7363 - Web-basierte Anwendungen

Eine Vertiefungsveranstaltung
mit Schwerpunkt auf XML-Technologien

SOAP

Vormals: **S**imple **O**bject **A**ccess **P**rotocol

SOAP

- Grenzen von XML-RPC
  - Datentypen
    - Definition *eigener* Datentypen?
    - Geringe Präzision: Keine Regexp, ...
    - Man merkt: XML Schema kam erst später.
  - Der Dokumentenmodus (EDI) fehlt
  - Introspektion
    - Notdürftig per Konvention ergänzt, aber nicht Teil der Spezifikation
    - Dokumentation der Methoden zu unpräzise
  - Transaktionsnetzwerke?
    - XML-RPC ist für *einfache* C/S-Beziehungen gedacht
  - Sicherheitsaspekte
  - Öffentliche XML-RPC Dienste
    - Wie finden? Wie benutzen? Wie gegen Missbrauch sichern?

SOAP

- Konzeptionelle Änderungen beim Wechsel von XML-RPC zu SOAP
  - Transportschicht
    - Aufgabe des "*request/response*"-Modells, Übergang zu "***conversations***"
    - Neu: Verschiedene Transportprotokolle möglich
    - Neu: ***message paths, intermediaries***
  - *Packaging* (*Marshalling*)
    - "***Envelope***" ersetzt sowohl "*methodCall*" als auch "*methodResponse*"
    - Keine festen *Encoding*-Regeln mehr
    - Regelfall: XML Schema-Datentypen für die Typisierung
    - XML-Namensräume spielen eine große Rolle
    - Neu bei *Arrays*: Mehrdimensional, Feldlängen, spärlich besetzte A., Teile
    - Keine feste Regel für die Übertragung von Methodennamen
    - Unterstützung sowohl für RPC- als auch für EDI-Modus
  - Fazit: Viel mehr Freiheiten - aber auch neue Verunsicherung...

SOAP

- Ein Web Services-Szenario: "Wetterfront"
  - Ziel:
    - Räumlich aufgelöste Darstellung der Zeitentwicklung einiger Wetterparameter wie Temperatur, Luftdruck, Luftfeuchte, Niederschlagsrate, Windgeschwindigkeit, Windrichtung
  - Der Weg:
    - Suche nach Wetterstationen im Zielgebiet
    - Regelmäßige Abfrage der Wetterdaten aller gefundenen Stationen
    - Animierte Anzeige im Zeitraffer über dem Zielgebiet, z.B. als Ausbreitung einer Wetterfront, als Dichtewelle, etc.
  - Herausforderungen:
    - Wie findet man die Wetterstationen?
    - Wie kann man sich zahlreichen WS-Interfaces dynamisch anpassen?
    - Wie geschehen die Abfragen?

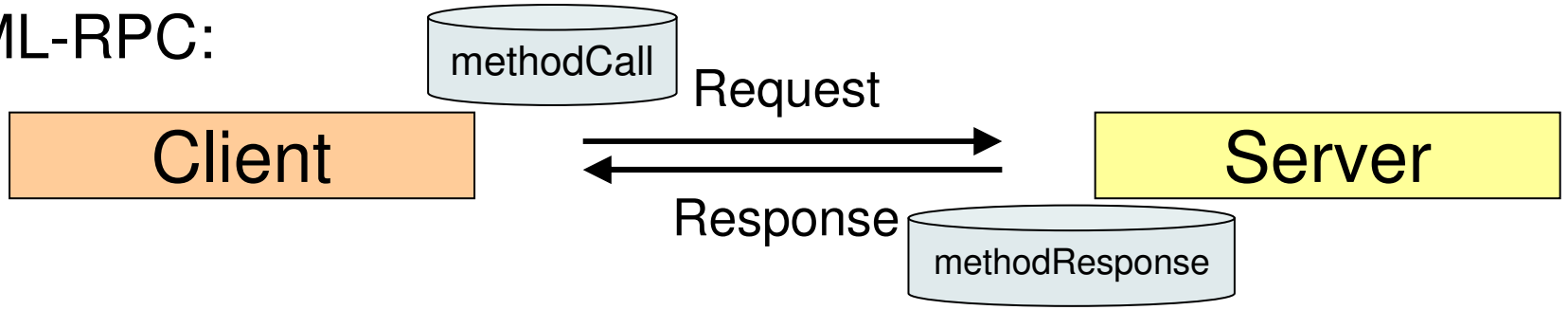
SOAP

Der Technologie-Stack von Web Services:

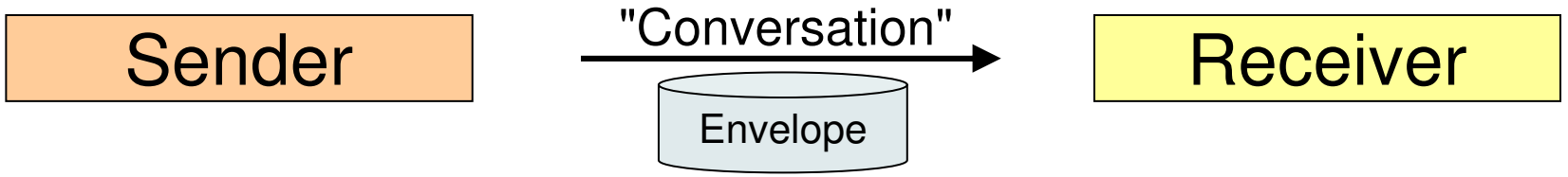
Discovery	**UDDI**, WS-Inspection	
Description	**WSDL**, RDF/DAML	Neu im SOAP-Kontext!
Packaging	XML-RPC Regeln, **SOAP**	**Aktuelles Thema**
Transport	**HTTP**, SMTP, FTP, Jabber, MQSeries, plain TCP, Instant Messaging, ...	
Network	**TCP/IP**, UDP/IP, OSI X.25, ...	Vorausgesetzt

SOAP

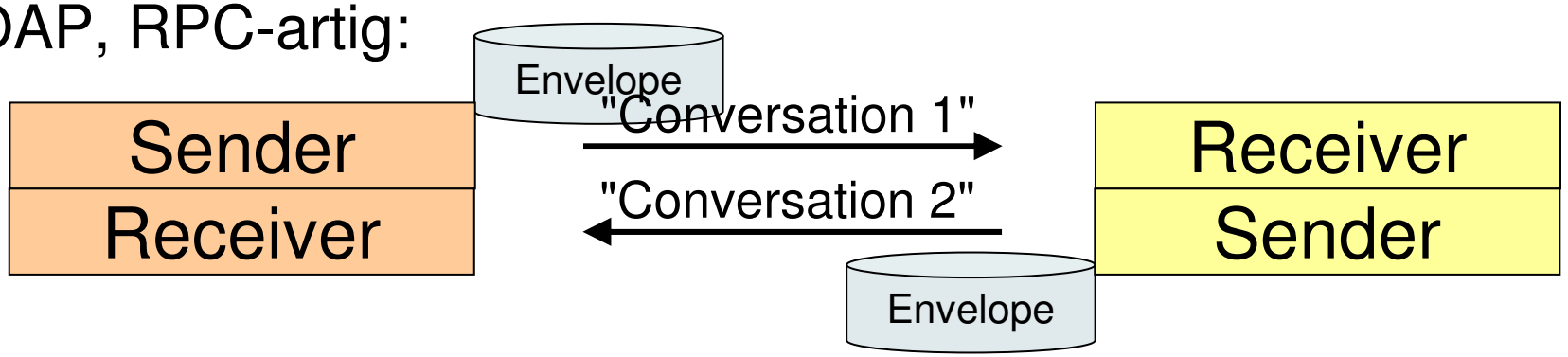
XML-RPC:



SOAP, elementar:



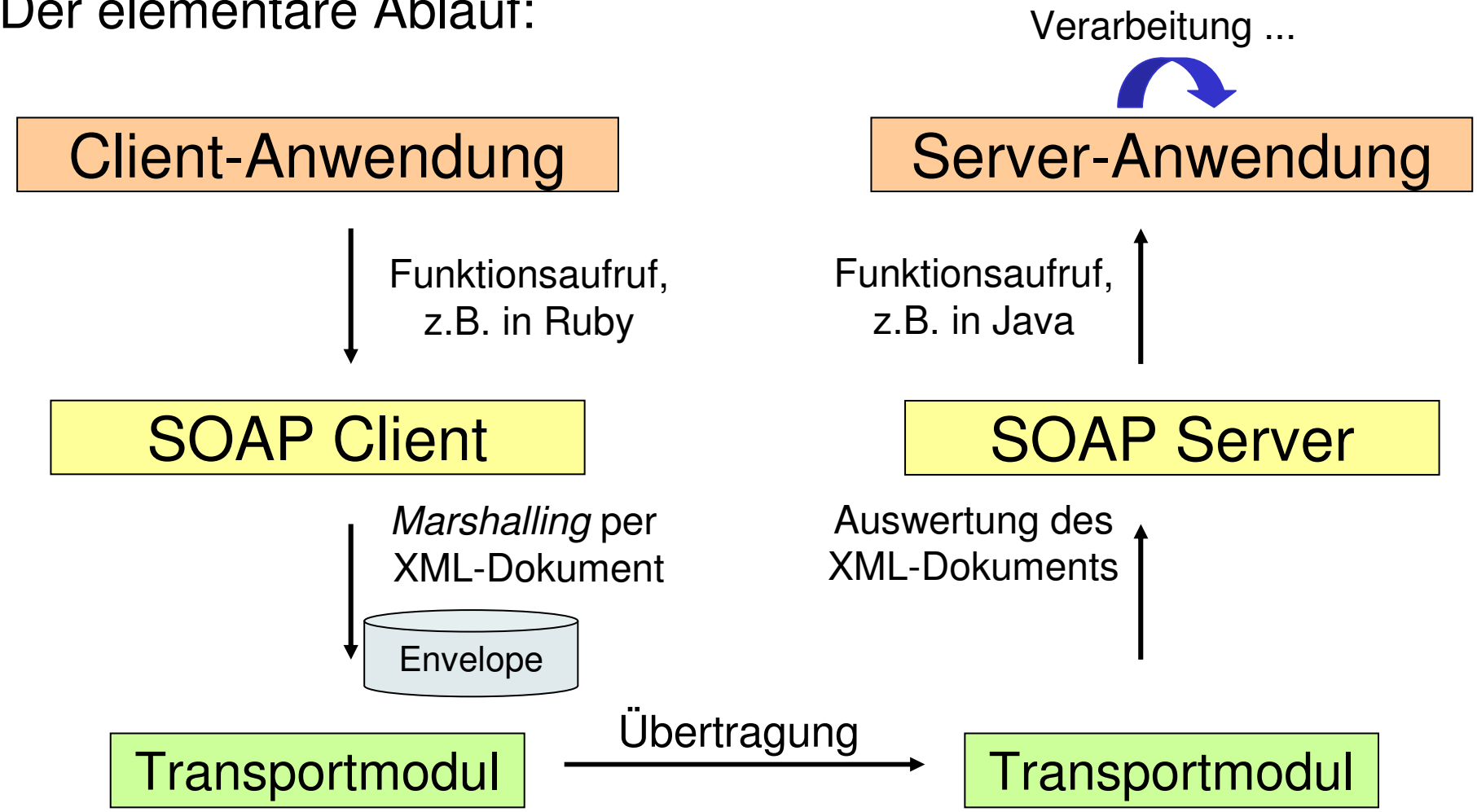
SOAP, RPC-artig:



Mit dem SOAP-Modell ist auch das *request/response* Modell realisierbar!

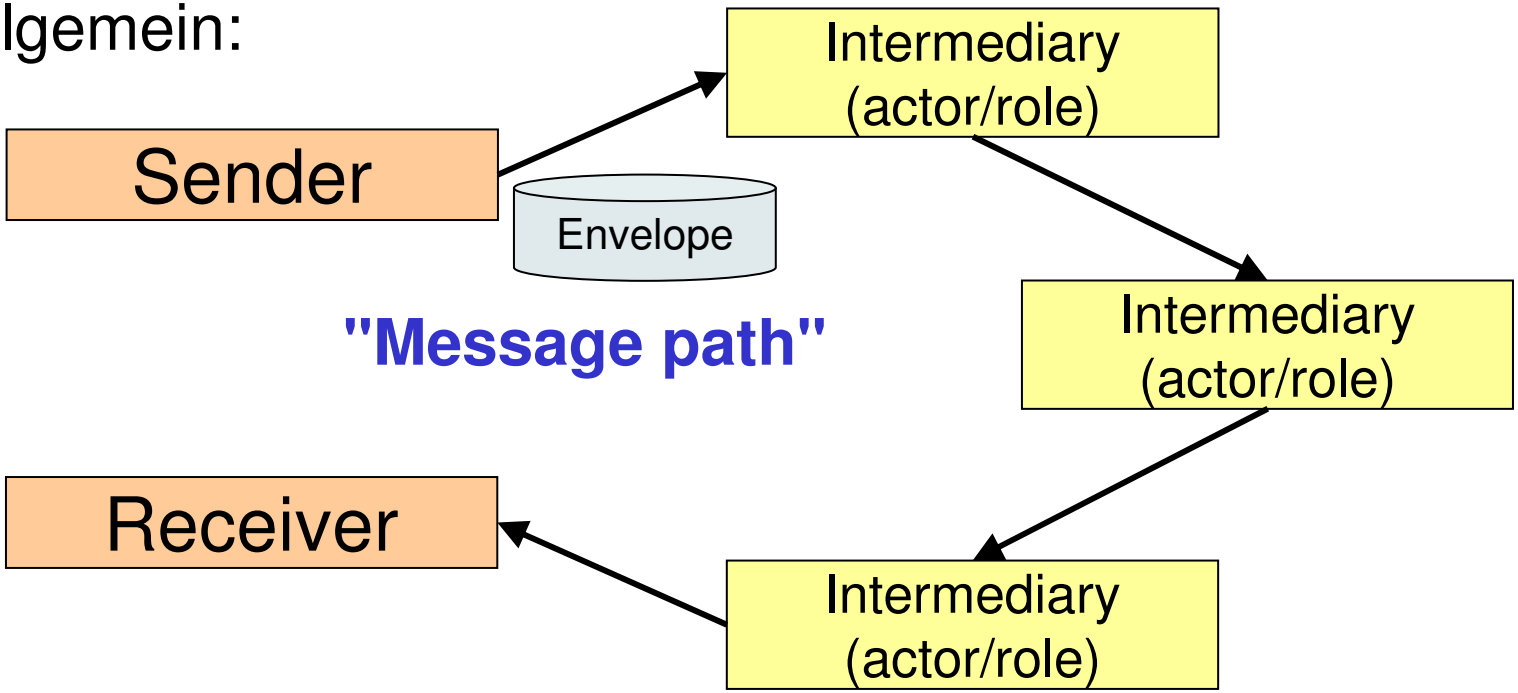
SOAP

Der elementare Ablauf:



SOAP

SOAP, allgemein:



Intermediäre spielen verschiedene **Rollen** (SOAP 1.1: "actor", 1.2: "role")
(Registrierungen, Policy-Verwaltung, Sicherheitsaspekte, Routing, ...)
Damit die eigentlichen Dokumente nicht verändert werden müssen,
gibt es "**Header**"-Elemente

SOAP Bindings

Zwischen ***Packaging*** und **Transport**

SOAP

- SOAP *Bindings*
  - Grundsätzliches
    - Generell kann SOAP viele Transportprotokolle nutzen.
    - Für spezielle Anforderungen sind einige Protokolle dennoch besser geeignet als andere.
      - So können *request/response*-Paare innerhalb einer HTTP *session* erfolgen
      - SMTP verursacht dagegen längere Latenzzeiten und benötigt einen Mechanismus, eine *response*-Nachricht mit der richtigen *request*-Nachricht zu korrelieren.
  - HTTP Binding
    - (Bsp. unten), Hinweis auf `SOAPAction`-Header in SOAP 1.1
  - SMTP Binding
    - (Bsp. unten), Hinweis auf Request ID
  - Medientyp, generell:
    `text/xml (SOAP 1.1), application/soap+xml (1.2)`

SOAP

- SOAP: HTTP-Binding (Beispiel zu SOAP 1.1)

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope xmlns:SOAP-ENV=
  "http://schemas.xmlsoap.org/soap/envelope/ " … > ...
</SOAP-ENV:Envelope>
```

"Request"

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

<SOAP-ENV:Envelope xmlns:SOAP-ENV=...
```

"Response"

SOAP

- SOAP: HTTP-Binding (Beispiel zu SOAP 1.2)

```
POST /Reservations HTTP/1.1
Host: travelcompany.example.org
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn

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

"SOAP Request-Response pattern"

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

"Response"

SOAP

- SOAP: HTTP-Binding (weiteres Beispiel zu SOAP 1.2)

```
GET /travelcompany.example.org/reservations?code=FT35ZBQ HTTP/1.1
Host: travelcompany.example.org
Accept: text/html;q=0.5, application/soap+xml
```

"SOAP Response pattern"

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn

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

"Fault Response"

SOAP

- SOAP: SMTP-Binding (Beispiel zu SOAP 1.2)

```
From: a.oyvind@mycompany.example.com
To: reservations@travelcompany.example.org
Subject: Travel to LA
Date: Thu, 29 Nov 2001 13:20:00 EST
Message-Id:
  <EE492E16A090090276D208424960C0C@mycompany.example.com>
Content-Type: application/soap+xml


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

Übermittlung des Request-Dokuments

SOAP

- SOAP: SMTP-Binding (Beispiel zu SOAP 1.2)

```
From: reservations@travelcompany.example.org
To: a.oyvind@mycompany.example.com
Subject: Which NY airport?
Date: Thu, 29 Nov 2001 13:35:11 EST
Message-Id: <200109251753.NAA10655@travelcompany.example.org>
In-reply-to:
  <EE492E16A090090276D208424960C0C@mycompany.example.com>
Content-Type: application/soap+xml

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

Antwort mit dem Response-Dokument

SOAP: Die *Packaging*-Ebene

Historische Entwicklung,
allgemeine Struktur von SOAP-Dokumenten

SOAP

- SOAP - oder: Die Packaging-Ebene
  - Warum ist ein Standard für die Serialisierung sinnvoll?
  - Dazu ein Vorlesungs-Versuch:

- **Vorlesungsaufgabe** (wenige Minuten)
  - Entwerfen Sie einen kleinen XML-Dokumententypen zur Übertragung von Telefonnummern.
    - Beachten Sie dabei die Strukturierung von Telefonnummern

- Auswertung (an der Tafel)
  - Wir sammeln die Varianten
  - Welche Probleme fielen Ihnen auf?

- Fazit:
  - Ein ordnender Standard sollte die Variantenvielfalt eindämmen!

SOAP

- Die Entstehung von SOAP
  - SOAP 1.0
    - Von den Machern von XML-RPC
    - Durch Abspaltung und Weiterentwicklung ca. 1999 entstanden
    - Quelle: http://www.scripting.com/misc/soap1.txt ? (unklar)
    - Heute ohne weitere Bedeutung, da durch SOAP 1.1 ersetzt
  - SOAP 1.1
    - 8. Mai 2000: Immer noch "Simple Object Access Protocol"
    - Quelle: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/
    - Autoren: I.w. die Autoren von SOAP 1.0
    - Status:
      - Nur "Note" - keineswegs "Recommendation" !
      - Formalisierter Abschluss der bisherigen Entwicklung, Anregung zur Bildung einer Standardisierungsgruppe des W3C zur Fortführung.
      - Grundlage zahlreicher Implementierungen auch heute (2005) noch.

SOAP

- Die Entstehung von SOAP
  - SOAP 1.2
    - 24. Juni 2003:
      - Erstes Release als W3C "Recommendation"
      - 4 Teile: *Primer*, *Messaging Framework*, *Adjuncts* und *Spec. Assertion and Test Collection*
      - SOAP wird zum Eigennamen (ist kein Akronym mehr)
    - Quelle:
      - http://www.w3.org/TR/soap/,
      - http://www.w3.org/TR/soap12/
    - Autoren:
      - Neues Team, von Microsoft, IBM, Canon, Sun Microsystems
    - Status:
      - W3C "*Recommendation*"
      - Strengere Formulierung, Anspruch: Interoperabilität sichern!
      - Ähnlich wie, aber nicht abwärtskompatibel zu SOAP 1.1

SOAP

- SOAP-Versionierung
  - Unterscheidung geschieht über Namensraum-URIs !
  - XML Schema statt DTD bildet die Grundlage, insb. bei SOAP 1.2

  - SOAP 1.1
    - SOAP-ENV http://schemas.xmlsoap.org/soap/envelope
    - SOAP-ENC http://schemas.xmlsoap.org/soap/encoding
  - SOAP 1.2
    - env http://www.w3.org/2003/05/soap-envelope
    - enc http://www.w3.org/2003/05/soap-encoding
    - rpc http://www.w3.org/2003/05/soap-rpc

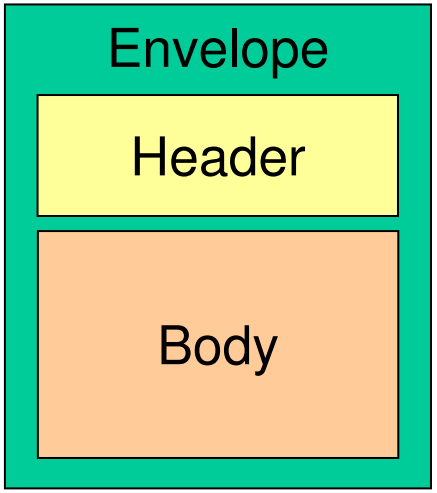
  - Ggf. gemeinsame Grundlagen:
    - xs, xsl http://www.w3.org/2001/XMLSchema
    - xsi http://www.w3.org/2001/XMLSchema-instance
    - tns (der jeweilige "*target namespace*")

SOAP

- Die Entstehung von SOAP
  - SOAP 1.2 vs. 1.1
    - Attribut "role" ersetzt "actor"
    - Neu: "relay"
    - *Fault names* reformiert: *qnames* anstatt *dot notation*
    - Änderungen in diversen Attributwerten
    - Neu: XML Base & XML Infoset als Grundlagen
    - Geändert: `encodingStyle, NotUnderstood, MustUnderstand` ...
    - Keine Elemente mehr im Anschluss an "Body"!
    - Neu: Bindings (SOAP 1.1 kennt nur HTTP Binding)
    - Quelle z.B.: Marc Hadley, Sun Microsystems http://www.idealliance.org/papers/xmle02/dx_xmle02/papers/02-02-02/02-02-02.html
    - Änderung bez. Arrays
      - (noch auszuarbeiten)

SOAP

- SOAP: XML-Dokumententyp
  - Sowohl *request* als auch *response* sind Exemplare des XML-Dokumententyps "**Envelope**"
  - Ein "Envelope" (Umschlag) besteht aus
    - einem (optionalen) "**Header**" und
    - einem (immer vorhandenen) "**Body**"
    - Dies gilt für SOAP 1.1 wie auch für SOAP 1.2!
  - SOAP 1.1 Schema (Auszug) dazu:




```
<xs:complexType name="Envelope">
  <xs:sequence>
    <xs:element ref="tns:Header" minOccurs="0"/>
    <xs:element ref="tns:Body" minOccurs="1"/>
    <xs:any namespace="##other" minOccurs="0"
            maxOccurs="unbounded" processContents="lax"/>
  </xs:sequence>
<xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>
```

Nur SOAP 1.1

SOAP

- SOAP: Header und Body
  - Das **Header**-Element ist neu, etwa im Vergleich zu XML-RPC.
    - Es nimmt Meta-Informationen zum "Body" auf; auch *Routing*-Angaben, "*Credentials*", Verarbeitungshinweise etc. gehören hierhin.
  - Das **Body**-Element enthält je nach Situation
    - das auszutauschende Dokument (EDI-Modus)
    - XML-codierte RPC-Angaben (RPC-Modus), incl. Rückgabedaten
    - Fehlerinformation (das Element "Fault")
  - ... **sind Container**!
    - **"Header" und "Body" sind i.w. Container, die beliebige XML-Elemente anderer Namensräume aufnehmen.**
    - (PRÜFEN) SOAP-interne Anforderungen werden per Konvention ergänzt.
    - SOAP definiert lediglich einige Attribute für „Header“-Elemente.
    - **Elemente** (und ggf. auch Attribute) **im "Body"-Element müssen mit eigenen Namensräumen qualifiziert sein!**

- SOAP 1.1: Kritische Anmerkungen
  - Umgang mit Namensräumen
    - Die Beispiele in den Spezifikationen von SOAP 1.1 (und in davon offenbar abgeleiteten Büchern) enthalten nach Meinung des Dozenten etliche Fehler im Umgang mit Namensräumen.
    - Beispiel (aus Ex. 1 der Spezifikationen):

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle=
              "http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>DIS</symbol> <!-- NS-Fehler: Präfix fehlt -->
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

    - Das rot markierte Element trägt kein Namensraum-Präfix, und es liegt keine Deklaration eines Default-Namensraums vor!

SOAP

- SOAP 1.1: Kritische Anmerkungen (Forts.)

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=
          "http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle=
          "http://schemas.xmlsoap.org/soap/encoding/"/>
  <SOAP-ENV:Header>
    <t:Transaction xmlns:t="some-URI"
      xsi:type="xsd:int" mustUnderstand="1">
      5
    </t:Transaction>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <m:GetLastTradePriceResponse xmlns:m="Some-URI">
      <Price>34.5</Price>
    </m:GetLastTradePriceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP

- SOAP 1.1: Kritische Anmerkungen (Forts.)
  - Vermutete Fehler im o.g. Beispiel:
    - Element "Envelope": Vermutlich nur ein Tippfehler
  - Präfix "xsi" nicht deklariert
    - Vielleicht nahmen die Autoren an, dass einige Standard-Präfixwerte stets vorhanden sind und deshalb nicht deklariert werden müssen. Dem ist aber nicht so.
  - Element "Price" ohne Präfix
    - Vermutlich gingen die Autoren von einem Vererbungskonzept aus, unterstellten also, dass "Price" den Namensraum vom Elternelement "GetLastTradePriceResponse" erbte. Das wäre aber falsch, denn nur für die Präfixwerte selbst gibt es eine Vererbungskonvention (d.h. diese müssen nicht immer wieder deklariert werden).
  - Hinweis: SOAP 1.2 scheint korrigiert worden zu sein.

SOAP-RPC: Demo

Allgemeines Vorgehen aus Entwicklersicht

Demo: Tilgungsplaner

-Normalbetrieb

-Fehlerfall (auf Serverseite)

SOAP-Demo

- Erzeugung eines Web Service auf Basis von SOAP

1) Entwicklung der Anwendung
   - Auf RPC-Eignung achten
   - Ansonsten: Noch keine SOAP-Einflüsse!
   - Demo: Datei **`tilgungplan.rb`**

2) Verpackung der Anwendung als SOAP-Server
   - Assoziation von Objekten, Methoden, Parametern mit öffentlichen Namen (von XML-Elementen)
   - Zuweisung eines Namensraums
   - Assoziation des Dienstes mit URL
   - Entscheidung über Server-Betriebsart: Standalone, CGI, FastCGI, …
   - Starten. Demo: Datei **`tp_server.rb`**

3) Client-Entwicklung
   - Proxy für den Web Service anlegen. Verbindung: URL und Namensraum
   - Methoden & Parameter anmelden. Verbindung: Veröffentlichte Namen
   - Normaler Methodenzugriff via Proxy. Demo: Datei **`tp_client.rb`**

RPC mit *SOAP*

RPC-Regeln

SOAP encoding

Typisierung

Von XML-RPC zu SOAP

- XML-RPC
  - Spezielle XML-Elemente zur Beschreibung und Typisierung von RPC-Parametern
  - Namensraum per Präfix
- Beispiel

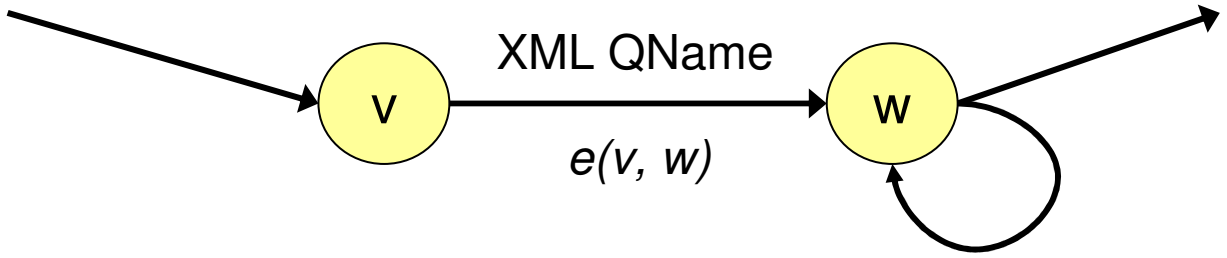
```
<methodCall>
  <methodName>
    beispiel.getCityNameByZIP
  </methodName>
  <params>
    <param>
      <value>
       <i4>65197</i4>
      </value>
    </param>
  </params>
</methodCall>
```

- SOAP im RPC-Modus
  - „Body“ bleibt ein Container
  - Inhalt wird per Konvention als RPC gedeutet → encodingStyle
  - Typisierung mit Attributen
  - XML-Namensräume!
- Beispiel

```
<env:Envelope xmlns:env=... >
 <env:Body>
  <b:getCityNameByZIP
       xmlns:b="http://beispiel"
       env:encodingStyle="..."
   <b:PLZ
        xsi:type="xsd:unsigned">
     65197</b:PLZ>
  </b:getCityNameByZIP>
 </env:Body>
</env:Envelope
```

SOAP

- SOAP RPC
  - Für RPC mit SOAP gelten strengere Regeln für den Aufbau und Inhalt des "Body"-Elements.
  - Einhaltung dieser Regeln ist eine freiwillige Entscheidung - SOAP lässt sich auch anders nutzen!
  - SOAP RPC basiert auf dem (optionalen) SOAP-Datenmodell.
- Das SOAP-Datenmodell
  - Datenstrukturen und Werte der Anwendungsebene werden als gerichtete Graphen mit beschrifteten Kanten verstanden:



  - Ein Knoten hat einen (lexikalischen) Wert und trägt optional eine Typenbezeichnung *xs:QName* aus http://www.w3.org/2001/XMLSchema

SOAP

- Das SOAP-Datenmodell
  - Einfache Werte
    - sind Knoten mit einem lexikalischen Wert
  - Zusammengesetzte Werte
    - sind Knoten mit ausgehenden Kanten:

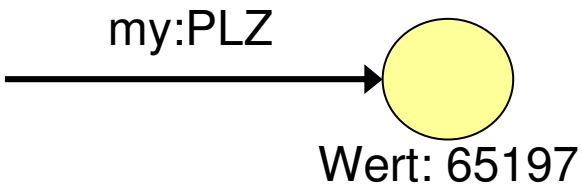
    1. "**structs**":
       - Kanten unterscheiden sich nur aufgrund ihrer Labels
       - Jede Kante muss ein eindeutiges Label tragen
    2. "**arrays**":
       - Kanten unterscheiden sich nur aufgrund ihrer Reihenfolge
       - Kanten dürfen nicht beschriftet sein.

  - Referenzen
    - Knoten können (einfach oder mehrfach) referenziert werden.
    - Dies ist erkennbar an eingehenden Kanten.

SOAP

- SOAP *encoding*
  - präzisiert das SOAP Datenmodell und spezialisiert es auf die Begriffe aus XML Infosets wie *element information item*, *attribute info. item* etc.
  - wird mittels (URI-) Wert des Attributs **`encodingStyle`** spezifiziert
  - kann in Header-Elementen wie auch im Body verwendet werden
  - existiert in zwei Varianten (1.1, 1.2)
  - URI (Attributwert) zur Kennzeichnung von SOAP *encoding* gemäß
    - SOAP 1.1: **`http://schemas.xmlsoap.org/soap/encoding/`**
    - SOAP 1.2: **`http://www.w3.org/2003/05/soap-encoding`**

- SOAP-Encoding, einfache Datenelemente
  - Beispiel: `<my:PLZ>65197</my:PLZ>` entspricht dem Graphen

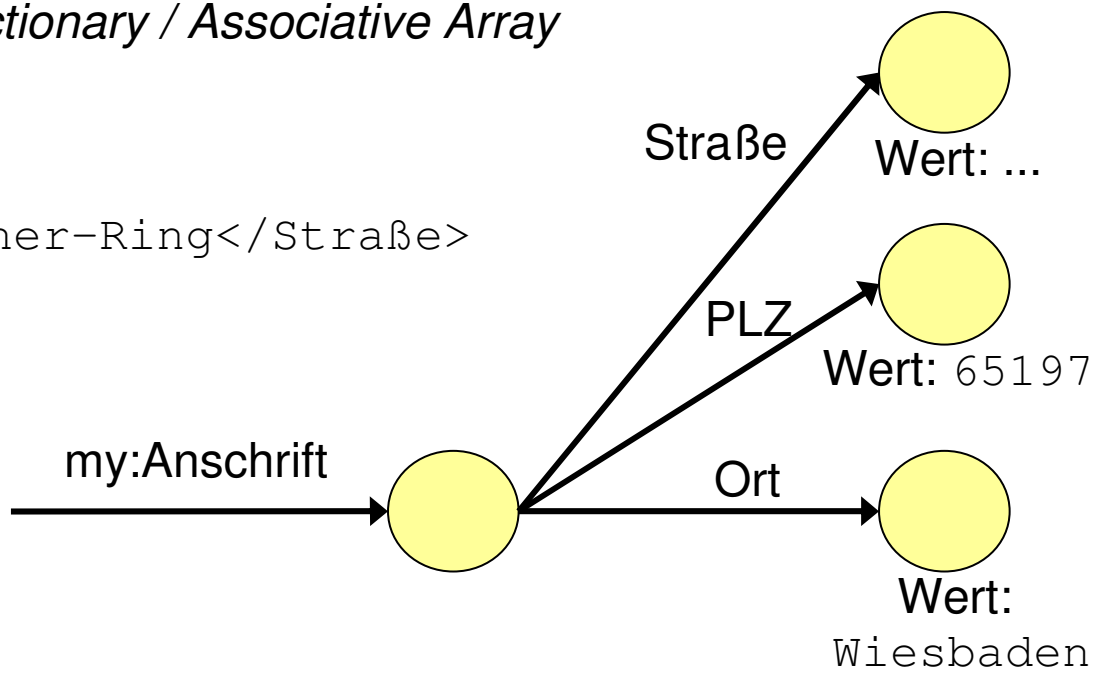


SOAP

- SOAP *encoding*: **struct**
  - Die Komponenten eines **struct** werden einfach auf Unterelemente (*element info items*) des zusammengesetzten Elements abgebildet.
  - Die Reihenfolge darf dabei keine Rolle spielen!
    - DTDs stellen dafür keinen Mechanismus zur Verfügung
    - XML Schema: Wenn, dann "**`all`**" - nicht "**`sequence`**"!
    - Jedes Unter-Element darf nur einmal erscheinen.
    - Direkte Analogie: *Hash / Dictionary / Associative Array*
  - Beispiel:

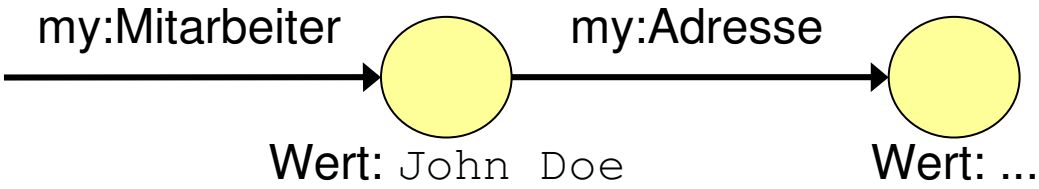
```
<my:Anschrift>
  <Straße>Kurt-Schumacher-Ring</Straße>
  <PLZ>65197</PLZ>
  <Ort>Wiesbaden</Ort>
</my:Anschrift>
```

  - Graph:



SOAP

- SOAP *encoding:* **Referenzen**
  - Innerhalb eines "envelope"-Elements können Werte auch referenziert werden. Redundanzen lassen sich so vermeiden.
    - SOAP 1.1 verwendet dazu den HTML-Mechanismus der *fragment identifier* (bzw. XPointer-Notation!) und die Attribute **`id`** und **`href`**
    - SOAP 1.2 setzt dagegen den XML-Standard ID / IDREF ein und verwendet dazu die Attribute **`id`** und **`ref`**
    - In SOAP 1.2 sind multiple Referenzen auf beliebige Elemente möglich, nicht nur auf solche der obersten Ebene.
  - Beispiel:
    - ```
      <my:Mitarbeiter ref="addr_firma2">
        John Doe
      </my:Mitarbeiter>
      <my:Adresse id="addr_firma2">
       ...
      </my:Adresse>
      ```
    - entspricht dem Graphen:


      ```

SOAP

- Multiple Referenzen, SOAP 1.1-Bsp.

```
<e:Books>
 <e:Book>
  <title>My Life and Work</title>
  <author href="#henryford" />
 </e:Book>

 <e:Book>
  <title>Today and Tomorrow</title>
  <author href="#henryford" />
 </e:Book>
</e:Books>

<author id="henryford">
 <name>Henry Ford</name>
</author>
```

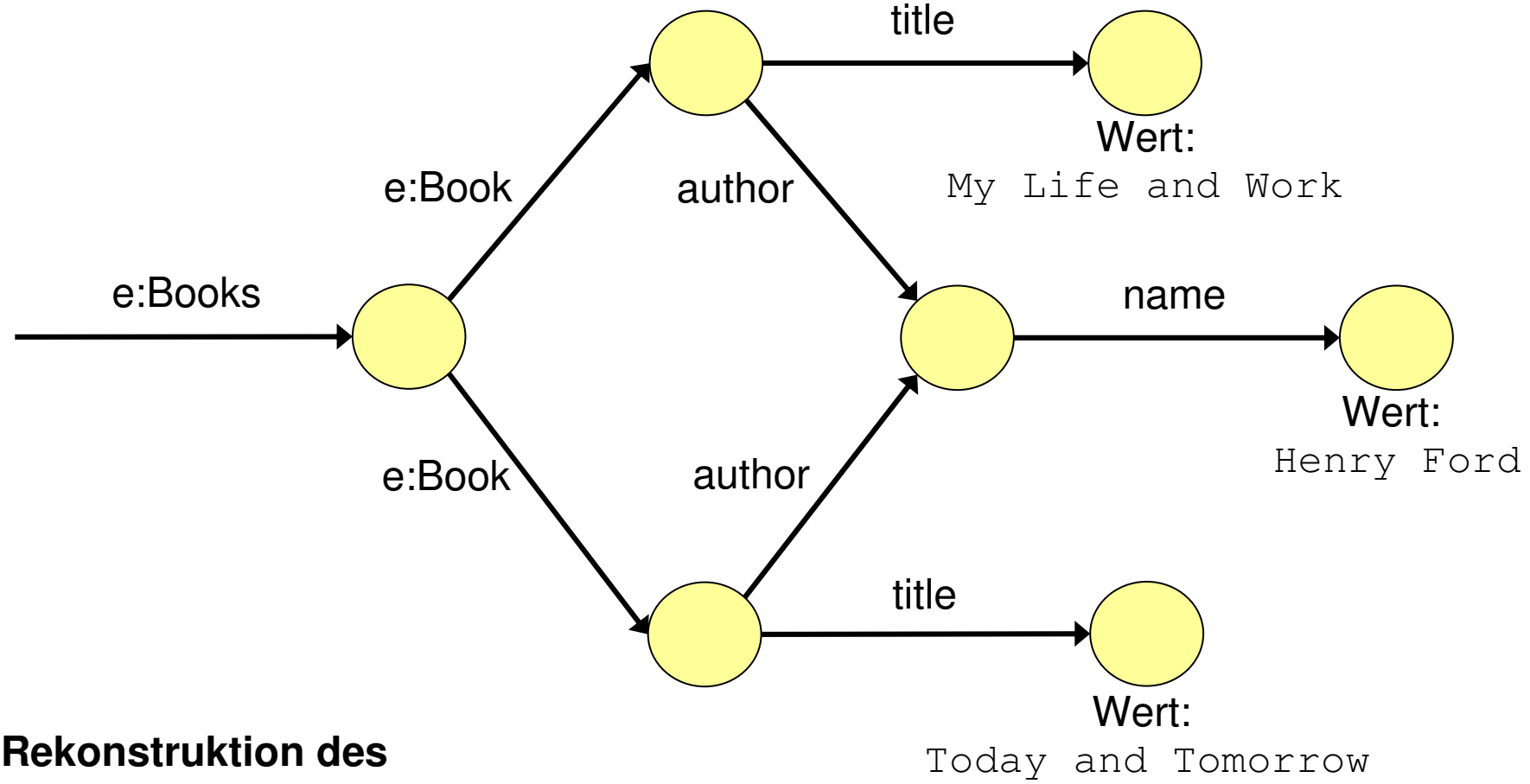
- Multiple Referenzen, SOAP 1.2-Bsp.

```
<e:Books>
 <e:Book>
  <title>My Life and Work</title>
  <author id="henryford" >
    <name>Henry Ford</name>
  </author>
 </e:Book>

 <e:Book>
   <title>Today and Tomorrow</title>
   <author ref="henryford" />
 </e:Book>
</e:Books>
```

SOAP

Referenzen im SOAP-Datenmodell:



Rekonstruktion des Graphen zu e:Books

SOAP

- SOAP *encoding*: **Typisierung**
  - SOAP benutzt i.w. die (einfachen) Datentypen aus XML Schema
  - Man verwendet eine Konvention aus diesem Kontext: "xsi:type".
  - Knoten können ein Attribut "type" aus dem "XMLSchema-instance"-Namensraum tragen. Dessen Wert ist der Datentyp (ein QName).
  - SOAP hat viele XML-Schema Datentypen in seine "encoding"-Namensräume aufgenommen, wie auch Elemente dieses Namens.

  - Beispiele:
    - `<PLZ xsi:type="xsd:int">65197</PLZ>`
    - `<PLZ xsi:type="xsd:unsignedShort">65197</PLZ>`
    - `<PLZ xsi:type="my:unsigned5">65197</PLZ>`
  - Beispiele (SOAP 1.1):
    (`xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"`)
    - `<PLZ xsi:type="enc:string">Mein Freitext</PLZ>`
    - `<PLZ xsi:type="xsd:string">Mein Freitext</PLZ>`

SOAP

- SOAP *encoding*: **Typisierung**
  - Auszüge aus dem SOAP 1.1-Schemadokument (das sich mit dem *targetNamespace*-URI real abrufen lässt):

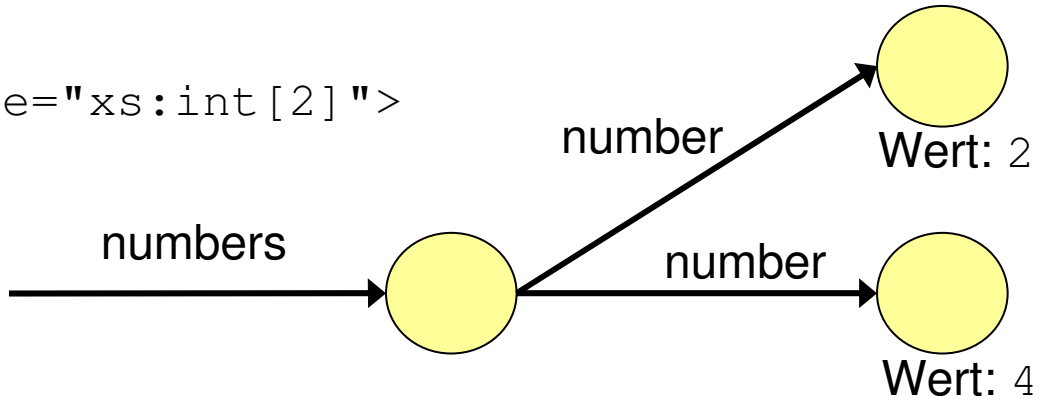
```
<xs:schema targetNamespace="http://schemas.xmlsoap.org/soap/encoding/">

    <xs:simpleType name="base64"> <!-- enc:base64 (Synonym) -->
      <xs:restriction base="xs:base64Binary"/>
    </xs:simpleType>

    <xs:element name="double" type="tns:double"/>
                              <!-- Spezielles, typisiertes Element -->
    <xs:complexType name="double">
      <xs:simpleContent>
        <xs:extension base="xs:double">      <!-- Übernahme -->
          <xs:attributeGroup ref="tns:commonAttributes"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
```

SOAP

- SOAP *encoding*: **array**
  - Array-Elemente werden auf Unterelemente (*element info items*) des zusammengesetzten Elements abgebildet.
  - Nur die Reihenfolge ist bedeutungstragend.
  - **Alle Unterelemente müssen gleich heißen**.
    - Direkte Analogie: Klassisches Array
  - Zwischen SOAP 1.1 und 1.2 gibt es erhebliche Unterschiede:
    - Attribute: **`arrayType`** --> **`itemType`** und **`arraySize`**
    - Nur SOAP 1.1 unterstützt spärliche Arrays und die Übertragung von Arrayteilen.
    - Die Syntax der Dimensionsangaben ist geändert.
  - Beispiel (SOAP 1.1):
    -
```
<numbers enc:arrayType="xs:int[2]">
  <number>2<number>
  <number>4<number>
</numbers>
```



SOAP

- Arrays, SOAP 1.1-Bsp.

```
<numbers enc:arrayType="xs:int[2]">
 <number>3</number>
 <number>4</number>
</numbers>

<!-- 2-dim Array mit Elementen
     unterschiedlichen Typs:   -->
<array-of-objects
   xsi:type="SOAP-ENC:Array"
   SOAP-ENC:arrayType=
   "SOAP-ENC:ur-type[2,2]">
 <!--e xsi:type="xsd:string">
   Ein Text </e -->
 <e xsi:type="xsd:int">123</e>
 <e xsi:type="xsd:double">3.14</e>
 <e xsi:type="xsd:boolean">true</e>
 <e xsi:type="xsd:anyURI">
   http://www.w3.org/some/path</e>
</array-of-objects>
```

- Arrays, SOAP 1.2-Bsp.

```
<numbers enc:itemType="xs:int"
   enc:arraySize="2">
 <number>3</number>
 <number>4</number>
</numbers>

<!-- 2-dim Array mit Elementen
     unterschiedlichen Typs:   -->
<array-of-objects
   xsi:type="enc:Array"
   enc:arraySize="2 2">
 <!-- e xsi:type="xsd:string">
   Ein Text </e -->
 <e xsi:type="xsd:int">123</e>
 <e xsi:type="xsd:double">3.14</e>
 <e xsi:type="xsd:boolean">true</e>
 <e xsi:type="xsd:anyURI">
   http://www.w3.org/some/path</e>
</array-of-objects>
```

SOAP

- SOAP *encoding (1.1)*: Mehr zu **`arrayType`**
  - **`enc:arrayType="xs:int[5]"`**
    - Ein Array mit 5 Elementen
  - **`enc:arrayType="xs:int[]"`**
    - Ein Array unspezifizierter Länge
  - **`enc:arrayType="xs:int[2,3]"`**
    - Ein 2-dim. Array, bestehend aus 2 * 3 Elementen
  - **`enc:arrayType="xs:int[,3]"`**
    - Ein 2-dim. Array, bestehend aus x * 3 Elementen (x unspezifiziert)
  - **`enc:arrayType="xs:int[3][]"`**
    - Ein Array aus einer beliebigen Zahl 1-dim. Arrays der Länge 3.

SOAP

- SOAP *encoding (1.2)*: Mehr zu **`arraySize`**
  - **`enc:arraySize="5"`**
    - Ein Array mit 5 Elementen
  - **`enc:arraySize="*"`**
    - Ein Array unspezifizierter Länge (default-Wert!)
  - **`enc:arraySize="2 3"`**
    - Ein 2-dim. Array, bestehend aus 2 "Zeilen" à 3 "Spalten"
  - **`enc:arraySize="* 3"`**
    - Ein Array aus einer beliebigen Zahl "Zeilen" mit je drei "Spalten".
    - Arrays von Arrays werden NICHT mit **`arraySize`** spezifiziert!
    - Nur die erste Dimensionsangabe darf unspezifiziert bleiben ("*") !

SOAP

- SOAP *encoding* (nur 1.1): **`offset`**
  - Mittels "**`offset`**" (Basis 0) lassen sich Arrayteile übertragen.
  - Beispiel: Zahlen von 1 bis 5, übertragen wird nur 3 bis 5:

```
<numbers xsi:type="enc:Array" enc:arrayType="xs:int[5]" enc:offset="[2]">
  <number>3</number>
  <number>4</number>
  <number>5</number>
</numbers>
```

- SOAP *encoding* (nur 1.1): **`position`**
  - Attribut "**`position`**" lokalisiert ein Arrayelement explizit, z.B. zur Übertragung spärlich besetzter Matrizen
  - Beispiel: Elemente 3. Zeile / 6. Spalte, 6. Zeile / 3. Spalte einer 10 x 10-Matrix

```
<numbers xsi:type="enc:Array" enc:arrayType="xs:int[10,10]">
  <number enc:position="[2,5]">3</number>
  <number enc:position="[5,2]">4</number>
</numbers>
```

SOAP

- SOAP *encoding*: Das Attribut **`nodeType`**
  - Zur leichteren Unterscheidung von Elementen, die Strukturen, Arrays oder einfache Datenelemente verkörpern, darf man das Attribut **`nodeType`** vergeben.
  - Die Werte von **`nodeType`** sind: **`array`**, **`struct`**, oder **`simple`**
  - Beispiel:
    - ```
      <numbers
            enc:nodeType="array"
            enc:arrayType="xs:int[2]">
        <number enc:nodeType="simple">2<number>
        <number enc:nodeType="simple">4<number>
      </numbers>
      ```
      ```

SOAP

- SOAP *encoding*: Komplexe Datentypen
  - Array-Elemente dürfen selbst Arrays oder Structs sein,
  - Struct-Elemente dürfen selbst Arrays oder Structs sein.
  - Dadurch lassen sich beliebig komplexe Strukturen erzeugen.
  - Durch Verwendung von Multi-Referenzen entstehen zusätzliche Möglichkeiten!
  - Gerade bei komplexen Datentypen: **`nodeType`** verwenden!

- **Achtung**:
  - SOAP 1.2 behandelt Arrays von Arrays nicht mehr explizit (mittels `arrayType`) - die Verschachtelung genügt.

SOAP

- SOAP *encoding*: Abgeleitete einfache Datentypen
  - Sie sind nicht auf die Verwendung der Standard-Datentypen aus XML Schema beschränkt. Eigene Datentypen sind ggf. über einen eigenen Namensraum / eigenes Präfix von anderen zu unterscheiden.

- SOAP *encoding*: Fehlende Daten, der **`nil`**-Typ
  - Kanten, die auf einen Knoten ohne Inhalt verweisen
    - entsprechen leeren XML-Elementen und
    - können entweder ausgelassen werden oder
    - werden als leere Elemente codiert, die den Datentyp **`xsi:nil`** tragen.
  - Beispiel

```
<concat3Method>
  <param1 xsi:type="xsd:string">ein String</param1>
  <param2 xsi:type="xsd:string">noch ein String</param2>
  <param3 xsi:type="xsi:nil"/>
</concat3Method>
```

SOAP

- SOAP RPC
  - *Binding*:
    - Da SOAP 1.2 mit verschiedenen Transportprotokollen kooperieren soll, muss geprüft werden, ob das gewählte (gut) geeignet ist zur Implementierung des *request/response*-Musters.
    - Traditionell - aber nicht ausschließlich - wird hier HTTP eingesetzt.
  - *Request* im RPC-Stil : Aufbauregeln für den "Body"
    - Enthält genau ein "struct"-Element, Elementname = Methodenname
    - Dessen Unterelemente repräsentieren die Übergabeparameter: ein struct
    - Reine Rückgabewerte erhält man via "Response", nicht hier.
    - Es gibt Regeln zur Übersetzung "schwieriger" Anwendungsnamen in zulässige XML-Namen. Beispiele (hier nicht näher erläutert):

```
Hello world -> Hello_x0020_world
Hello_xorld -> Hello_x005F_xorld
Helloworld_ -> Helloworld_
x -> x, xml -> _x0078_ml, -xml -> _x002D_xml, x-ml -> x-ml Ælfred -
> Ælfred, άγνωστος -> άγνωστος
```

SOAP

- SOAP RPC
  - RPC Response: Aufbau des "Body"
    - Enthält genau ein "struct"-Element
      - Der Elementname ist nicht signifikant, üblich ist Methodenname+"Response"
    - Dessen Unterelemente repräsentieren die Rückgabeparameter
      - Einer dieser Rückgabeparameter stellt den eigentlichen **Rückgabewert** dar. Von SOAP 1.2 wird er gesondert identifiziert (s.u.). Die anderen Parameter sind i.d.R. In/Out-Werte, also solche, die schon im *request* enthalten waren (→ call by ref.).
    - Auch hier gelten die Regeln zur Übersetzung von Anwendungsnamen
  - SOAP 1.2: Identifizierung des Elementnamens für den Rückgabewert
    - Elementname = Inhalt eines Elements "`result`" aus dem Namensraum "`http://www.w3.org/2003/05/soap-rpc`"
    - Typ dieses Inhalts: QName
  - **ACHTUNG**:
    - Element "result" MUSS bei non-void Rückgabe vorhanden sein,
    - Element "result" DARF bei void-Rückgabe NICHT vorhanden sein!

SOAP

- SOAP RPC: *request/response*-Beispiel

Request:

```
<?xml version="1.0">
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <env:Body>
  <sb:echoInteger xmlns:sb="http://soapinterop.org/"
      env:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
   <inputInteger xsi:type="xsd:int">123</inputInteger>
  </sb:echoInteger>
 </env:Body>
</env:Envelope>
```

Response (nur "Body"):

```
<env:Body>
  <sb:echoIntegerResponse xmlns:sb="http://soapinterop.org/"
      xmlns:rpc="http://www.w3.org/2003/05/soap-rpc"
      env:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
   <rpc:result>return</rpc:result>
   <return xsi:type="xsd:int">123</return>
  </sb:echoIntegerResponse>
</env:Body>
```

SOAP

- SOAP Faults
  - Hier nicht näher besprochen.
  - Beispiele finden Sie etwa im Tutorial der Spezifikationen (Teil 0).

SOAP

- Zwischenstand:
  - Mehr zu SOAP-Datentypen / SOAP *encoding*
    - *OK*
  - SOAP 1.1 vs. 1.2
    - OK
  - Umgang mit Faults, der SOAP-Header
    - Besprechung der Beispiele aus dem Tutorial!
    - Weitere Quellen
      - Beispiele in der SOAP 1.1-Note
      - Testfälle in der SOAP 1.2 Test Collection

SOAP

- SOAP Tutorial
  - Ex 1
    - *Envelope, Header vs. Body, Versioning, XML doc. vs. infoset*
    - *Namespaces & NS-prefixes, conversational style, document/EDI mode, binding options*
    - *Header: message path, role, mustUnderstand*
  - Ex 2, Ex 3
    - *Conversational style: Refinements. header/reference!*
  - Ex 4:
    - *RPC demo using reserve-and-charge process, target URI, no IDL*
    - *structs, encodingStyle*
    - *transaction-Header, implizit(!) an role=ultimate recipient*
    - Bsp. ist nur Teil einer Transaktion! Neu: code "FT35ZBQ"

SOAP

- SOAP Tutorial
  - Ex 5 (a, b):
    - *RPC responses*: Konvention Methodenname+"Response"
    - Konvention: rpc:result, identifiziert den Return-Wert "status"
    - Neu in 5b: "status" *(confirmed, pending)*
  - Ex 6
    - *Fault: Code, Reason, Node, Detail, Role, Subcode, Text, Value*
  - ...
  - Ex 16
    - *Using intermediaries*
  - Ex 17
    - *Using other encodingStyle values (here: RDF)*