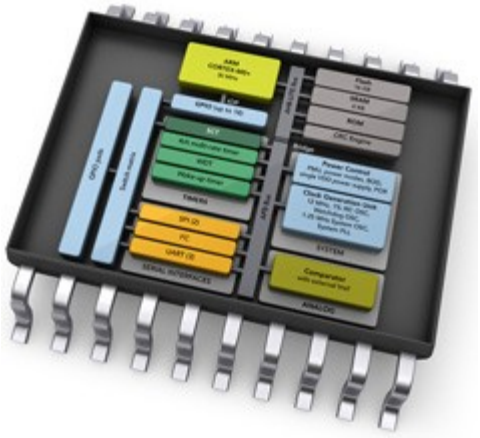
FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

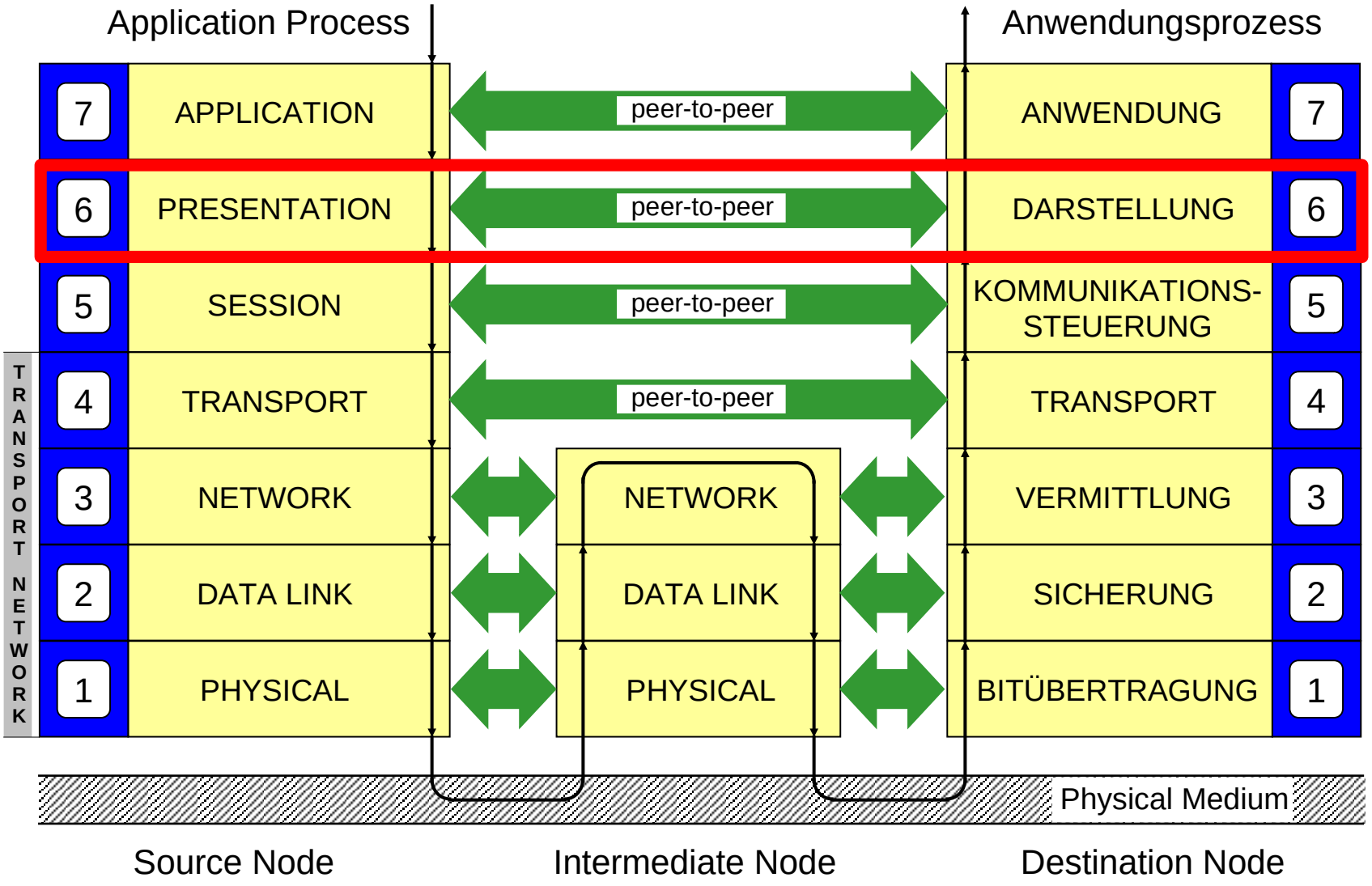
Kommunikationssysteme

(Modulcode 941306)

Prof. Dr. Andreas Terstegge



Das ISO-OSI-Referenzmodell für offene Systeme



Probleme bei der Datenübertragung

Heterogenität im verteilten System:

- Heterogenität der Plattform (Betriebssystem, Hardware)
- Heterogenität der Programmiersprache/APIs

Heterogene Hardware-Architekturen

- Unterschiedliche Reihenfolge der Speicherung von Bytes: Little Endian / Big Endian
- Unterschiedliche Zeichencodierung:
  - ASCII auf PCs / EBCDIC auf IBM Mainframes

Problem: Uneinheitliche interne Darstellung

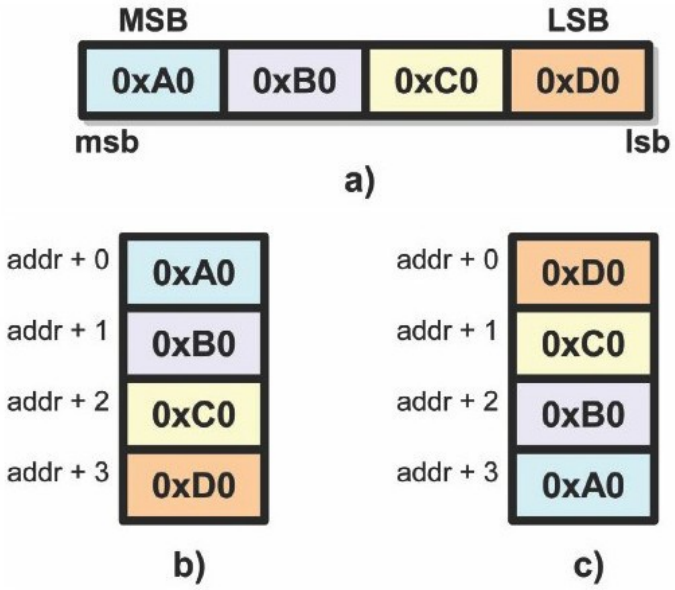
Z.B. Unterschiedliche Darstellungen von Integer-Werten im Arbeitsspeicher: Little-Endian / Big-Endian

a) Ausgangswert

b) Big-Endian Speicherung

c) Little-Endian Speicherung

→ Probleme bei Interpretation der Daten nach Versenden ja nach Rechnerarchitektur

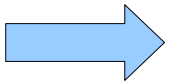


Zeichenketten sind von dieser Problematik nicht betroffen!

Problem: Übertragung von Datenstrukturen


```
struct Student {
  String Studiengang;
  int Alter;
  bool imAuslandssemester;
}
```

```
Student s;
s.Studiengang = „AMI“;
s.Alter = 20;
…
send( s );
```



...01011001101...

```
Student s = receive();

if (Alter > 25) …
```

- Häufig besteht der Bedarf, nicht nur einzelne Werte/Strings zu übertragen, sondern ganze Datenstrukturen
- Daher müssen 2 Probleme gelöst werden:
  - Übertragung der grundsätzlichen Struktur der Daten (**abstrakte Syntax**), *Serialisierung - Deserialisierung*
  - Übertragung des konkreten Datenstroms mit einer vereinbarten Kodierung (**Transfersyntax**), *Marshalling - Unmarshalling*

Datendarstellung sollte eigenständig sein

Idee:

- Definiere eine **Menge abstrakter Datentypen** und eine **Kodierung** (ein genaues Bit-Format) für jeden dieser Typen
- Stelle **Werkzeuge** zur Verfügung, die die abstrakten **Datentypen in** Datentypen der verwendeten **Programmiersprache** übersetzen
- Stelle **Werkzeuge** zur Verfügung, die die Datentypen der verwendeten **Programmiersprache in** die abstrakten **Datentypen** und damit in das kodierte Format übersetzen
- *Senden (**Marshalling**)*: wenn ein bestimmter Datentyp übertragen werden soll, rufe die Kodierfunktion auf und übertrage das Ergebnis
- *Empfangen (**Un-Marshalling**)*: dekodiere den Bit-String und erzeuge eine neue lokale Repräsentation des empfangenen Typs

Beispiele verschiedener Datendarstellungs-Standards

ISO: - **ASN.1** (**A**bstract **S**yntax **N**otation)

Sun ONC (Open Network Computing)-RPC:

- **XDR** (eXternal Data Representation)

OSF (*O*pen *S*ystem *F*oundation)-RPC:

- **IDL** (Interface Definition Language)

Corba: - **IDL** und **CDR** (*C*ommon *D*ata *R*epresentation):

- **CDR** bildet IDL-Datentypen in Bytefolgen ab.

W3C: - **XML/SOAP**

- Darstellung aller Datentypen als (Maschinen-)lesbarer Text. Zu klären: Zeichenkodierung

Java: - **Objektserialisierung**, d.h. Abflachung eines (oder mehrerer) Objektes zu einem seriellen Format inkl. Informationen über die Klassen. Deserialisierung ist die Wiederherstellung eines Objektes ohne Vorwissen über die Typen der Objekte

JavaScript - **JSON** (JavaScript Object Notation)

ASN1: Definition

ASN.1 (abstrakte Syntax-Notation eins) ist eine von der ISO genormte Beschreibungssprache zu darstellungsunabhängigen Spezifikation von Datentypen und Werten

- findet z.B. zur Definition von Managementobjekten bei SNMP Verwendung

- **Elementare Datentypen:**
  - **`Boolean, Integer, Bitstring, Octetstring, IA5String, …`**

- **Strukturierte Datentypen:**
  - **`Sequence:`** Geordnete Liste von Datentypen (struct in C)
  - **`Set:`** Ungeordnete Menge von Datentypen
  - **`Sequence OF:`** Geordnete Liste von Elementen des gleichen Datentyps (Array in C)
  - **`Set OF`**: Ungeordnete Menge des gleichen Datentyps
  - **`Choice:`** Ungeordnete Menge von Datentypen, aus der einige Datentypen ausgewählt werden können (Union in C)

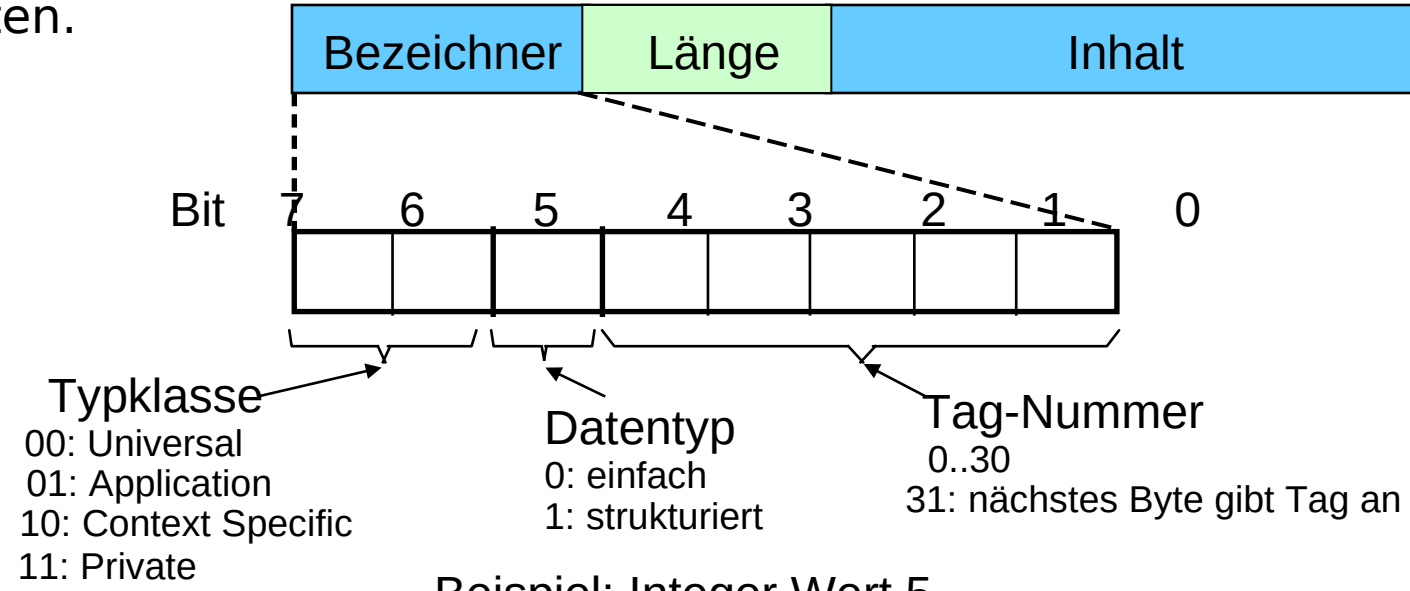
Beispiel:

```
Mitarbeiter ::= Set { Name IA5String,
                      Alter Integer,
                      Personalnr Integer }
```

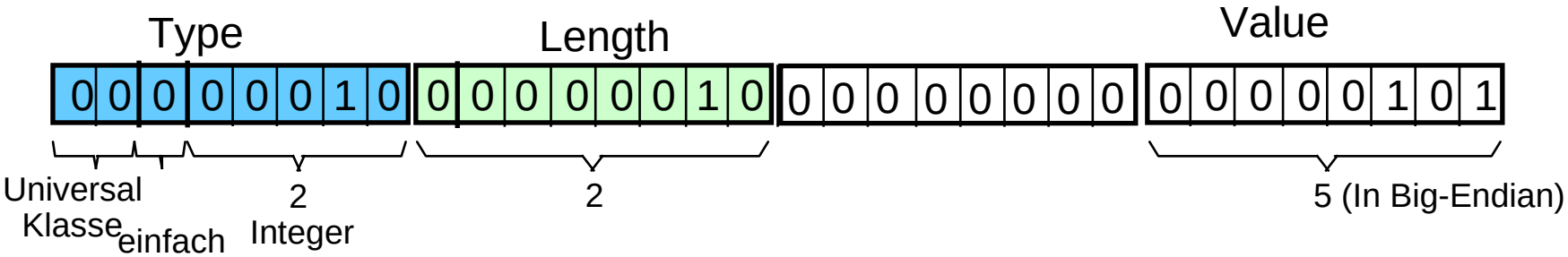
ASN1: Übertragungssyntax

http://en.wikipedia.org/wiki/Distinguished_Encoding_Rules#BER_encoding

Zusätzlich zur Bereitstellung einer Datenbeschreibungssprache bietet ASN.1 auch sogenannte **Basic Encoding Rules**, also Regeln, die spezifizieren, wie ASN.1-Objekte über das Netzwerk versendet werden sollten.



Beispiel: Integer Wert 5



Objektserialisierung in Java

- Die Klasse **ObjectInputStream** bietet die Möglichkeit, mit der Methode `readObject()` serialisierte Objekte aus dem darunterliegenden Stream zu lesen

- Serialisierung ist der synonym verwendete Begriff für Objekt-Streams, d.h. das Lesen und Schreiben von Objekten. Serialisierung ist nicht auf singuläre Objekte beschränkt, da diese ja wieder andere Objekte referenzieren können.

    - `writeObject()` wandelt ein Objekt in einen Byte-Stream um
    - `readObject()` wandelt einen Byte-Stream in ein Objekt um

- Im Unterschied zum allgemeinen I/O- bzw. Streaming-Konzept beruht die Objekt-Kommunikation auf Interfaces. Somit ist auch eine komplett eigencodierte Serialisierung möglich

XML: eXtensible Markup Language

HTML wurde zur Darstellung von Information für Menschen entwickelt, und nicht für die Informationsverarbeitung in Programmen

- Die Auszeichnungselemente sind **fest vorgegeben** und sind daher nicht zur Beschreibung beliebiger Daten geeignet
- Problematische **Vermengung** von Auszeichnung und Präsentationssemantik

Dies motivierte die Entwicklung einer offenen, textbasierten Auszeichnungssprache: e**X**tensible **M**arkup **L**anguage

XML: eXtensible Markup Language

- XML ist eine Metasprache zur Strukturierung von Dokumenten und Daten
- Basis von XML sind beliebige Auszeichnungselemente, die geringen syntaktischen Regeln zu folgen haben
  - Wohlgeformte XML-Dokumente
- XML macht erst wirklich Sinn, wenn es in einer Anwendung konkretisiert wird. XML-Anwendungen ergeben sich aus
  - **Namensräumen**

    Festlegen der semantisch logischen Bedeutung bei Homonymen in den Auszeichnungselementen
  - **Stylesheets**

    Definition des Erscheinungsbildes von Auszeichnungselementen
  - **Schemata**

    Beschreiben die Dokumentenstruktur eines bestimmten Typs

Beispiel: XSD

XML-Dokument: Beispiel

```
<?xml version="1.0" encoding="UTF-8"?>
<rechnung kundennummer="k333063143">
    <monatspreis>0,00</monatspreis>
    <einzelverbindungsnachweis>
    <verbindung>
    <datum>26.2.</datum>
    <zeit>19:47</zeit>
    <nummer>200xxxx</nummer>
    <einzelpreis waehrung="Euro">0,66</einzelpreis>
    </verbindung>
    <verbindung>
    <datum>27.2.</datum>
    <zeit>19:06</zeit>
    <nummer>200xxxx</nummer>
    <einzelpreis waehrung="Euro">0.46</einzelpreis>
    </verbindung>
    <verbindungskosten_gesamt
    waehrung="Euro">2.19</verbindungskosten_gesamt>
    </einzelverbindungsnachweis>
</rechnung>
```

XML-Dokument: Eigenschaften

- XML-Dokumente enthalten **Daten** und **Strukturinformation** über die Daten in einem Dokument (selbstbeschreibend)
- XML-Dokumente haben **Strukturvorgaben** (Wohlgeformtheit)
- Informationen im XML-Dokument haben einen **Datentyp** (typisiert)

Elemente eines XML-Dokuments

- XML-Dokumente: Zeichendaten und Auszeichnungen
  - XML- und Dokumenttyp-Deklarationen

    ```
    <?xml version="1.0" encoding="UTF-8"?>
    ```
  - Elemente mit möglichem Inhalt

    ```
    <name>Volker Sander</name>
    ```
  - Attributen in Elementen

    ```
    <name attribute="Wert">
    ```
  - Entity-Referenzen (`<` statt <)
  - Kommentare

    ```
    <!– Das ist ein Kommentar-->
    ```
  - Processing Instructions – werden an die aufrufende Instanz weitergeleitet

    ```
    <?name pidata?>
    ```

Elemente eines XML-Dokuments

- Jedes Dokument entspricht einer Baumstruktur (DOM – Document Object Model)
- Die Baumknoten werden Elemente genannt
- Es kann nur ein Element an der Wurzel geben
  - Eine Applikation weiß sofort wann das Dokument zu ende ist
- Syntaxregeln müssen strikt eingehalten werden
  - Jedes Starttag muss ein Endtag haben

    ```
    <BOOK>...</BOOK>
    <BOOK />
    ```

  - Verschachtelung nur mit vollständigen Tags möglich:

    ```
    <BOOK> <LINE> This is OK </LINE> </BOOK>
    <LINE> <BOOK> This is </LINE> definitely NOT </BOOK> OK
    ```

  - Attributwerte müssen mit ‘ oder “ abgegrenzt werden

Wann ist ein XML-Dokument „korrekt“?

XML: Korrektheitsbegriffe

Wohlgeformtheit

Dokument entspricht den syntaktischen Regeln

- Genau ein Dokument-Element
- Jedes öffnende Tag hat ein schließendes Tag.
- Die Verschachtelung ist balanciert.

Gegenbeispiel:

<A><B><C> ... </A> ... </C>

Validität

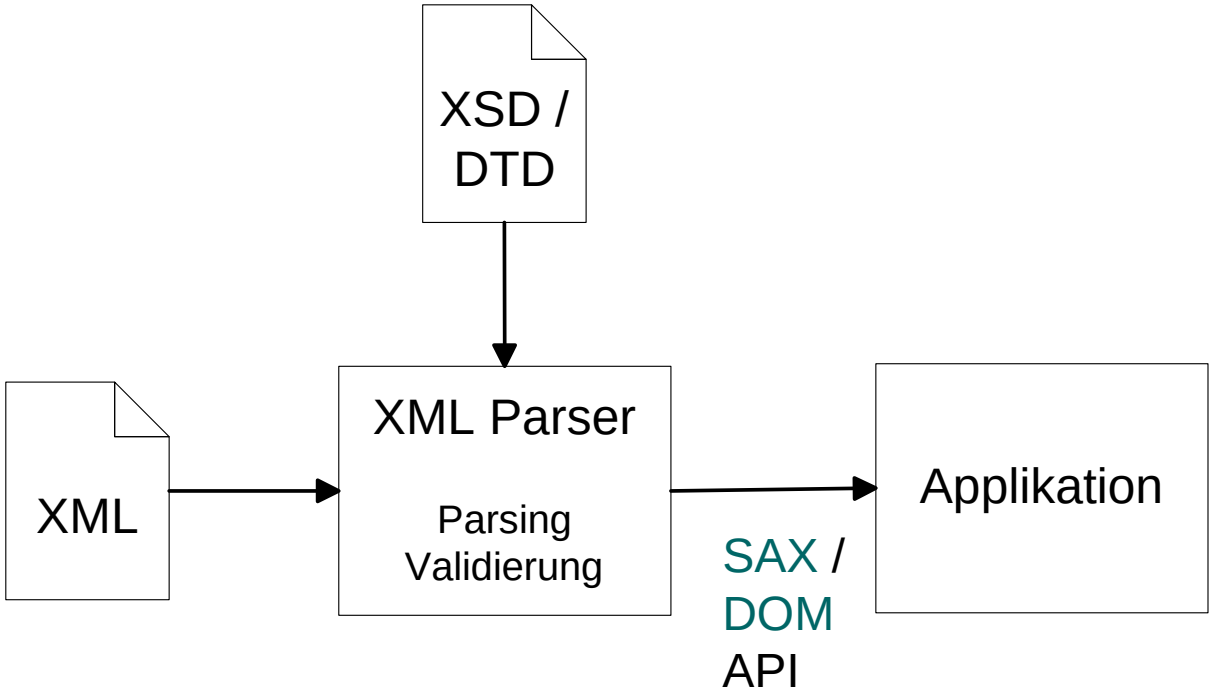
Dokument ist

- wohlgeformt
- konform zu einer fest vorgegebenen Dokumentenstruktur ist

Document Type Description (DTD)

XML Schema Definition (XSD)

XML Parsing: Übersicht



XML Namensräume: Namespaces

Zur Erklärung eines XML-Schemas müssen wir erst ein anderes Problem diskutieren:

- Zum Test eines Dokumentes gegen eine Strukturvorgabe müssen Elementbezeichnungen verarbeitet werden.
- Gleiche Elementbezeichnungen können jedoch unterschiedliche Bedeutung haben:

```
<Schloss>Neuschwanstein</Schloss>
<Schloss>Sicherheitsschloss Typ X</Schloss>
```

- Ähnlich wie Variablen in Programmiersprachen haben Elementbezeichnungen einen Namensraum
- Namespaces dienen der Trennung der Bedeutung der Bezeichner und ermöglichen so, Elementbezeichnungen und Inhalte eindeutig zu referenzieren

XML Namensräume: Namensraumdeklaration

- Namensräume werden weltweit eindeutig durch eine URI (Uniform Resource Identifier) identifiziert
- Diese wird über einen Prefix zugänglich gemacht
- Namensraum-Präfix muss ein XML-Name sein
  - darf keine Sonderzeichen usw. enthalten
  - URI´s beinhalten aber oftmals Sonderzeichen (http://test.com/namespace)
- Namensraumdeklaration ordnet Präfix und URI einander zu:

```
<person xmlns:job="http://www.berufe-online.de/berufe">
        <vorname>Carl Friedrich</vorname>
        <nachname>Gauß</nachname>
        <job:berufsbezeichnung>Mathematiker</job:berufsbezeichnung>
</person>
```

XML Schema

- Ein XML-Schema ist eine XML-Anwendung, mit der die Struktur einer Klasse von XML-Dokumenten beschrieben werden kann
- Der Zweck eines XML-Schemas entspricht dem seines Vorgängers, dem DTD (welches nicht XML war)
- Ein XML Schema enthält in XML notierte Regeln, die erlaubte Elementbezeichner, Reihenfolgen, Inhalte und Attribute mit Wertebereichen aufzählen
- Hierzu existieren vielfältige vordefinierte Datentypen, die Möglichkeit zur Definition eigener Datentypen, und die Möglichkeit zur Darstellung komplexer Integritätsbedingungen

XML Schema

- Um ein Dokument validieren zu können muss es das seine Struktur beschreibende Schema referenzieren (hierzu ist es nicht verpflichtet):

```
<?xml version=“1.0” ?>
   <artikel xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="artikel_simple.xsd“>
```

- Das Schema selber ist immer einem Schema-Namespace zugeordnet

```
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema>
```

  - Somit können in einem Dokument auch Elemente verschiedener Schemata verwendet werden!

XML Schema: Beispiel

Beispiel: `<datum jahr="1979"/>`

Namensraumdeklaration für XML-Schema

```
<xsd:schema xmlns:xsd="http://www.w3c.org/2001/XMLSchema">

<xsd:element name = "datum" type= "datumsTyp"/>

<xsd:complexType name= "datumsTyp">
   <xsd:attribute name = "jahr">
      <xsd:simpleType>
         <xsd:restriction base = "xsd:integer">
            <xsd:maxInclusive value = "2500">
            <xsd:minInclusive value = "0">
         </xsd:restriction>
      </xsd:simpleType>
   </xsd:attribute>
</xsd:complexType>
```

Element-Deklaration

Typdefinitionen

Attribut-Deklaration

XML Schema: Elementdeklarationen

Neben der Benennung eines Tags für ein Element kann auch der Datentyp angegeben werden (per Referenz oder direkter Auflistung)

- Primitive Datentypen
  Schema: `<element name="float-element" type="float" />`

  Dokument: `<float-element>123.456 </float-element>`

- Deklaration von Datentypen (Elementen)
  ```
  <complexType name="studenttype"> ... </complexType>
  ```

- Erweiterung existierender Datentypen mittels Angabe eines Musters (Mit Großbuchstaben beginnend, dann ein oder mehrere Zahlen)
  ```
  <simpleType name="studentid" base="string">
    <pattern value="[A-Z][0-9]+" />
  </simpleType>
  ```

XML Schema: Einfache Datentypen

- Dienen zur Definition der Wertebereiche von Elementen und Attributen

Beispiele für Elementdeklarationen mit einfachen Datentypen:

```
<xsd:element name="name" type="xsd:string"/>
<xsd:element name="schuhgroesse"
                          type="xsd:positiveInteger"/>
<xsd:element name="geburtsdatum" type="xsd:date"/>
```

Verwendung des Schemata in einer XML-Datei:

```
<name>Carl Friedrich Gauß</name>
<schuhgroesse>42<schuhgroesse/>
<geburtsdatum>1777-04-30<geburtsdatum/>
```

XML Schema: Komplexe Typen

- Neben diesen eingebauten Datentypen bietet XML-Schema die Möglichkeit, eigene Datentypen abzuleiten
- Komplexe Typen benötigt man immer dann, wenn man Elemente deklarieren möchte, die Attribute besitzen oder andere Elemente beinhalten können, denn folgendes ist nicht möglich:

```
<xs:element name="Kunde">
   <xs:element name="Vorname" type="xs:string">
   <xs:element name="Zweitname" type="xs:string">
   <xs:element name="Nachname" type="xs:string">
</xs:element>
```

XML Schema: Komplexe Typen

- Erfolgt die Definition eines komplexen Typs für ein Element innerhalb des <xs:element>-Elements, so spricht man von einem **anonymen Typen**

```
<xs:element name="Kunde">
  <xs:complexType>
     <xs:sequence>
        <xs:element name="Vorname" type="xs:string"/>
        <xs:element name="Zweitname" type="xs:string"/>
        <xs:element name="Nachname" type="xs:string"/>
     </xs:sequence>
  </xs:complexType>
</xs:element>
```

XML Schema: Komplexe Typen

- Sollen mehrere Elemente die gleichen Kindelemente und Attribute tragen, so sollten **benannte komplexe Typen** benutzt werden
- Diese werden über das type-Attribut in der Element-Deklaration zugewiesen

```
<xs:complexType name="KundenTyp">
  <xs:sequence>
       <xs:element name="Vorname" type="xs:string"/>
        <xs:element name="Zweitname" type="xs:string"/>
       <xs:element name="Nachname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="Kunde" type="KundenTyp">
```

XML Schema:
Werkzeuge zur Bildung komplexer Typen

- Kompositoren können mehrere Element-Deklaration zu einer unbenannten Modellgruppe
- Es gibt drei Kompositoren, die kombiniert werden können:

XML Schema: sequence

Sequenzen:

Die Inhalte einer Sequenz müssen in der Reihenfolge erscheinen, in der sie angegeben werden:

```
<xsd:sequence>
        [Inhalt 1]
        ...
        [Inhalt n]
</xsd:sequence>
```

XML Schema: all

Die **all**-Gruppe:

- Alle Elemente, die in der **all**-Gruppe deklariert sind, dürfen in beliebiger Reihenfolge instanziiert werden, aber höchstens einmal.
- Außerhalb der **all**-Gruppe darf es keine weiteren Element-Deklarationen geben.

```
<xsd:all>
    [Elementdeklarationen]
</xsd:all>
```

XML Schema: choice

XML – Schema (choice)

Auswahllisten:

Von den aufgeführten Inhalten muss ein frei wählbarer instanziiert werden:

```
<xsd:choice>
    [Inhalt a]
    ...
    [Inhalt x]
</xsd:choice>
```

Ergänzungen zu den Kompositoren

Auswahllisten und Sequenzen können beliebig geschachtelt werden.

Neben der Anordnung kann auch die Häufigkeit festgelegt werden, mit der Elemente erscheinen dürfen.

minOccurs gibt an, wie oft ein Element mindestens erscheinen muss (default 1; Angabe optional).

maxOccurs gibt an, wie oft ein Element höchstens auftreten darf (default 1; ∞ = unbounded;Angabe optional).

default gibt den Wert an, der dem Element zugeordnet wird, wenn es vorhanden, aber leer ist.

```
<element name="vorstandsmitglied" minOccurs=4 maxOccurs=6>
```

Abgeleitete Typen

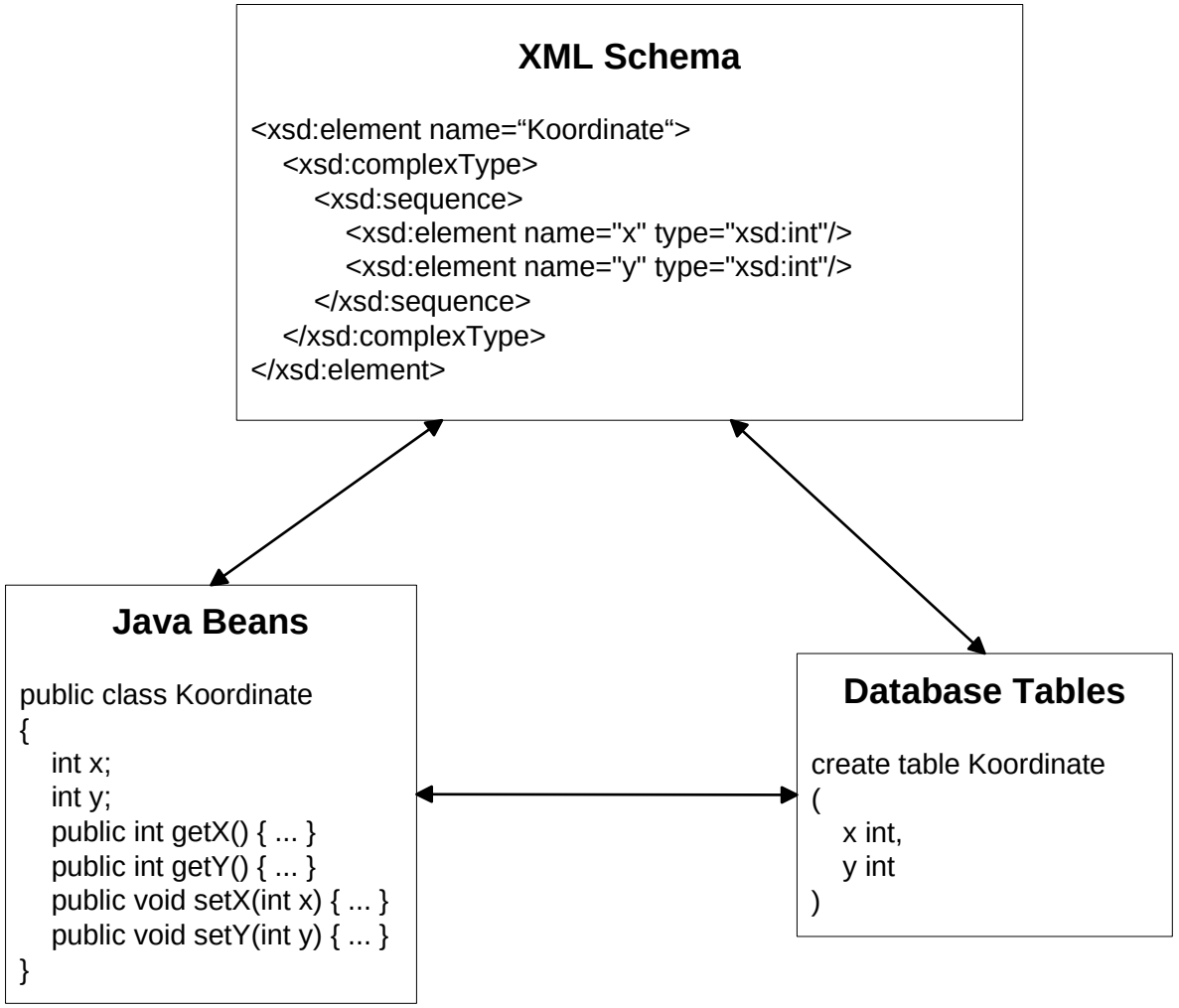
- Ein abgeleiteter Datentyp wird auf der Basis eines anderen Datentyps durch Einschränkung oder Erweiterung spezifiziert.
- Auch von abgeleiteten Datentypen können weitere Datentypen abgeleitet werden.
- Datentypen können in verschiedener Weise abgeleitet werden:
  - durch Einschränkung des Wertebereichs (**restriction**)
  - über eine durch Whitespaces notierte Liste verschiedener Werte (**list**)
  - über eine Vereinigung mehrerer Typen (**union**)
  - durch Erweiterung (nur komplexe Typen) (**extension**)

Objekt Mappings

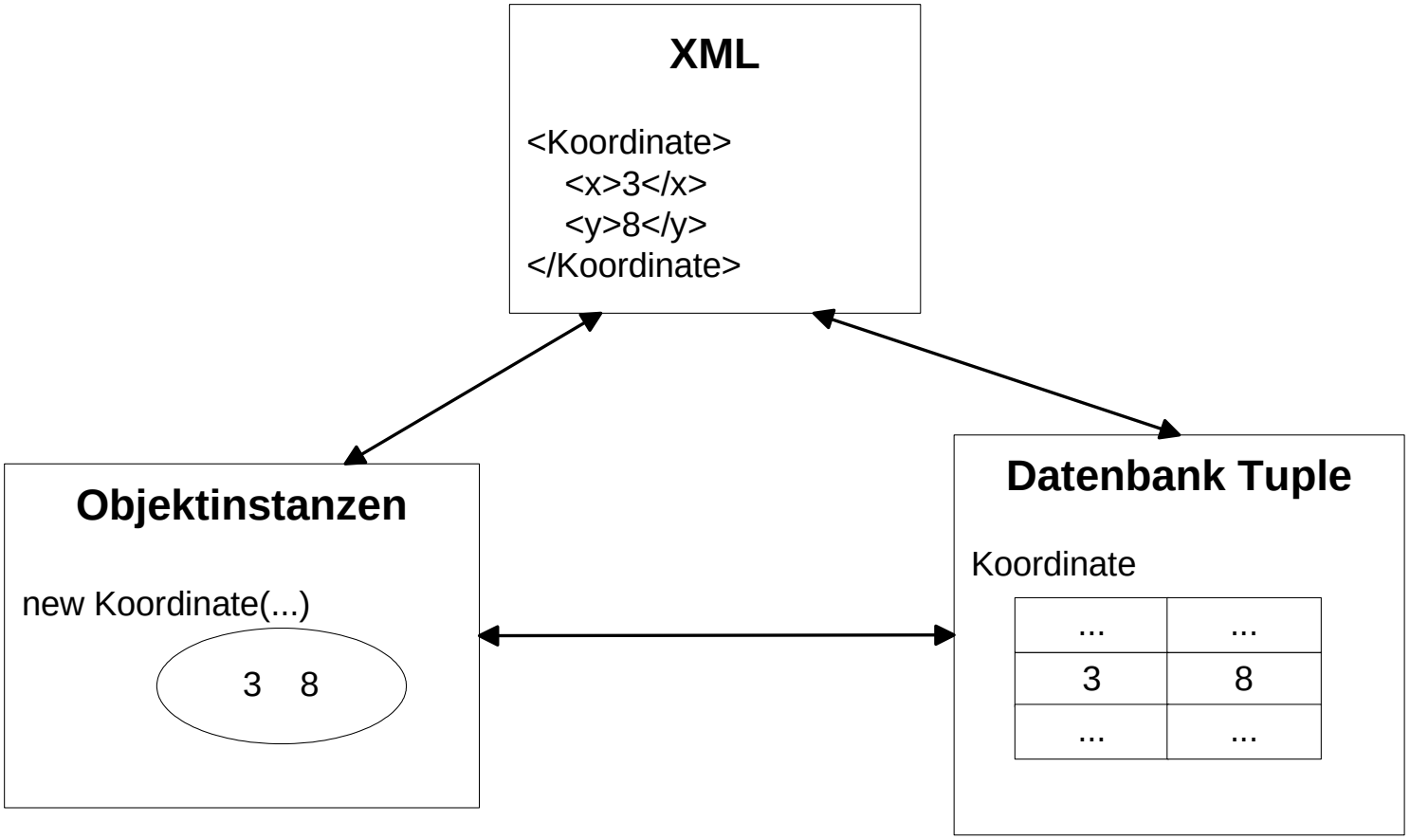
- XML-Schema beschreiben eine Klasse von Dokumenten
- XML-Dokumente sind somit eine Instanz einer Klasse von Dokumenten
- Dies ermöglicht parallelen zu objektorientierten Welt:
  - Der Zustand eines Objektes kann in einem XML-Dokument festgehalten werden
  - Die Struktur der Klasse liefert das Schema

- Die Grenzen zwischen XML, Strukturen in Programmier-sprachen und Datenbanken verschwimmen
- Einige Standards nehmen sich dieses Themas an
  - SQL 2003
  - Sun JDO
- Viele Tools sind bereits verfügbar
  - LINQ2XSD .NET
  - Hibernate

Mapping auf der Schema-Ebene



Mapping auf der Instanz-Ebene



Remote Procedure Calls mittels XML: SOAP

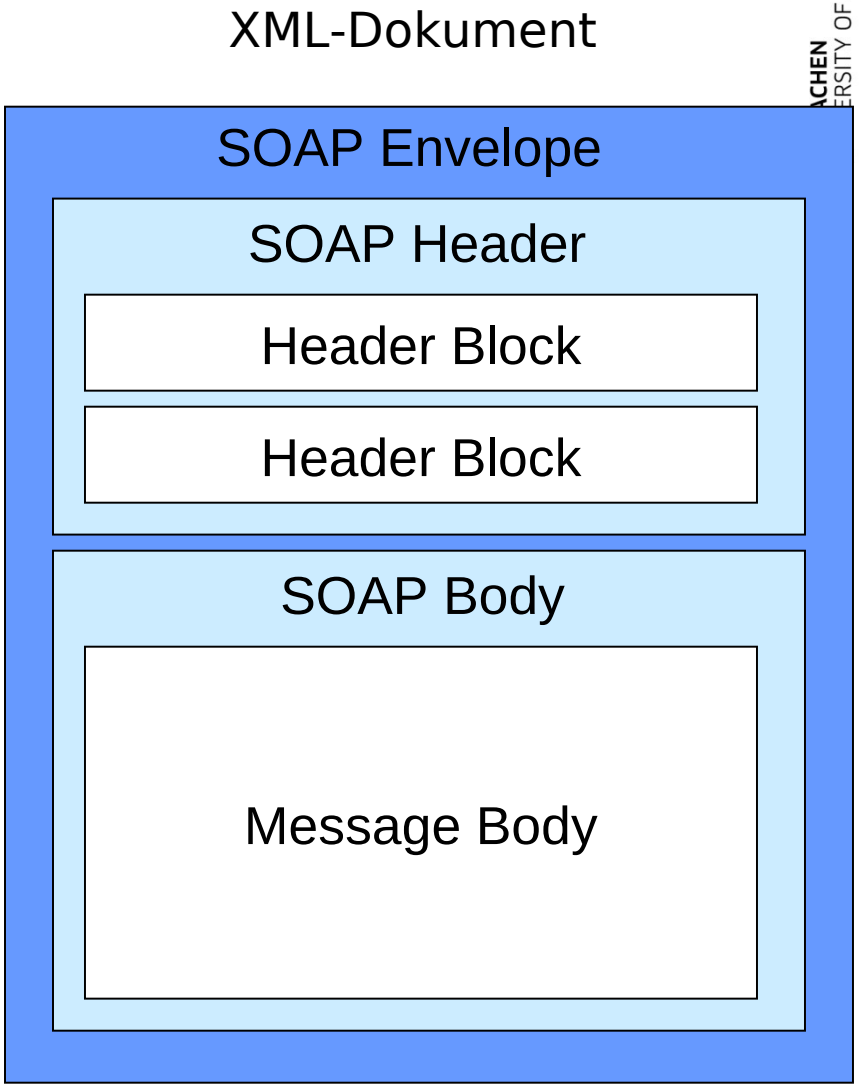
- SOAP (ursprünglich für Simple Object Access Protocol) ist ein Netzwerkprotokoll, mit dessen Hilfe Daten zwischen Systemen ausgetauscht und Remote Procedure Calls durchgeführt werden können.
- SOAP ist ein industrieller Standard des World Wide Web Consortiums (W3C)
- SOAP ist ein in XML definiertes, anwendungsunabhängiges Nachrichtenformat
- SOAP definiert auch, wie Anwendungsdaten kodiert werden sollen

XML und SOAP

- SOAP ist genauso wie XHTML eine Anwendung der XML-Spezifikation
- SOAP legt durch Einschränkung der erlaubten Tags und Attribute fest, welche Sprachmittel erlaubt sind
- XML-Schema und XML-Namespaces sind wichtige Komponenten bei der Beschreibung von SOAP
- SOAP regelt, wie Daten in der Nachricht abzubilden und zu interpretieren sind
- SOPA nutzt hierbei existierende Transportprotokolle, ohne diese jedoch festzulegen
- SOAP stellt so eine Konvention für entfernte Prozedur-/Methodenaufrufe dar

SOAP Nachrichten

- Eine SOAP-Nachricht besteht zunächst auf der obersten Ebene aus einem Envelope
- Der Envelope enthält einen optionalen Header sowie die eigentliche Nachricht im SOAP Body
- Aufgabe des Headers: Infos über Transaktionskontexte, Authentifizierung, Autorisierung, Routing- und Auslieferungsinformationen



TCPMonitor

Admin | Port 9090

Stop | Listen Port: 9090 | Host: localhost | Port: 8080 | Proxy

State	Time	Request Host	Target Host	Request...
---	Most Recent	---	---	---
Done	06/24/02 02:18:36 PM	localhost.localdomain	localhost	POST /axis/services/Hello HTTP/1.0
Done	06/24/02 02:18:36 PM	localhost.localdomain	localhost	POST /axis/services/Hello HTTP/1.0

Remove Selected | Remove All

Request

```
POST /axis/services/Hello HTTP/1.0
Content-Length: 489
Host: localhost
Content-Type: text/xml; charset=utf-8
SOAPAction: ""

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/e
 <SOAP-ENV:Body>
  <setHelloMessage>
   <arg0 xsi:type="xsd:string">ciao, mundi</arg0>
  </setHelloMessage>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Response

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 368
Date: Mon, 24 Jun 2002 13:18:36 GMT
Server: Apache Tomcat/4.0.4 (HTTP/1.1 Connector)
Set-Cookie: JSESSIONID=8F57C539BD75B6A07BBBEDE307F9DD31;Path=/axis

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope
 <SOAP-ENV:Body>
  <setHelloMessageResponse SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

XML Format | Save | Resend | Sw

Request: setHelloMessage

Request: getHelloMessage

TCPMonitor

Admin | Port 9090

Stop | Listen Port: 9090 | Host: localhost | Port: 8080 | Proxy

State	Time	Request Host	Target Host	Request...
---	Most Recent	---	---	---
Done	06/24/02 02:18:36 PM	localhost.localdomain	localhost	POST /axis/services/Hello HTTP/1.0
Done	06/24/02 02:18:36 PM	localhost.localdomain	localhost	POST /axis/services/Hello HTTP/1.0

Remove Selected | Remove All

Request

```
POST /axis/services/Hello HTTP/1.0
Content-Length: 419
Host: localhost
Content-Type: text/xml; charset=utf-8
Cookie: JSESSIONID=8F57C539BD75B6A07BBBEDE307F9DD31
SOAPAction: ""

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/e
 <SOAP-ENV:Body>
  <getHelloMessage/>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Response

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 480
Date: Mon, 24 Jun 2002 13:18:36 GMT
Server: Apache Tomcat/4.0.4 (HTTP/1.1 Connector)

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope
 <SOAP-ENV:Body>
  <getHelloMessageResponse SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/
   <getHelloMessageReturn xsi:type="xsd:string">ciao, mundi</getHelloMessageRetu
  </getHelloMessageResponse>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

XML Format | Save | Resend | Switch Layout | Close

JSON

JSON: JavaScript Object Notation

- JSON-Dokumente sind gültiges JavaScript
- Einfache Struktur
- Typisierung eingebaut!
- Beispiel:

```
{
    "id" : 2648,
    "Name" : "Mustermann",
    "Vorname" : "Max",
    "adr" : {
        "Stadt" : "Aachen",
        "plz" : 52064
    },
    "tel" : [ "0241 1234", "0160123456" ],
    "partner" : null,
    "maennlich" : true
}
```

Weitere Informationen: http://www.json.org/

JSON

Typen:

- `Number` (wie in JavaScript)
- `String` (nur in doppelten Anführungszeichen)
- `Boolean` (true/false)
- `Array` (nur in eckigen Klammern)
- `Object` (in geschweiften Klammern)
- `Null`

Deckt nicht alle möglichen JavaScript-Werte ab

- NaN, Infinity werden zu null serialisiert
- Function- und RegExp-Objekte werden verworfen

JSON in Java mittels *org.json*

Objektserialisierung:

```
DemoBean demo = new DemoBean();
demo.setId(1);
demo.setName("lorem ipsum");
demo.setActive(true);

JSONObject jo = new JSONObject(demo);
```

{"name":"lorem ipsum","active":true,"id":1}

Meistens nimmt man aber automatische Marshaller/Unmarshaller wie Jackson

FH Aachen
Fachbereich 9 Medizintechnik und Technomathematik
Prof. Dr.-Ing. Andreas Terstegge
Straße Nr.
PLZ Ort
T +49. 241. 6009 53813
F +49. 241. 6009 53119
Terstegge@fh-aachen.de
www.fh-aachen.de