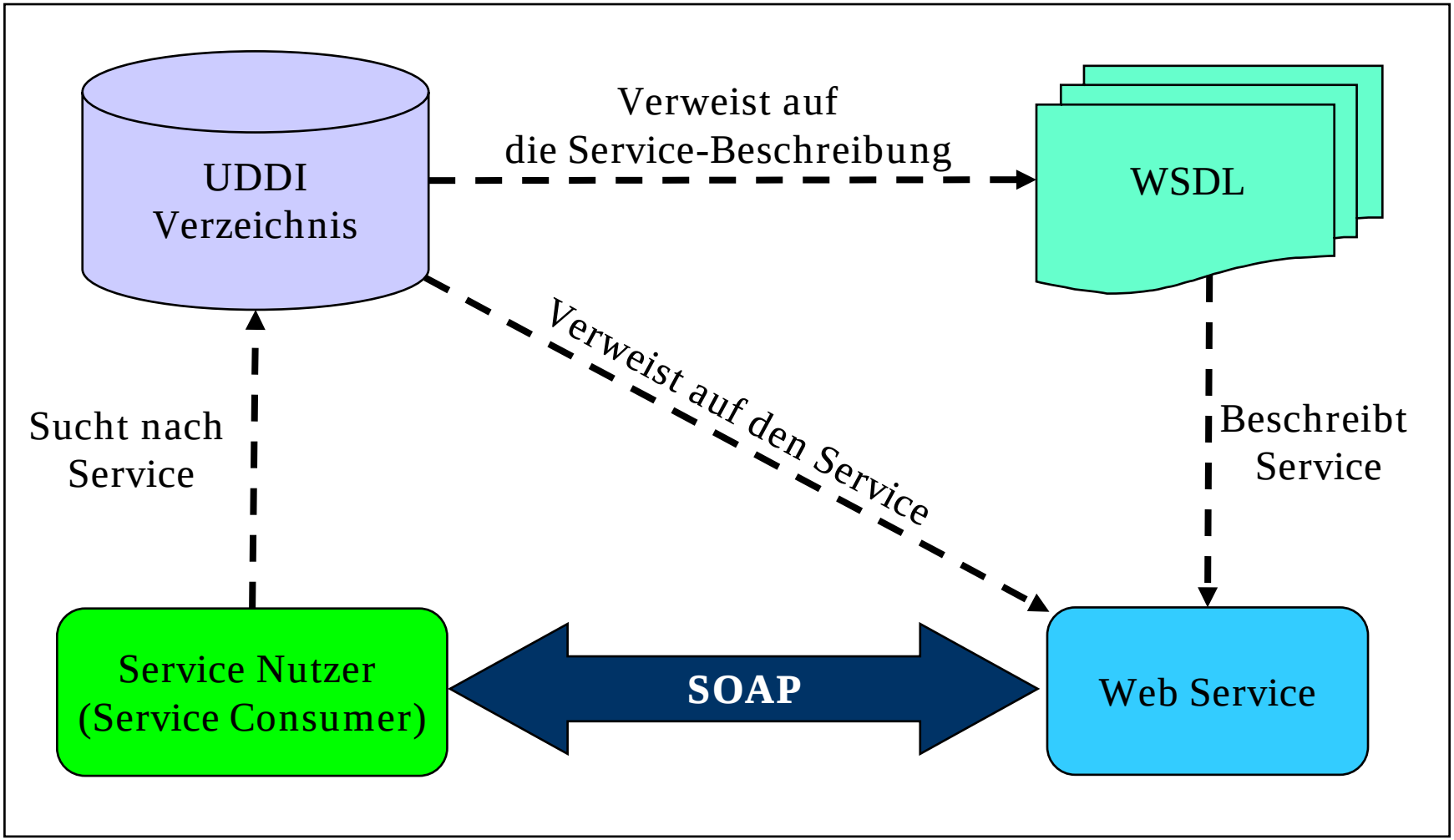
Heutige Vorlesung

letzte Woche

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

heutige Vorlesung → SOAP im Detail

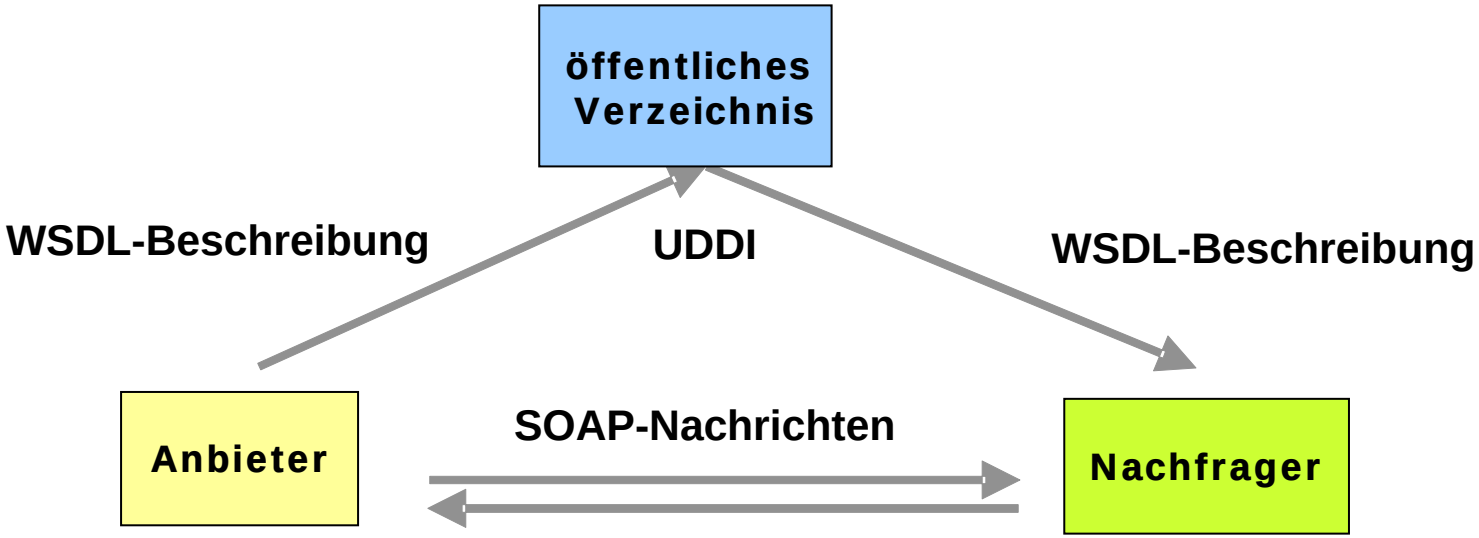
- prinzipieller Aufbau
- Kodierung von RPCs
- Verarbeitung & Übertragung
- Vor- und Nachteile


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

Was ist SOAP?

- Kommunikationskomponente von Web Services
- Protokoll für Nachrichtenaustausch zwischen Web Service-Konsument und Web Service-Anbieter
- XML-basiert
- Plattformunabhängig
- Programmierspracheunabhängig
- basierte auf Entwicklungen von Microsoft und IBM

Wozu SOAP?



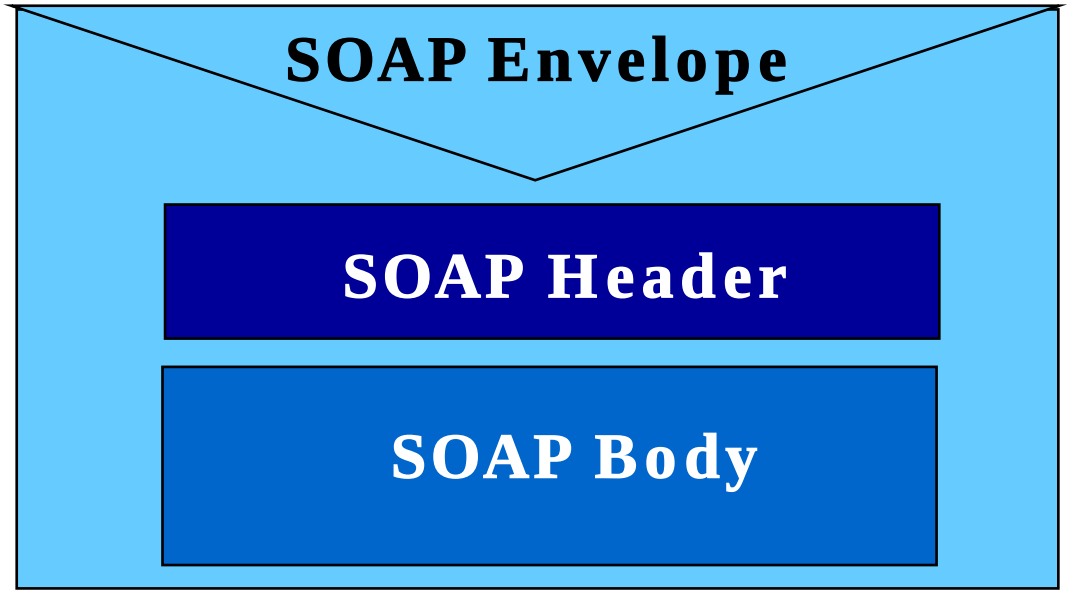
- SOAP: Format zum Austausch von Daten
- Warum spezielles Format und nicht einfach beliebige XML-Syntax zulassen?

Wozu SOAP?

- 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

Versionen

SOAP 1.1

W3C Note (2000)

- kein offizieller W3C-Standard, aber weit verbreitet
- wird von aktueller Version von WSDL benutzt
- wird von Google benutzt

SOAP 1.2

W3C Recommendation (seit 2003)

Second Edition -April 2007

- einzige Version, die vom W3C als Standard offiziell akzeptiert wurde
- Vorlesung benutzt diese Version

Zum Unterschied

- http://www.hadleynet.org/marc/whatsnew.html

Version = Envelope-Namensraum

Freie Universität Berlin

SOAP 1.1

- XMLSOAP-Namensraum

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env=" http://schemas.xmlsoap.org/soap/envelope">
   ...
</env:Envelope>
```

- unterschiedliche Namensräume

⇨ nicht kompatibel

SOAP 1.2

- W3C-Namensraum

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

Prinzipieller Aufbau (SOAP V.1.1)

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

- Wurzel-Element: Envelope aus SOAP-Namensraum
- kein W3C-Namensraum
- Header: optional
- Body: obligatorisch

SOAP Version 1.2

is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment.

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

- 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>
```

- 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 Block

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

Beispiel

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

Zusatz-information (Header Block)

Nachricht

mustUnderstand Attribut

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

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

Erweiterung von SOAP-Nachrichten

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

⇨ mächtiges Konzept für Versionierung

Hypothetisches Beispiel

```
<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

- 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

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

RPC-Aufruf

Freie Universität Berlin

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

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

```
<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

Code

```
<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>
```

- **Value** obligatorsich
- **env:Sender**: Anfrage nicht korrekt, nochmalige korrigierte Anfrage erwartet

Subcode

```
<env:Envelope … xmlns:rpc="http://www.w3.org/2003/05/soap-rpc">
  <env:Body>
    <env:Fault>
      <env:Code>
        <env:Value>env:Sender</env:Value>
        <env:Subcode>
          <env:Value>rpc:BadArguments</env:Value>
        </env:Subcode>
      </env:Code>
      <env:Reason>…</env:Reason>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

- Subcode optional
- Subcode genauso strukturiert, wie Code: Value (obligatorisch) + Subcode (optional)
- rpc:BadArguments: standardisierter Fehlercode

Reason

```
<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>
```

- mindestens ein Text-Element
- xml:lang-Attribut obligatorsich

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</number>
</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

b

a

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

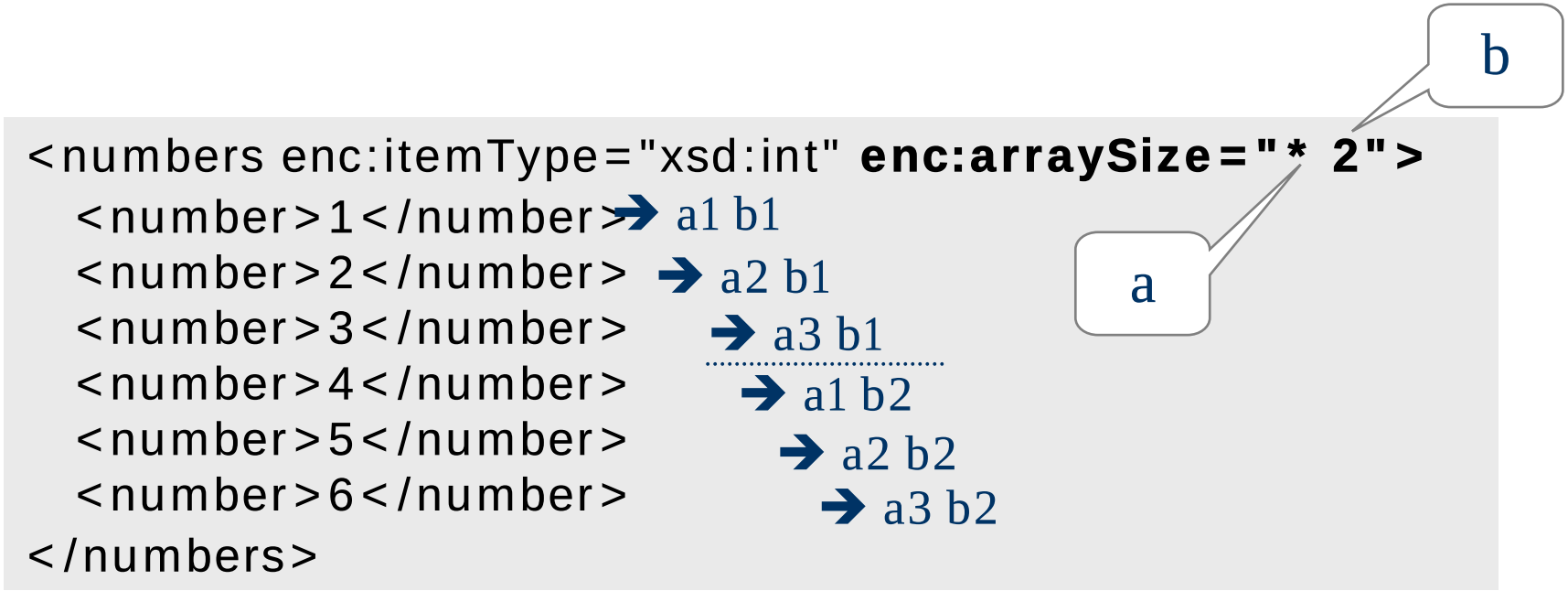
⇨ eindeutig auflösbar

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]
- 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"



- #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

- 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

Beispiel Google

```
<?xml version='1.0' encoding='UTF-8'?>
<env:Envelope
        xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Body>
    <doGoogleSearch xmlns="urn:GoogleSearch"
        env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <key xsi:type="xsd:string">328975487054809 7</key>
      <q xsi:type="xsd:string">Eine Anfrage</q>
      <start xsi:type="xsd:int">0</start>
      <maxResults xsi:type="xsd:int">10</maxResults>
        ...
    </doGoogleSearch>
  </env:Body>
</env:Envelope>
```

Verarbeitung von SOAP-Nachrichten

Allgemeine Anforderungen

Empfänger **muss verarbeiten**:

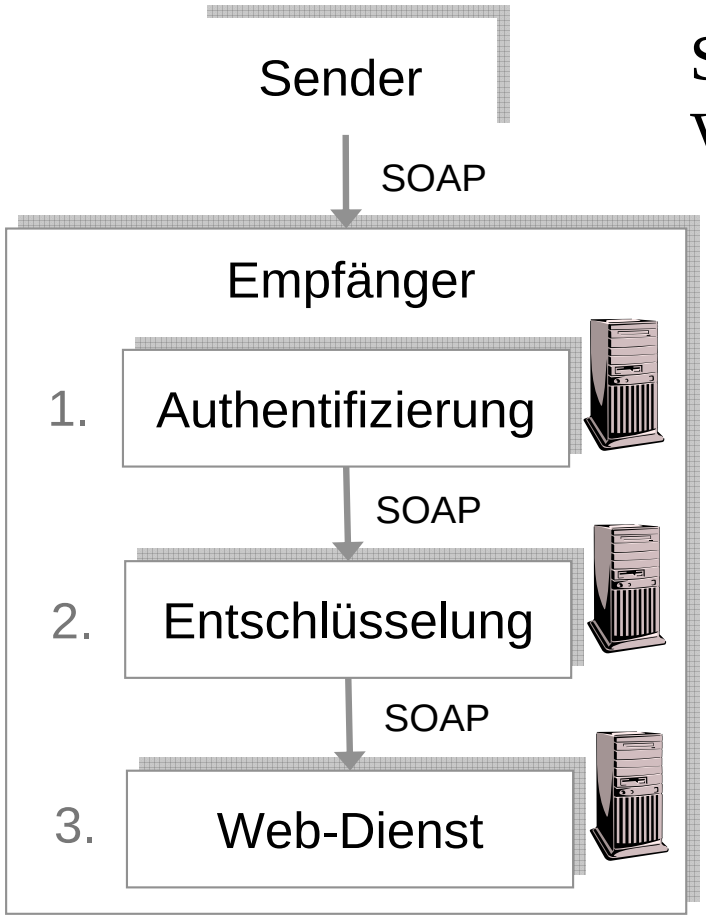
- Body
- Header Blocks mit mustUnderstand="true"

Empfänger **darf ignorieren**:

- Header Blocks mit mustUnderstand="false"
- Header Blocks ohne mustUnderstand-Attribut

  Grund: "false" Standardwert von mustUnderstand

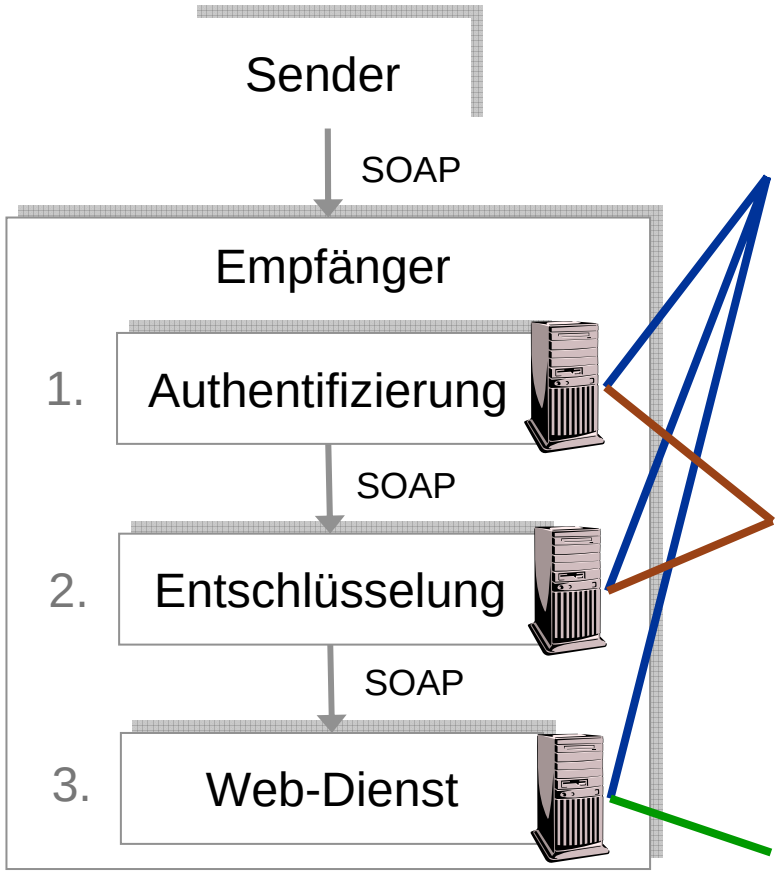
Schrittweise Verarbeitung



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

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

Identifizierung von SOAP-Knoten

Freie Universität Berlin

- SOAP-Knoten werden mit URIs identifiziert

- 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

- 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

- 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"

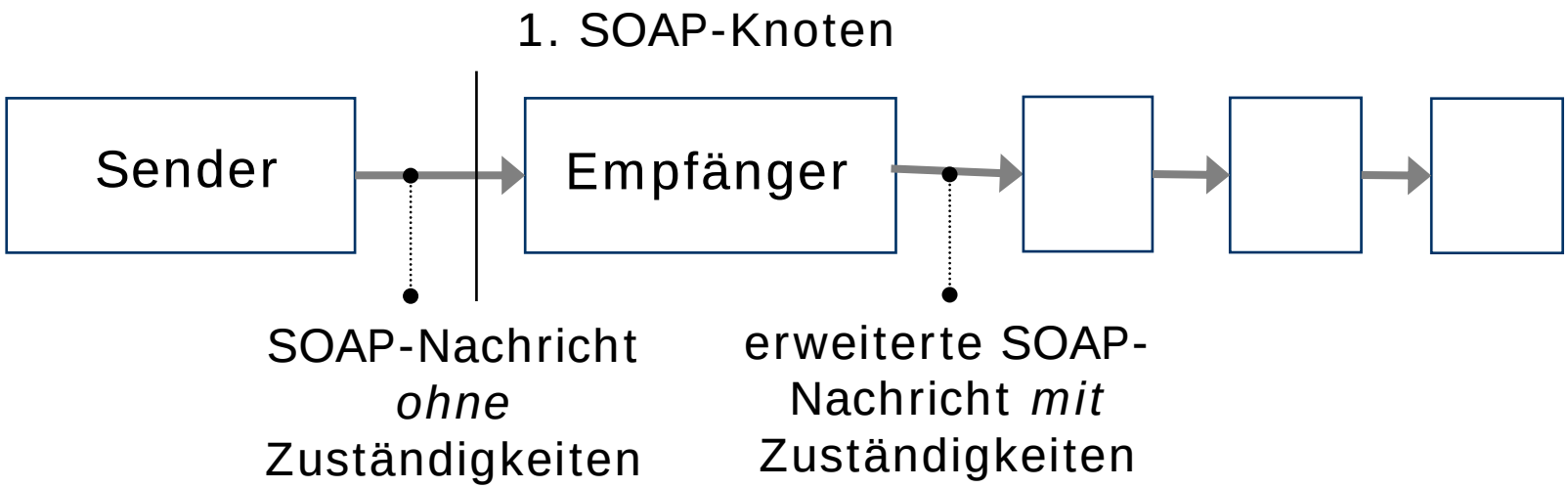
- Empfänger kann Header Block ignorieren
- Löschen von unbekannten Header Blocks erlaubt

Aufgabe des Endknoten

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

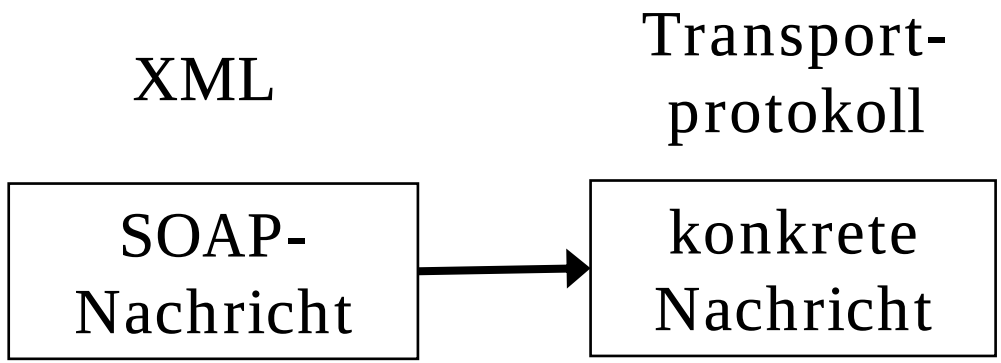
Festlegung der Zuständigkeiten?

- 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?
- z.B. für HTTP GET nicht trivial
- SOAP-Spezifikation schreibt nicht vor, **wie** Protokoll-Bindung spezifiziert wird

Anforderungen an Protokoll-Bindung

- 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

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

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

HTTP-GET vs. HTTP-POST

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 über HTTP POST: Anfrage

Freie Universität Berlin

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

URL

HTTP Header

```
<?xml version='1.0' encoding='UTF-8'?>
<env:Envelope ...>
  <env:Body>
    <doGoogleSearch xmlns="urn:GoogleSearch">
      <key xsi:type="xsd:string">3289754870548097</key>
      <q xsi:type="xsd:string">Eine Anfrage</q>
        ...
    </doGoogleSearch>
  </env:Body>
</env:Envelope>
```

SOAP-Nachricht

Daten

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

REpresentational State Transfer

- kein Protokoll sondern ein Architekturstil
- Verwaltet beliebige Menge von Ressourcen
- RESTful → eine Anwendung konform zum REST-Architekturstil

- Amazon – der populärster Anbieter einer REST-Anwendung
- Google bietet solche REST-Schnittstelle NICHT an

REST-Grundsatz

- REST-Architektur des WWW (Fielding 2000):
  jede Web-Ressource soll eindeutig über eine URI identifiziert werden

- Beispiel: online gebuchte Reise = Web-Ressource
- gebuchte Reise sollte daher auch über eine URI eindeutig identifiziert werden

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

- 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
```

WS-Addressing

Ziele von WS-Addressing

- neutrale Formulierung von Service-Endpunkten
- direkte Kodierung on Informationen in SOAP, die bisher in der Transportschicht angesiedelt waren
- Übertragung von SOAP-Nachrichten durch multiple Zwischenknoten unabhängig vom Transportprotokoll
- Unterstützung von asynchronen und Publish-Subscriber-Interaktionsmustern

Endpunkt-Referenz (endpoint reference)

- XML Dokument
- beinhaltet notwendige Informationen für die Verwendung eines Web Services-Endpunktes

Referenz-Eigenschaft (reference property)

- Erweitert die Identität des Endpunktes um zusätzliche Informationen

Referenz-Parameter (reference parameter)

- beschreibt spezifische Details des Aufrufs

Nachrichten-Informationselemente

Nachrichten-Informationselemente (message information header)

- **Nachrichten-ID** (wsa:MessageID) zur Identifikation der Nachricht in Zeit und Raum
- **Ziel-Adresse** (wsa:To) des Empfängers als URI
- **Quelle-Endpunkt** (wsa:From) des Absenders = Default-Ziel für die Antwort
- **Antwort-Endpunkt** (wsa:ReplayTo) spezifiziert den gewünschten Empfänger
- **Fehler** (wsa:FaultTo) für Behandlung von Fehlern zuständige Endpoint.
- ...

Vor- und Nachteile von SOAP

Vorteile

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

Nachteile

- RPCs über Firewalls hinweg oft nicht erwünscht
- zusätzlicher Verarbeitungsaufwand
- für viele notwendige Erweiterungen noch kein etablierter Standard

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

Web Service Interoperability Organisation

➔http://ws-i.org/

Wie geht es weiter?

Freie Universität Berlin

heutige Vorlesung

- ☑ SOAP im Detail

Übung diese Woche

- SOAP

nächste Woche

- WSDL im Detail