



SOAP im Detail

- prinzipieller Aufbau
- Datenkodierung
- Verarbeitung
- Übertragung
- SOAP-Engines
- Vor- und Nachteile

Warum SOAP?



- Warum Daten mit SOAP und nicht einfach mit reinem XML übertragen?
- SOAP bietet folgende Vorteile:
 - Konvention für RPCs und Messaging
 - Konvention für Arrays
 - Konventionen für Übertragung mit HTTP
 - Konzept für Erweiterung eines Nachrichtenformates

Prinzipieller Aufbau



Prinzipieller Aufbau



```

<?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

Eine SOAP-Anfrage an Google



```

<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Body>
    <doGoogleSearch xmlns="urn:GoogleSearch">
      <key xsi:type="xsd:string">3289754870548097</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>
    
```

- doGoogleSearch(key, q, start, maxResults,...)
- doGoogleSearch aus Google-Namensraum
- Datentypen aus XML-Schema

Und die Antwort von Google

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

- Antwort: doGoogleSearchResponse(return(...))
- Datentyp ns1:GoogleSearchResult in WSDL-Beschreibung definiert.

SOAP-Version = Envelope-Namensraum

- SOAP 1.1
- W3C-Note (2000)

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

- SOAP 1.2
- W3C-Standard (Juni 2003)

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

aktuelle Version

Versionen

SOAP 1.1

- kein offizieller W3C-Standard, aber heute noch weit verbreitet
- Google benutzt diese Version

SOAP 1.2

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

Zum Unterschied

- knapp: <http://www.w3.org/TR/soap12-part0>, Kapitel 6.
- ausführlich: <http://www.hadley.net.org/marc/whatsnew.html>

Nachrichteninhalt

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

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

Briefkopf

```
<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
Blocks

- **Header**: beliebige XML-Inhalte erlaubt
- Struktur von Anwendung festgelegt
- **Header Blocks**: Kind-Elemente von Header
- Zusatzinformation zur eigentlichen Nachricht

Beispiel

```
<env:Envelope ...>
  <env:Header>
    <alertcontrol xmlns="http://example.org/alertcontrol">
      <priority>1</priority>
      <expires>2003-10-12T14:00:00-05:00</expires>
    </alertcontrol>
  </env:Header>
  <env:Body>
    <alert-msg xmlns="http://example.org/alert">
      Pick up Mary at school at 2pm!
    </alert-msg>
  </env:Body>
</env:Envelope>
```

Zusatz-
information
(Header
Block)

Nachricht

→ Erweiterung des ursprünglichen
Nachrichtenformates

Obligatorische & optionale Header Blocks

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

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

© Klaus Schild, 2004

13

Erweiterbarkeit von SOAP-Nachrichten

- Konzept: Trennung von Zusatzinformationen von eigentlicher Nachricht
- Nachrichtenformat kann erweitert werden, ohne ursprüngliches Format (Body) zu modifizieren.
- Erweiterungen können jeweils obligatorisch oder optional sein.
- Erweiterung unabhängig voneinander

© Klaus Schild, 2004

14

Beispiel

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

eigentliche
Nachricht
unverändert

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

```
<env:Header>
  ...
  <Expires>
  9-6-2004, 5:16:18 pm
</Expires>
</env:Header>
```

Erweiterungen unabhängig voneinander

© Klaus Schild, 2004

15

Anfrage-Antwort-Muster

- SOAP selbst unterstützt nur Einweg-Kommunikation
- Anfrage-Antwort-Muster können auf verschiedene Weise realisiert werden:
 - mit HTTP
 - mit eindeutiger Referenz im Briefkopf
- Vorteil: Unabhängig vom Übertragungsprotokoll

© Klaus Schild, 2004

16

Referenz im Briefkopf

- Anfrage enthält eindeutige Referenz („Mein Zeichen“), worauf anschließend verwiesen werden kann:

```
<env:Header>
  <wsu:identifier xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility/"
    URI als eindeutige Referenz
  </wsu:identifier>
</env:Header>
```

- so beliebig komplexe Interaktionen möglich
- oft verwendet, aber noch *kein* etablierter Standard
- Alternative zu `wsu:identifier`:
MessageID aus Namensraum von WS-Addressing

→ eingeschränkte Interoperabilität

© Klaus Schild, 2004

17

Entfernter Prozeduraufruf (RPC)

```
Procedure(In-Parameter-1,...,In-Parameter-n)
<env:Envelope ...>
  <env:Body>
    <m:Procedure xmlns:m="URI">
      <m:In-Parameter-1>...</m:In-Parameter-1>
      ...
      <m:In-Parameter-n>...</m:In-Parameter-n>
    </m:Procedure>
  </env:Body>
</env:Envelope>
```

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

© Klaus Schild, 2004

18

Entfernter Prozeduraufruf (RPC)

physischer Aufenthaltsort (URI) der Prozedur:

- außerhalb von SOAP im Transportprotokoll spezifiziert
- kann aber auch im Briefkopf repräsentiert werden:

```
<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>
```

- WS-Addressing allerdings *kein* etablierter Standard

Ergebnis eines RPCs

```
<env:Envelope ...>
  <env:Body>
    <m:ProcedureResponse xmlns:m="URI">
      <m:Out-Parameter-1>...</m:Out-Parameter-1>
      ...
      <m:Out-Parameter-n>...</m:Out-Parameter-n>
    </m:ProcedureResponse>
  </env:Body>
</env:Envelope>
```

- *ProcedureResponse* Kind-Element von Body
- **Beachte:** Name für *ProcedureResponse* beliebig
- Rückgabewerte Kind-Elemente von *ProcedureResponse*: Ausgangsparameter und eigentliches Ergebnis
- In-Out-Parameter erscheinen im Aufruf und der Antwort

Ausgezeichneter Ergebnistyp

```
public Out-Parameter-i Procedure(...)
<env:Envelope ...>
  <env:Body>
    <m:ProcedureResponse xmlns:m="URI"
      xmlns:rpc="http://www.w3.org/2003/05/soap-rpc">
      <rpc:result>m:Out-Parameter-i</rpc:result>
      <m:Out-Parameter-1>...</m:Out-Parameter-1>
      ...
      <m:Out-Parameter-n>...</m:Out-Parameter-n>
    </m:ProcedureResponse>
  </env:Body>
</env:Envelope>
```

- `rpc:result`: Verweis auf ausgezeichnetes Ergebnis

Datenkodierung



Datenkodierung

- mit SOAP werden Daten übertragen
- Daten müssen auf irgendeine Weise dargestellt werden
- **Beispiel:** Als Parameter soll Array von drei Zahlen 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>
```

- Und wie multidimensionale Arrays darstellen?

Datenkodierung

- verwendete Datenkodierung kann als Attribut angegeben werden:
`env:encodingStyle="URI"`
- `"URI"`: eindeutiger Bezeichner für verwendetes Kodierungsschema
- verschiedene Kodierungsschemata innerhalb einer SOAP-Nachricht erlaubt

Beispiel

```
<?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">3289754870548097</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>
```

Kodierungsschemata

- Kodierungsschema kann anwendungsspezifisch sein: "http://www.ibm.com/soap-encoding" (fiktiv)
- Anwendung muss entspr. Kodierungsschema kennen.
- Darstellung von grundlegenden Dingen durch vordefiniertes Kodierungsschema festgelegt:
 - **"http://www.w3.org/2003/05/soap-encoding"**
 - legt Darstellung folgender Dinge fest:
 - RPCs: wie oben beschrieben
 - Referenzen: mit Attributen id und ref
 - Arrays: mit SOAP-Arrays

SOAP-Arrays

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

- Array mit zwei Elementen vom Typ xsd:int.
- enc:arraySize="*": Array mit beliebig vielen Elementen

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": nx2-Matrix
- enc:arraySize="* *": **nicht erlaubt!**

Verarbeitung



Verarbeitung einer SOAP-Nachricht

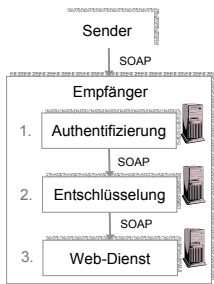
Empfänger *muss* verarbeiten:

- Body
- Header Blocks mit mustUnderstand="true"

Empfänger *kann* ignorieren:

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

Schrittweise Verarbeitung



SOAP-Nachrichten können schrittweise verarbeitet werden, z.B.:

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

Vorteile

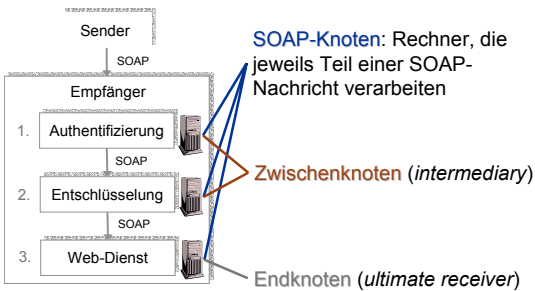
Aufgabenteilung

- spezialisierte Server
- z.B. Authentifizierungs-Server und Server, auf dem der eigentliche Dienst läuft
- Authentifizierungs-Server kann *vor* der Firewall liegen, die anderen Server *hinter* der Firewall

Lasterverteilung

- Verteilung auf verschiedene Server

SOAP-Knoten

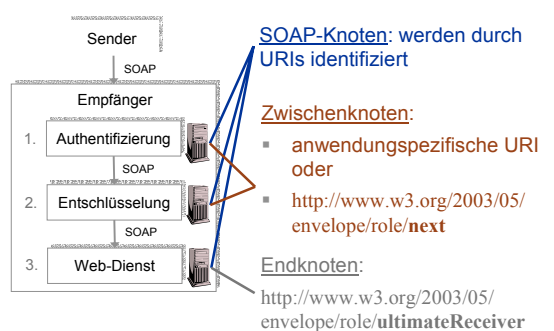


SOAP-Knoten: Rechner, die jeweils Teil einer SOAP-Nachricht verarbeiten

Zwischenknoten (*intermediary*)

Endknoten (*ultimate receiver*)

Bezeichnung von SOAP-Knoten



SOAP-Knoten: werden durch URIs identifiziert

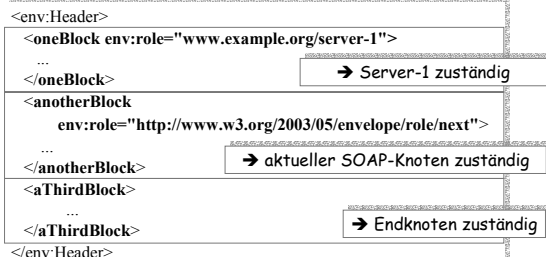
Zwischenknoten:

- anwendungsspezifische URI oder
- <http://www.w3.org/2003/05/envelope/role/next>

Endknoten:

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

Festlegung der Zuständigkeiten



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

Aufgabe eines Zwischenknoten

1. verarbeitet Header Blocks mit
 - **role="http://www.w3.org/2003/05/envelope/role/next"**
 - **role="URI"**, wobei "URI" den betreffenden Zwischenknoten bezeichnet
2. löscht alle verarbeiteten Header Blocks
3. kann neue Header Blocks hinzufügen
4. entscheidet, welcher Knoten nächster SOAP-Knoten ist
5. leitet modifizierte SOAP-Nachricht an diesen SOAP-Knoten weiter

Beachte: Für Header Blocks *ohne* `mustUnderstand="true"` kann Verarbeiten auch einfaches Löschen bedeuten!

Aufgabe des Endknoten



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

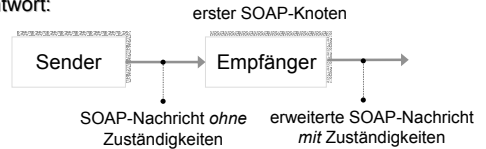
Beachte: Body muss *immer* verstanden werden.

Festlegung der Zuständigkeiten in SOAP



- Vorteile der schrittweisen Verarbeitung klar: Aufgabenteilung und Lastverteilung
- Warum aber Zuständigkeiten in SOAP-Nachricht festlegen?
- Zuständigkeiten sollten doch für Sender *transparent* (nicht sichtbar) sein!

Antwort:



Übertragung



Protokoll-Bindungen (Bindings)



- Spezifikation, wie SOAP-Nachrichten mit einem bestimmten Protokoll übertragen werden
- Beschreibt, wie SOAP-Nachrichten in konkrete Nachrichten umgewandelt (serialisiert) werden.
- *nicht* vorgeschrieben, wie eine Protokoll-Bindung spezifiziert wird
- **Beachte:** innerhalb einer Kette von SOAP-Knoten unterschiedliche Protokoll-Bindungen möglich

Protokoll-Bindungen



- konkrete Nachricht meist XML, kann aber auch beliebig anderes Format sein:
z.B. komprimiertes BinärfORMAT
- **wichtig:** Umwandlung in konkrete Nachricht *ohne* Informationsverlust, Umwandlung also symmetrisch



Standardisierte Protokoll-Bindungen



- HTTP-Bindung bisher als *einzige* Protokoll-Bindung für SOAP standardisiert
- zwei unterschiedliche HTTP-Bindungen:
 - für HTTP-POST
 - für HTTP-GET
- SMTP-Bindung trivial:
einfach SOAP-Nachricht als E-Mail zu Empfänger senden

HTTP



- Anfrage-Antwort-Muster
- zustandsloses Protokoll
- jedes Anfrage-Antwort-Paar isolierte, abgeschlossene Einheit
- Daten von vorherigen Anfragen oder Antworten stehen später nicht mehr zur Verfügung

Die wichtigsten HTTP-Formate



HTTP GET

- fragt Web-Ressource mit einer URI ab
- Parameter können in eine URI kodiert werden, z.B.:
`http://myserver.com/search?q=hello%20there`

HTTP POST

- fragt Web-Ressource ab, übermittelt gleichzeitig Daten

SOAP über HTTP POST: Anfrage



POST /search/beta2/doGoogleSearch HTTP/1.1

Host: api.google.com

Content-Type: application/soap+xml; charset="utf-8"

Content-Length: nnnn

```
<?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 über HTTP POST: Antwort



HTTP/1.1 200 OK

Content-Type: application/soap+xml; charset="utf-8"

Content-Length: nnnn

```
<?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 über HTTP GET



GET /Reservations/itinerary?reservationCode=FT35ZBQ HTTP/1.1

Host: travelcompany.example.org

Accept: application/soap+xml

- ruft `getItinerary(reservationCode)` auf
- Vorteil: entspricht Grundsatz, dass jede Web-Ressource über URI identifiziert wird
- gebuchte Reise ist Web-Ressource, URI sollte daher Reservierungsnummer (`reservationCode`) enthalten
- HTTP GET empfohlen, wenn *alle* Parameter Web-Ressourcen identifizieren
- Nachteil: offen, wie aus SOAP-Nachricht URI generiert wird



SOAP-Engines

SOAP-Engines



Problem

- Wie sende ich SOAP-Nachrichten von A nach B?
- Wer empfängt die Nachrichten und gibt sie an den Web-Dienst weiter?
- Wie bekomme ich die Antwort zurück?

Lösung

- SOAP-Engine, d.h. eine spezielle Software zur Verarbeitung von SOAP-Nachrichten

SOAP-Engines



- typischerweise mit HTTP-, SMTP oder FTP-Server betrieben
- bekannte SOAP-Engines:



→ <http://ws.apache.org/axis/>



SOAP::Lite for Perl

→ <http://www.soaplite.com/>



Vor- und Nachteile

Vorteile



- + etablierter Standard, wird u.a. in .net verwendet
- + für RPCs und Messaging geeignet
- + RPCs über Firewalls hinweg möglich
- + einfach erweiterbar
- + Erweiterungen unabhängig voneinander



Nachteile



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

Beispiel: wsu:identifizier vs. wsa:MessageID

- Protokoll-Bindungen können unterschiedliche Semantik haben

Beispiel: SMTP-Binding asynchron, HTTP-Binding synchron

- heutzutage nicht vollständig interoperabel

→ <http://www.ws-i.org>



Wie geht es weiter?



heutige Vorlesung

- ☑ Nachrichtenformat SOAP

nächste Woche

- Schnittstellenbeschreibung WSDL

30.6.

- Web-Dienste in der Praxis

Anhang



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="cs">Chyba zpracování</env:Text>
      </env:Reason>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

- Code und Reason obligatorisch
- Code: zur maschinellen Verarbeitung
- Reason: zusätzliche Information, nicht zur maschinellen Verarbeitung

Fehlermeldungen: 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="cs">Chyba zpracování</env:Text>
      </env:Reason>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

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

Fehlermeldungen: 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="cs">Chyba zpracování</env:Text>
      </env:Reason>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

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

Fehlermeldungen: 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:Text xml:lang="en-US">Processing error</env:Text>
        <env:Text xml:lang="cs">Chyba zpracování</env:Text>
      </env:Reason>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

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