Webservices mit JEE

Marc Nguidjol

marcnguidjol@gmail.com

Motivation

Webservices - SOA, verteilte Systeme

Alter Wein in neuen Schläuchen?

- Webservices stellen eine **Technologie** dar, die sich zur Aufgabe gemacht hat:

1. Fremdsysteme **lose zu koppeln** und sie miteinander **interagieren** zu lassen.
2. Interaktion **(Interoperabilität)**, schafft man mit **Standards**, an denen sich Hersteller und Anwender orientieren müssen.

Die Entstehung von Webservices

Verteilte Anwendungen

Client-Server C/S (Silos)

Web-Anwendungen

Peer-to-Peer (P2P)

Anwendungen C/S vs. Webservices

Client-Server Anwendungen

- Laufen in einem Container
- Eine Programmiersprache
- Prozedural
- Gebunden an einem bestimmten Transport-Protokoll
- Enge Kopplung
- Effiziente Verarbeitung

Webservices

- Benötigen keinen Container
- Mehrere Programmiersprachen
- Nachrichtenbasiert
- Unterschiedliche Transport-Protokolle
- Lose Kopplung
- Keine effiziente Verarbeitung

Web-Anwendungen vs. Webservices

Web-Anwendungen

- Benutzer-Interaktion
- Statische Integration von Komponenten
- Monolithische Dienste

Webservices

- Programme(Maschinen)-Interaktion
- Dynamische Integration von Komponenten
- Zusammensetzung von Dienste durch Orchestrierung

Probleme ohne Webservices

- Bisher in verteilte Anwendungen verwendete Technologien:
  (RPC, CORBA, DCOM, RMI)

- Aber…

1. Abhängigkeit von einer Programmiersprache
2. Abhängigkeit von einer (Lauf-)Umgebung
3. Ports müssen explizit bei der **Firewall** freigeschaltet werden

Wozu Webservices?

- Allgemeiner Ansatz für die **lose gekoppelte** Entwicklung in verteilte Systemen!

- Alternative zu bekannten Lösungen wie Remote Prozedur-Aufrufe (RPC)

1. Middleware Lösungen: CORBA über IIOP (binäres Protokoll), RMI, …
2. RPC über HTTP-Protokolle: XML, SOAP

Use Cases Webservices

- implementieren häufig keine neuen Systeme, können eine **Fassade** für legacy Systeme sein!
- abstrahieren zudem von Programmiersprache und Plattform mit der Anwendungen realisiert sind
- ermöglichen wie JMS die Integration von verteilten (meistens heterogenen) Systemen

- Zwei Erscheinungsformen:
  RPCs (synchron), Messaging (asynchron)

SOA (Service Oriented Architecture)

SOA is a **paradigm** for organizing and utilizing distributed capabilities that may be under the control of different ownership domains
Wikipedia.org

SOA (Service Oriented Architecture)

- SOA ist als **Modell einer Architektur** anzusehen

- Menge von Diensten

1. Dienst muss immer eine komplette Business-Funktion beschreiben
2. Dienste können sich über **Orchestrierung** zu komplexeren Prozesse zusammengesetzt werden

- Webservices könnten die allgemein akzeptierte Strategie der Umsetzung einer SOA!

Was ist ein Dienst?

- kleinster fachlich und wieder verwendbare Bestandteil einer SOA

1. beschreibt eine eigenständige Funktionalität
2. eigenständiges Artefakt
   (unabhängig von der Anwesenheit anderer Dienste)
3. offene Schnittstelle
   (Dienstnutzer kann allein aufgrund dieser Beschreibung die Kommunikation aufnehmen).
4. Kenntnisse seines Interna sind nicht notwendig, um den Dienst nutzen zu können!

Agenda

- Webservices Allgemein
- W3C Standards und Protokolle
- Webservices mit JEE - JAXWS, JAXRS

Vorwort

- Gut zu wissen

1. Kenntnisse in XML, XML Schema und Namespaces
2. Kenntnisse in Java SE / EE und Web

- Organisatorisch

1. 09:00 – 16:00
2. Mittag 12:00

Agenda

- **Webservices Allgemein**
- W3C Standards und Protokolle
- Webservices mit JEE - JAXWS, JAXRS

Webservices - W3C

- Webservices Allgemein

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically **WSDL**). Other systems interact with the Web service in a manner prescribed by its description using **SOAP messages**, typically conveyed using **HTTP** with an **XML serialization** in conjunction with other **Web-related standards**.
Web services provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks.
https://www.w3.org/TR/ws-arch/

Webservices - Wikipedia

- Webservices Allgemein

Ein Webservice (auch Webdienst) ermöglicht die Maschine-zu-Maschine-Kommunikation auf Basis von HTTP oder HTTPS über Rechnernetze wie das Internet.
https://de.wikipedia.org/wiki/Webservice

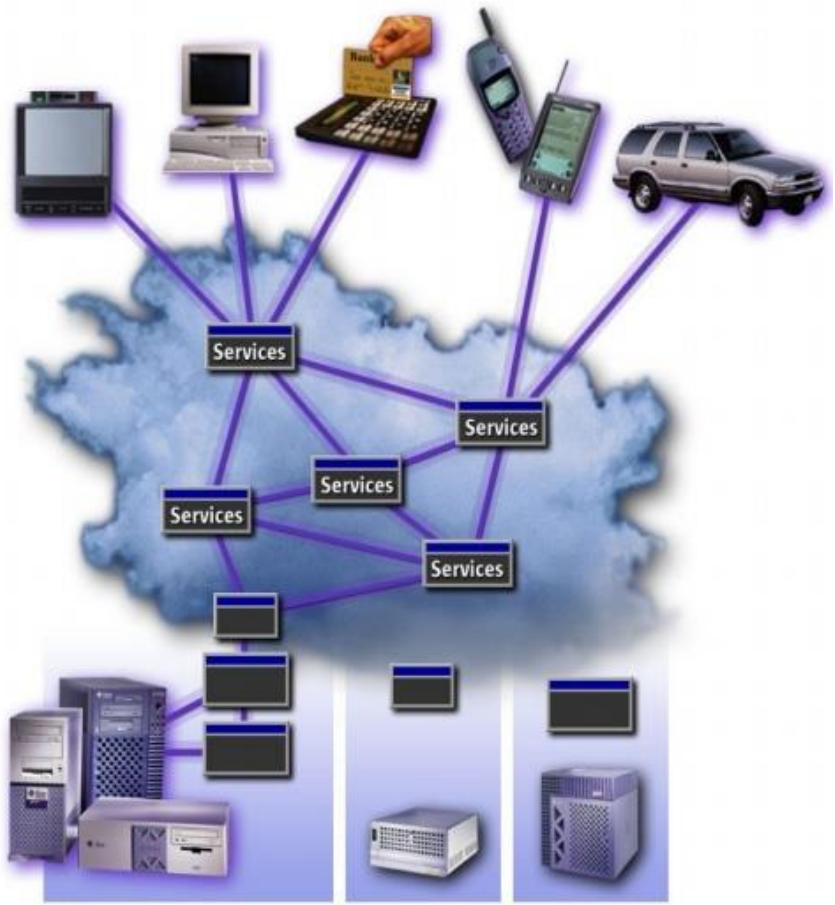
Merken - Was sind XML Webservices?

- **Software Komponenten**

1. Bieten Ihre Dienste über Endpunkte (URIs) an
2. Schnittstelle in einem Maschinen lesbaren Format *(WSDL)*
3. Kommunizieren über XML Nachrichten

- **Vorteile:**

1. Sprach- und Plattformunabhängigkeit
2. Lose Kopplung



WS-I Compliance / Basic Profile

- **Interoperabilität** zwischen Anbieter und Konsumenten von Webservices kann nur gewährleistet durch:

1. Einhaltung von etablierten **Web Standards**
2. Austausch über **XML Nachrichten**
3. Zentraler Ort zum Publizieren und Auffinden von Webservices (UDDI: Gelbe Seiten)

- Mehr dazu: *http://ws-i.org/profiles/basicprofile-2.0-2010-11-09.html*

Web(W3C) Standards?

- Etablierte (Web-)Standards (XML, XSD, SOAP, WSDL)
- Protokolle HTTP(s)
- Architecture Style REST

Fazit - W3C Standards

- Anwendung dieser W3C Standards ermöglicht dass Webservices

1. von allen **Sprachen** genutzt,
2. von allen **Betriebssystemen** integriert und
3. von allen **Plattformen** erstellt werden können

Agenda

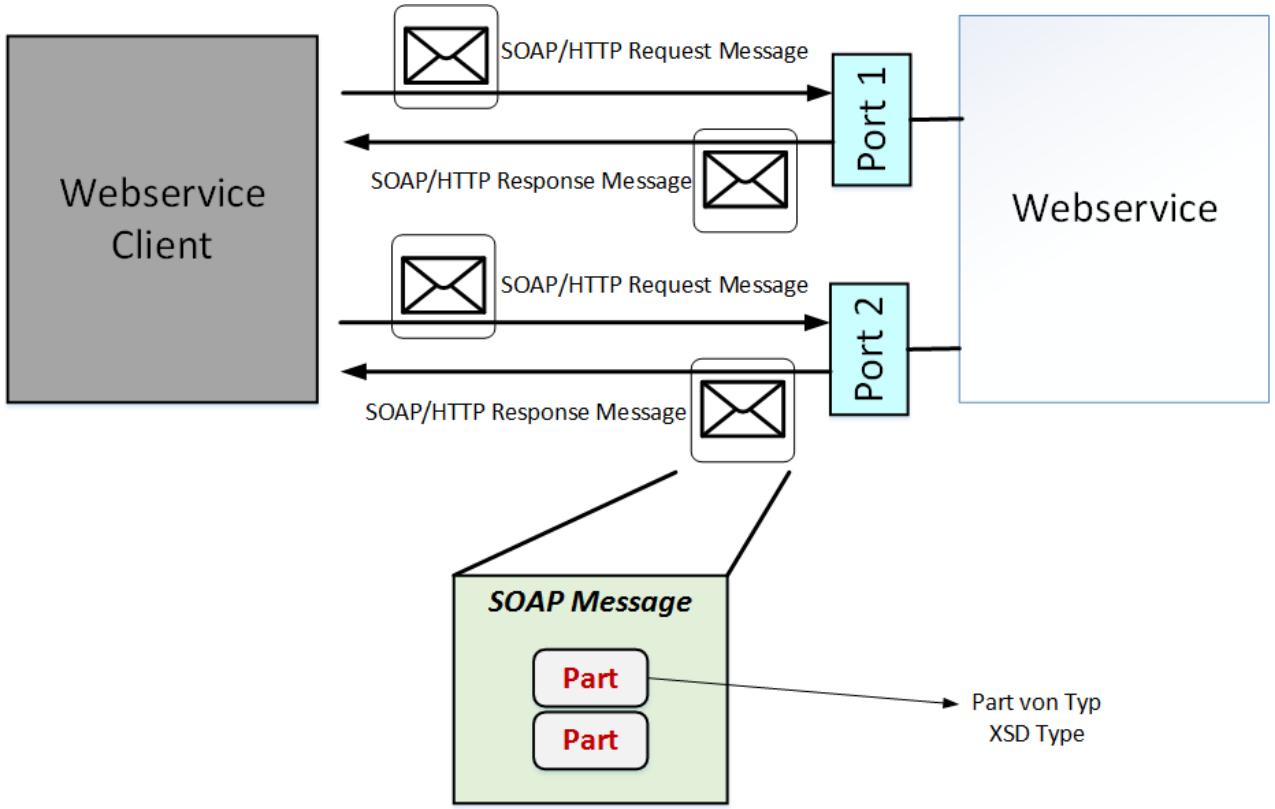
- Webservices Allgemein
- **W3C Standards und Protokolle**
- Webservices mit JEE - JAXWS, JAXRS

SOAP

Spezifikation

SOAP Webservices

- **XML Nachrichten**



SOAP?

- Es muss auf jeden Fall festgelegt werden wie

1. Methoden beim RPC in XML Nachrichten kodiert(serialisiert) werden **procedure (param-1, …, param-N)**
2. Fehlermeldungen in XML Nachrichten serialisiert werden
3. Arrays type[ ] und Matrizen type[ ][ ] serialisiert werden

- Und genau dies leistet SOAP!

SOAP-Spezifikation 1/2

- SOAP ist die **Kommunikationskomponente** von XML Webservices
- Art Protokoll für den **Nachrichtenaustausch** zwischen Webservice-Clients und Webservice-Anbieter

1. RPC (XML Nachrichten)
2. Wegen XML, Plattform- & Programmiersprachenunabhängig

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

SOAP-Spezifikation 2/2

- Früher Simple Object Access Protocol, Heute **keine Akronym** mehr!

- Zustandslose (Stateless) Kommunikation im Austausch von XML Nachrichten

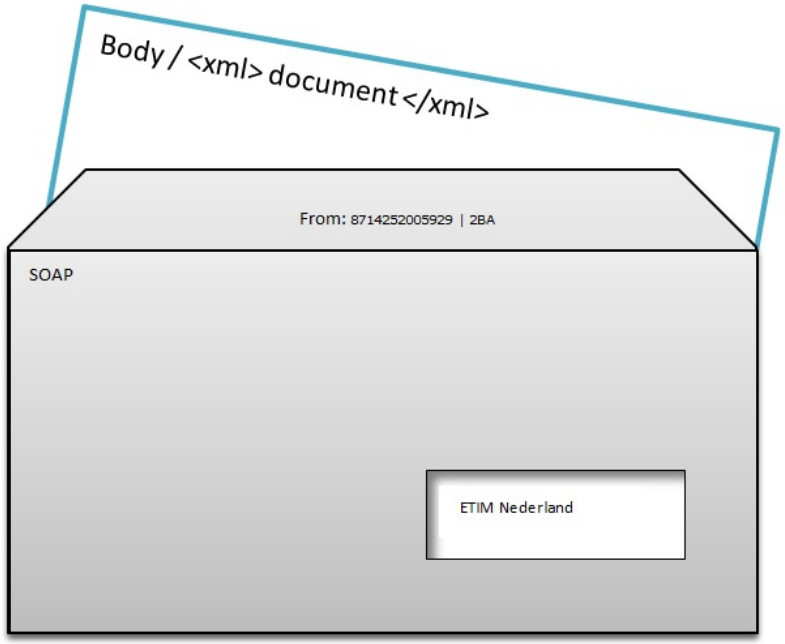
1. Dafür unterschiedliche Message Exchange Patterns (MEP)
2. Nachrichten-Struktur standardisiert (SOAP Encoding Style)
3. Etablierte Transport Protokolle (SOAP Binding)

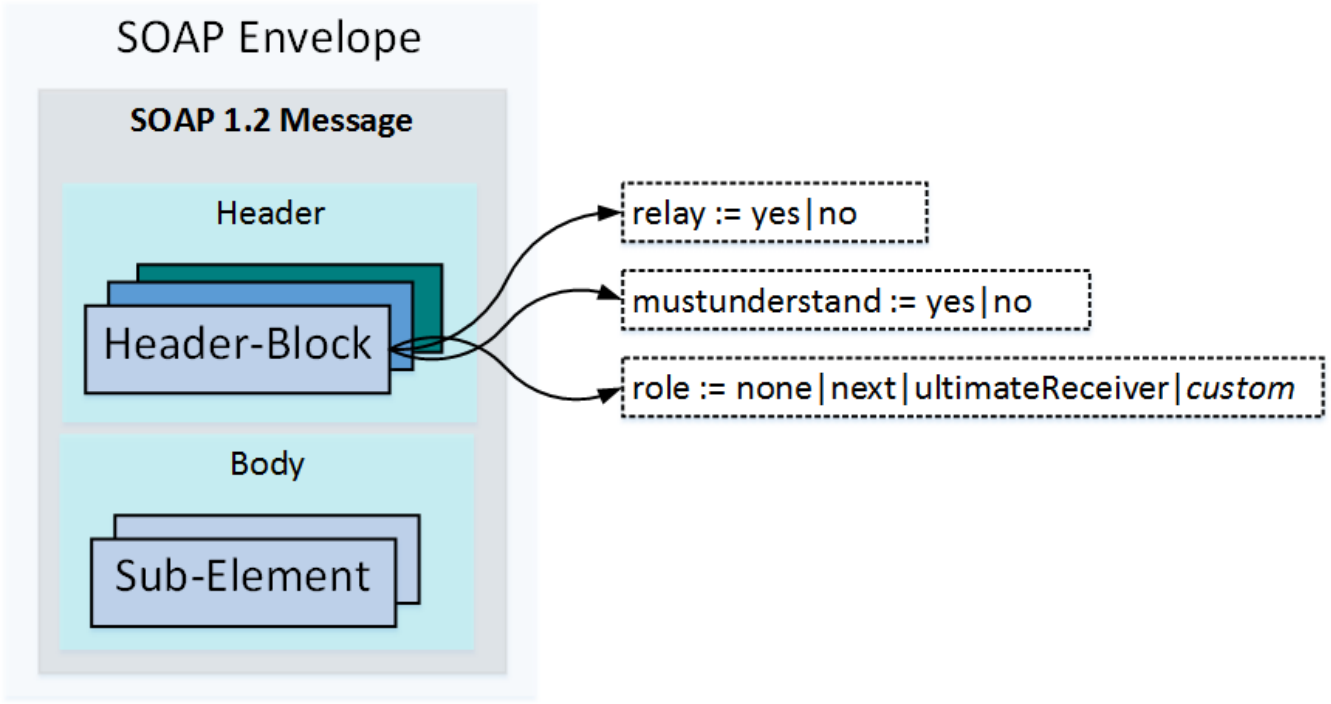
Definition (SOAP)

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 stützt sich auf XML zur Repräsentation der Daten und auf Internet-Protokolle der Transport- und Anwendungsschicht (vgl. TCP/IP-Referenzmodell) zur Übertragung der Nachrichten.
Die gängigste Kombination ist SOAP über HTTP und TCP.
https://de.wikipedia.org/wiki/SOAP

SOAP (XML-) Nachricht / SOAP Envelope

- **SOAP Envelope** (Header und Body)





SOAP Namensräume

- Folgende Namensräume sind notwendig um jede XML Nachricht als **SOAP Nachricht** zu deklarieren:

1. SOAP Envelope:
   http://www.w3.org/2003/05/soap-envelope/
2. encodindStyle Attribut:
   SOAP Encoding und Datentypen
   http://www.w3.org/2003/05/soap-encoding

3. Weitere Attribute: *actor, mustUnderstand*

Beispiel: SOAP Envelope

```
POST /OrderEntry HTTP/1.1
Host: localhost
Content-Type: application/soap;  charset="utf-8"
Content-Length: 420

<?xml version = "1.0"?>
<soap:Envelope
 xmlns:soap = "http://www.w3.org/2003/05/soap-envelope"
 soap:encodingStyle = " http://www.w3.org/2003/05/soap-encoding">
   ...
   Message information goes here
   ...
</soap:Envelope>
```

SOAP Header

- Optional in jede SOAP Nachricht:

1. Context relevante Informationen
2. Authentifizierung
3. Transaktionen
4. …

- Sammlung von **Block optionalen Elementen** (Header Entries)

Beispiel: SOAP Header

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
<soap:Header>
  <m:Transaction xmlns:m="http://localhost:8080/ws/transactions/"
        soap:mustUnderstand="1">123467890
  </m:Transaction>
</soap:Header>
...
</soap:Envelope>
```

SOAP Headers - SOAP Knoten

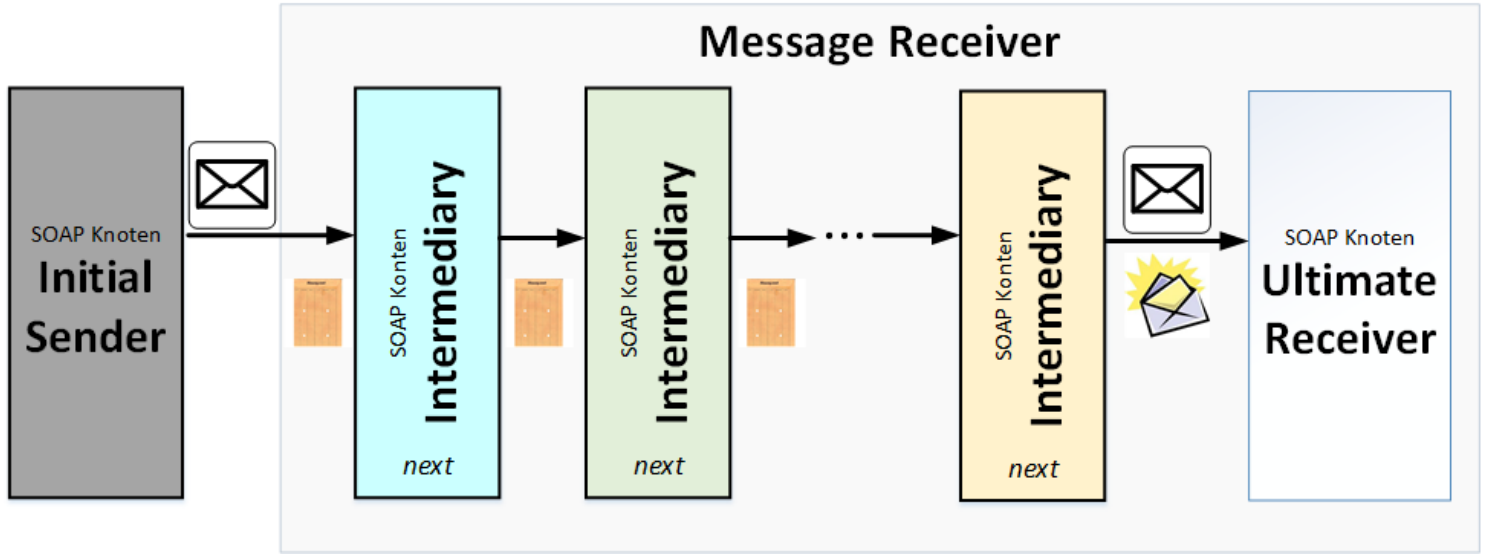
- Mächtiges **Konzept zur Erweiterung** von SOAP-Nachrichten:

1. Header Elemente (SOAP Knoten) können in einer SOAP-Nachricht hinzugefügt werden, ohne die ursprüngliche Nachrichteninhalt (Body) zu modifizieren
2. Einzelne Header Elemente (Blocks) sind immer **voneinander unabhängig**!

- Art **Interceptors**, erledigen eine bestimmte Verarbeitung bevor die SOAP Nachricht dem Empfänger erreicht

SOAP Message Path

SOAP Processing Model



- Nachricht kann auf dem Weg zum Empfänger kann von **0 bis N Zwischen-SOAP-knoten (Intermediaries)** verarbeitet werden!

SOAP Knoten - Standardisierte URI

- SOAP Knoten werden **Roles** als ***URI*** zugeordnet:

1. Anwendungsspezifische URI
   z.B. http://…./log muss vom Empfänger interpretiert werden können
2. SOAP 1.2 Standardisierte URIs

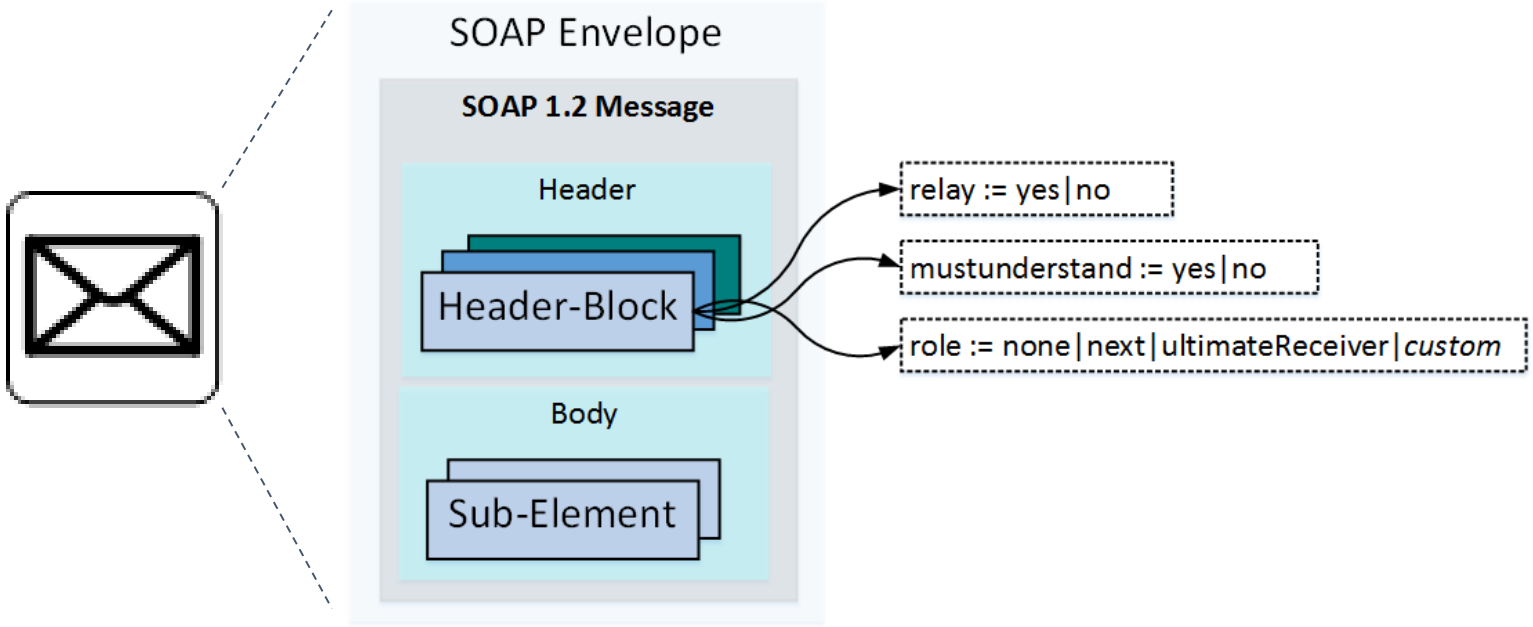
SOAP Roles	
Role Name	**Role URI**
next	http://www.w3.org/2003/05/soap-envelope/role/next
none	http://www.w3.org/2003/05/soap-envelope/role/none
ultimateReceiver	http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver

Standardisierte SOAP-Knoten

- Initial SOAP sender
  Erzeugt und sendet die SOAP Nachricht, Startpunkt des SOAP Message Path
- SOAP Intermediary
  Empfängt die SOAP Nachricht, verarbeitet, modifiziert und löscht an ihm adressierten Teile der Nachricht bevor die Nachricht weitergeleitet wird
- Ultimate SOAP Receiver
  Endpunkt des SOAP Message Path (Empfänger)

Vorteile der schrittweisen Verarbeitung ermöglicht eine Aufgabenverteilung auf spezialisierte Server!

Fazit: SOAP-Header



- Attribute: *mustUnderstand, relay*
  bestimmen ob ein Knoten den Header-Block verarbeiten muss oder ignorieren darf

SOAP Body

- Pflichtelement in jeder SOAP Nachricht:

1. Eigentliche Nachricht (Payload)
2. Fehlernachricht (Fault)

- *End-to-End* Informationen:

1. Anwendungsdaten als XML Dokument *(document style)*
2. Methodenaufruf mit Parameter als RPC *(rpc style)*
3. Fehlerinformationen als SOAP Fault

Beispiel 1: SOAP Nachricht - Request

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

<soap:Body>
  <m:GetPrice xmlns:m="http://localhost:8080/ws/catalog/prices">
    <m:Item>Apples</m:Item>
  </m:GetPrice>
</soap:Body>

</soap:Envelope>
```

Beispiel 2: SOAP Nachricht - Response

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

<soap:Body>
  <m:GetPriceResponse xmlns:m="http://localhost:8080/ws/catalog/prices">
    <m:Price>8.50</m:Price>
  </m:GetPriceResponse>
</soap:Body>

</soap:Envelope>
```

Beispiel 3: SOAP Fault

```
<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
xmlns:xsd = "http://www.w3.org/1999/XMLSchema">
  <soap:Body>
    <soap:Fault>
      <faultcode xsi:type = "xsd:string">soap:Client</faultcode>
      <faultstring xsi:type = "xsd:string">
        Failed to locate method (validateCard) in class (CreditCardValidator)...
      </faultstring>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

SOAP Fault

- Nur einmal in einer SOAP Nachchricht erlaubt

- Transportiert **Fehler** und/oder **Status Informationen**

1. Technischer Code - *faultcode* mit vordefinierte Werte *VersionMismatch, MustUnderstand, Client, Server*
2. Fehlerbeschreibung - *faultstring*
3. Auslöser des Fehlers - *faultactor*
4. Anwendungsspezifische Daten zu der Fehlermeldung - *detail*

SOAP

Encoding von SOAP Nachrichten (XML Serialisierung)

SOAP - Body *Encoding Style*

- *style="rpc“*
  (SOAP-Nachricht = **Prozedur**)

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

- *style="document“*
  (SOAP-Nachricht = **XML Dokument**)

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

- Legt lediglich die **Struktur** vom **SOAP-Body** fest!
- Darüber hinaus keine Bedeutung!

RPC/Document - literal

When using a **literal use model**, the **body contents should conform** to a user-defined **XML-schema(XSD) structure**.
The advantage is two-fold. For one, you can validate the message body with the user-defined XML-schema, moreover, you can also transform the message using a transformation language like XSLT.
Stackoverflow

RPC/Document - encoded

With a (SOAP) **encoded use model**, the message has to use **XSD datatypes**, but the structure of the **message need not conform to any user-defined XML schema**. This makes it difficult to validate the message body or use XSLT based transformations on the message body.
Stackoverflow

Fazit: SOAP Body Encoding

- Verschiedene *Style* Angaben in der WSDL möglich

1. document/literal
2. document/encoded
3. rpc/literal
4. rpc/encoded

- Mehr dazu:
  Russell Butek (IBM)
  https://www.ibm.com/developerworks/library/ws-whichwsdl/

Beispiel 1: document/literal WRAPPED

public void meineWebMethode (int x, float y);

```
<soap:envelope>
   <soap:body>
      <meineWebMethode xmlns="<targetNS>">
         <xElement>5</xElement>
         <yElement>5.0</yElement>
      </meineWebMethode>
   </soap:body>
</soap:envelope>
```

```
@WebService
@SOAPBinding(
   style = SOAPBinding.Style.DOCUMENT
   use = SOAPBinding.Use.LITERAL
   parameterStyle = SOAPBinding.ParameterStyle.WRAPPED)
```

- **Default SOAP 1.2 Binding**
- Bessere Performanz (da keine Typkodierung mehr in der Nachricht)
- Einfache Schema **Validierung** möglich, da *types* Abschnitt im Schema vorhanden
- Methodenname erscheint im Body

Beispiel 2: document/encoded *Style*

public void meineWebMethode (int x, float y);

```
<soap:envelope>
   <soap:body>
      <xElement>5</xElement>
      <yElement>5.0</yElement>
   </soap:body>
</ soap:envelope >
```

- Keine Typkodierung in der SOAP Nachricht.
- **Methodenname** erscheint **nirgendwo** mehr im Body, nur Methoden-Parameter !
- Unübersichtlich (kann komplex werden)!

Beispiel 3: WSDL Abschnitt

public void meineWebMethode (int x, float y);

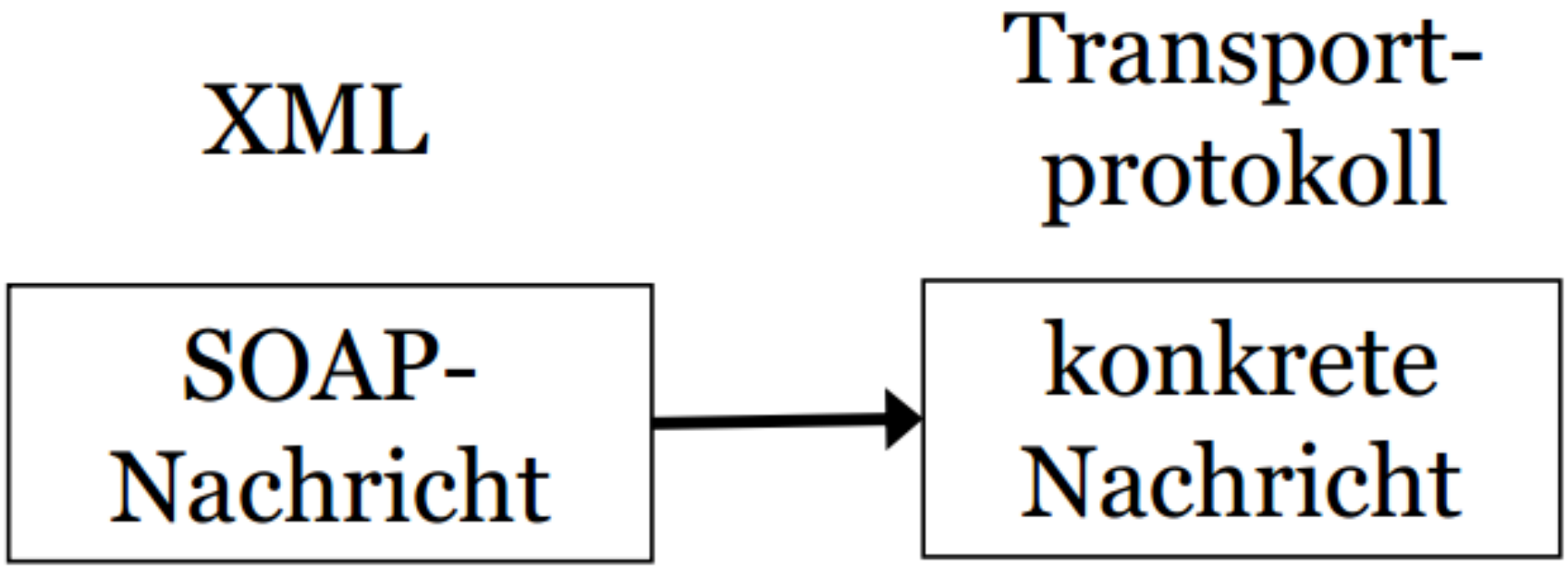
```
<types>
   <schema>
      <element name="xElement" type="xsd:int" />
      <element name="yElement" type="xsd:float" />
   </schema>
</types>
<message name="meineWebMethodeRequest">
   <part name="x" element="xElement" />
   <part name="y" element= "yElement" />
</message>
<binding …>
```

WSDL - Abschnitt

SOAP

Übertragung von SOAP Nachrichten (Protocol Binding)

Protocol-Binding?



Standard - Protokoll?

- Übertragung von SOAP Nachrichten

1. muss **ohne Informationsverlust** erfolgen!
2. Auch beliebige andere Formate sein als XML möglich

- WS-I BP 1.1:

1. HTTP (-Binding)
2. Einzige standardisierte Protokoll-Bindung für SOAP Nachrichten!
3. Modell Request/Response von SOAP passt sehr gut zu HTTP Request/Response!

Fazit: HTTP-Binding

- Empfehlung HTTP-Bindungen: HTTP-POST, HTTP-GET

1. SOAP HTTP Request benutzt **POST**
2. SOAP HTTP Response mit **HTTP Code** (200 *OK Antwort*, 500 *Fault*)

- Verwendung (Message Exchange Pattern)

1. http://www.w3.org/2003/05/soap/mep/**request-response** --> "POST“
2. http://www.w3.org/2003/05/soap/mep/**soap-response** --> "GET"

Beispiel: SOAP 1.2 Request

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

WS-I BP 1.1 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>
```

WS-I BP 1.1 SOAP-Nachricht

Beispiel: SOAP 1.2 (Antwort)

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

WS-I BP 1.1
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>
```

WS-I BP 1.1
SOAP-Response

Fazit: SOAP Protocol-Binding

- SOAP legt nur die Struktur der Nachrichten fest, aber wie diese Nachrichten ausgetauscht werden, obliegt das **Protocol-Binding**
- Am meisten verwendete Protokolle: HTTP und SMTP
- SOAP Nachricht über HTTP mit 2 Pflicht HTTP Header-Attribute: *Content-Type* und *Content-Length*

POST /InStock HTTP/1.1
Content-Type: **application/soap+xml; charset=utf-8**
Content-Length: 250

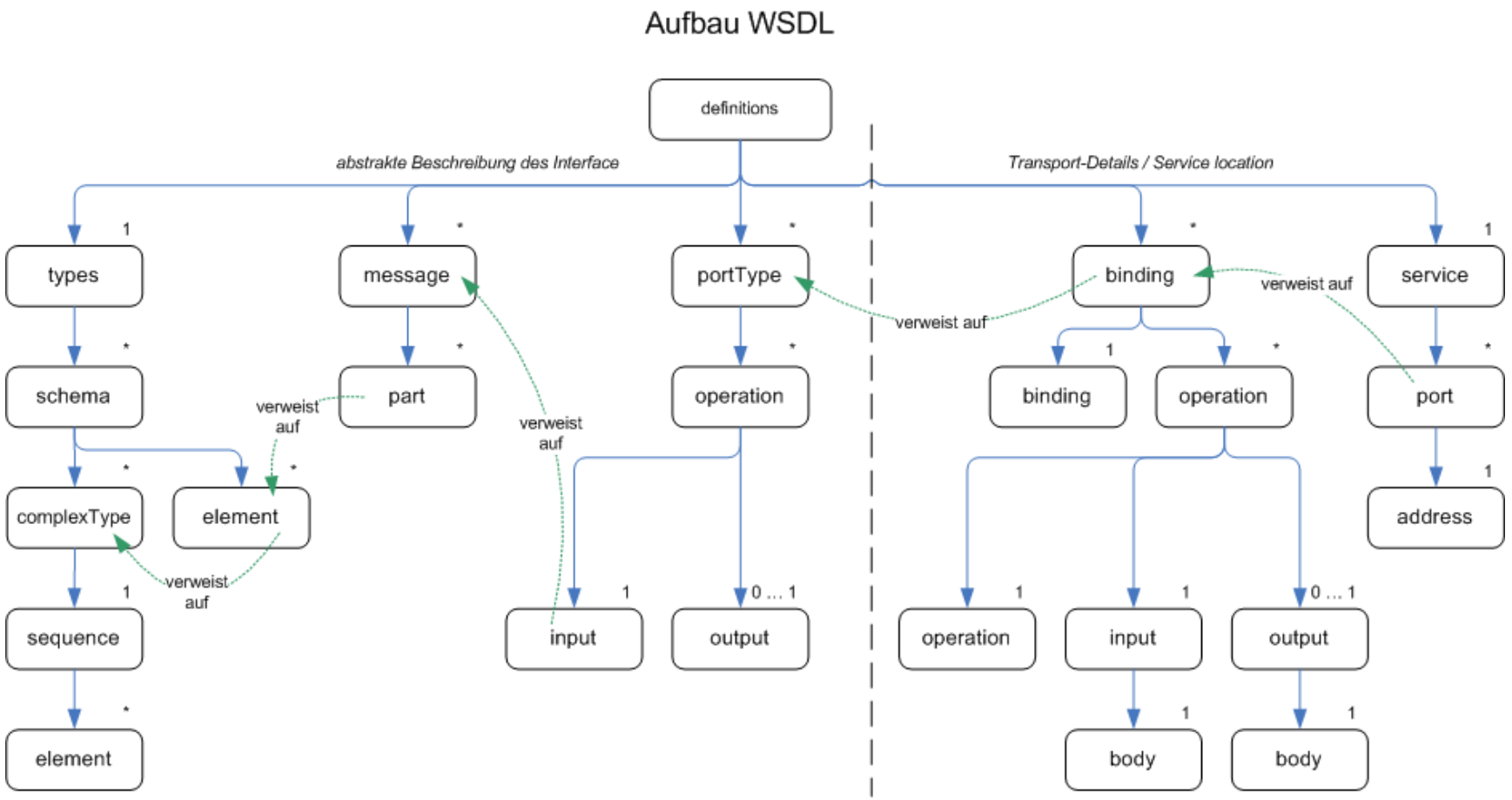
WSDL

Interface (WSDL)

- Definition

Die **Web Services Description Language** (**WSDL**) ist eine plattform-, programmiersprachen- und protokollunabhängige Beschreibungssprache für Netzwerkdienste (Webservices) zum Austausch von Nachrichten auf Basis von XML.
WSDL ist ein industrieller Standard des World Wide Web Consortiums (W3C).
https://de.wikipedia.org/wiki/Web_Services_Description_Language

WSDL Elemente



https://upload.wikimedia.org/wikipedia/commons/8/87/WSDL.png

Beispiel

```
<wsdl:definitions
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl"
  targetnamesapce="my namespace here" xmlns="my namespace here">

 <wsdl:types> W3C XML Schema data type definitions.... </wsdl:types>
 <wsdl:message name ="some operation input"> soap messages ... </wsdl:message>
 <wsdl:message name ="some operation output"> soap messages ... </wsdl:message>
 <wsdl:portType name ="port type" > set of operations + messages involved </wsdl:portType>

<wsdl:binding name ="binding name" type= "tns:port type above" >
    protocol and data format specification....
 </wsdl:binding>

<wsdl:service> define a port using the above binding and a URL .... </wsdl:service>

</wsdl:definitions>
```

MEP in der WSDL

- Messages Exchange Patterns:

1. One Way
   Operation mit Input Message, aber ohne Output
2. **Request-Response**
   Operation mit Input und Output Message (evtl. fault Message)
3. Solicit-Response
   Operation mit Output Message und ein Input Message
4. Notification
   Operation mit Output Message

Agenda

- Webservices Allgemein
- W3C Standards und Protokolle
- **Webservices mit JEE - JAXWS, JAXRS**

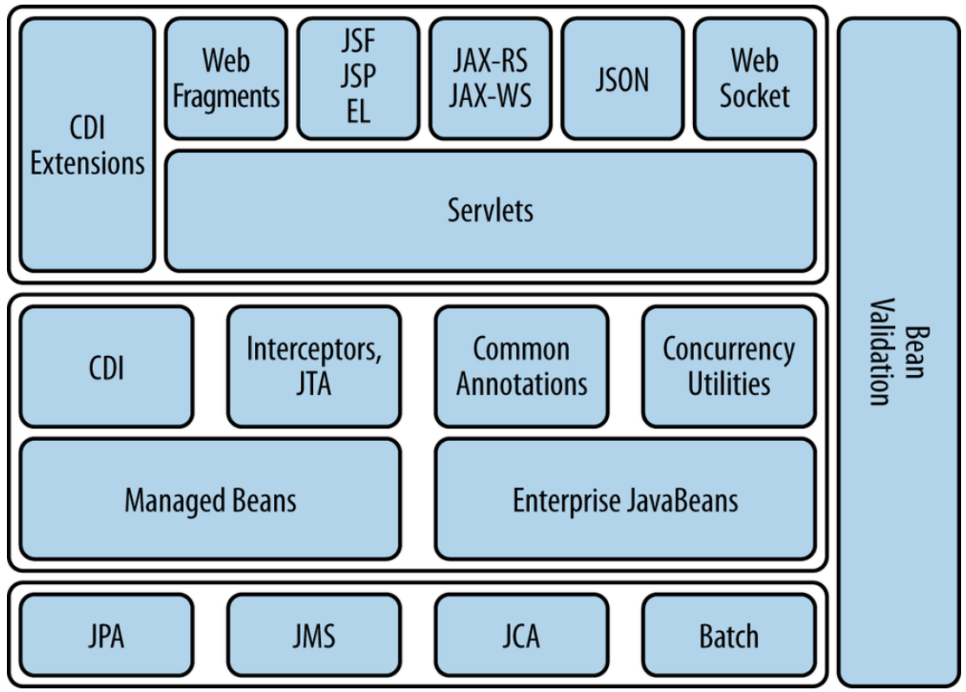
Webservices - Frameworks

- Für die Entwicklung von Webservices:

1. JAX-WS Spezifikation SUN (ORACLE) (jax-ws.dev.java.net) *
2. AXIS 1 und 2 (ws.apache.org/axis, ws.apache.org/axis2)
3. Apache CXF (cxf.apache.org)
4. XFire Codehaus (xfire.codehaus.org)
5. JBossWS JBoss (www.jboss.org/jbossws) *

Webservices - JEE Plattform

- JEE Plattform (JAX-WS, JAX-RS)



Java XML Webservices

JAXWS

JAX-WS Grundlagen

- Java API for XML Web Services **(JAX-WS)**

1. Programmiermodel (Server- und Clientseitig)
2. Toolkit bzw. Werkzeuge (wsimport, wsgen)

- Standard und Referenzimplementierung für Webservices in Java (**METRO** bzw. **JAX-WS RI**, *https://javaee.github.io/metro-jax-ws/*)

1. Aktuelle Version 2.3.0 (Stand 10/2018)
2. Nachfolger von JAX-RPC und seit JDK 6 in JRE eingebunden

JAX-WS Spezifikationen

- Setzt sich aus folgenden **JSR Spezifikationen** zusammen:

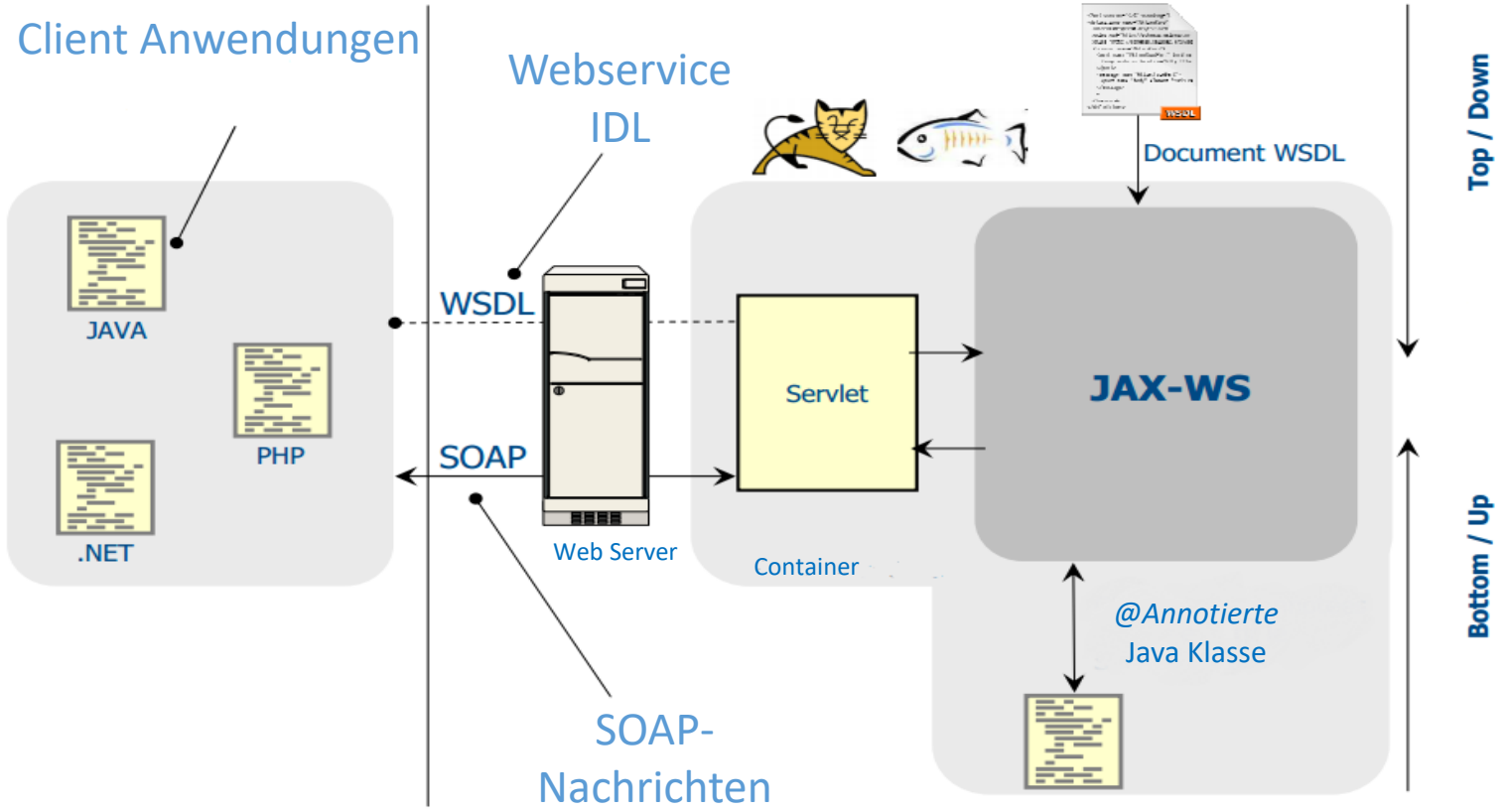
1. JSR 224 Java API for XML Web Services (JAX-WS)
2. JSR 181 Web Services Metadata for the Java Platform
3. JSR 222 Java Architecture for XML Binding (JAXB)
4. JSR 250 Common Annotations for the Java Platform

Serverseitiges Programmiermodel

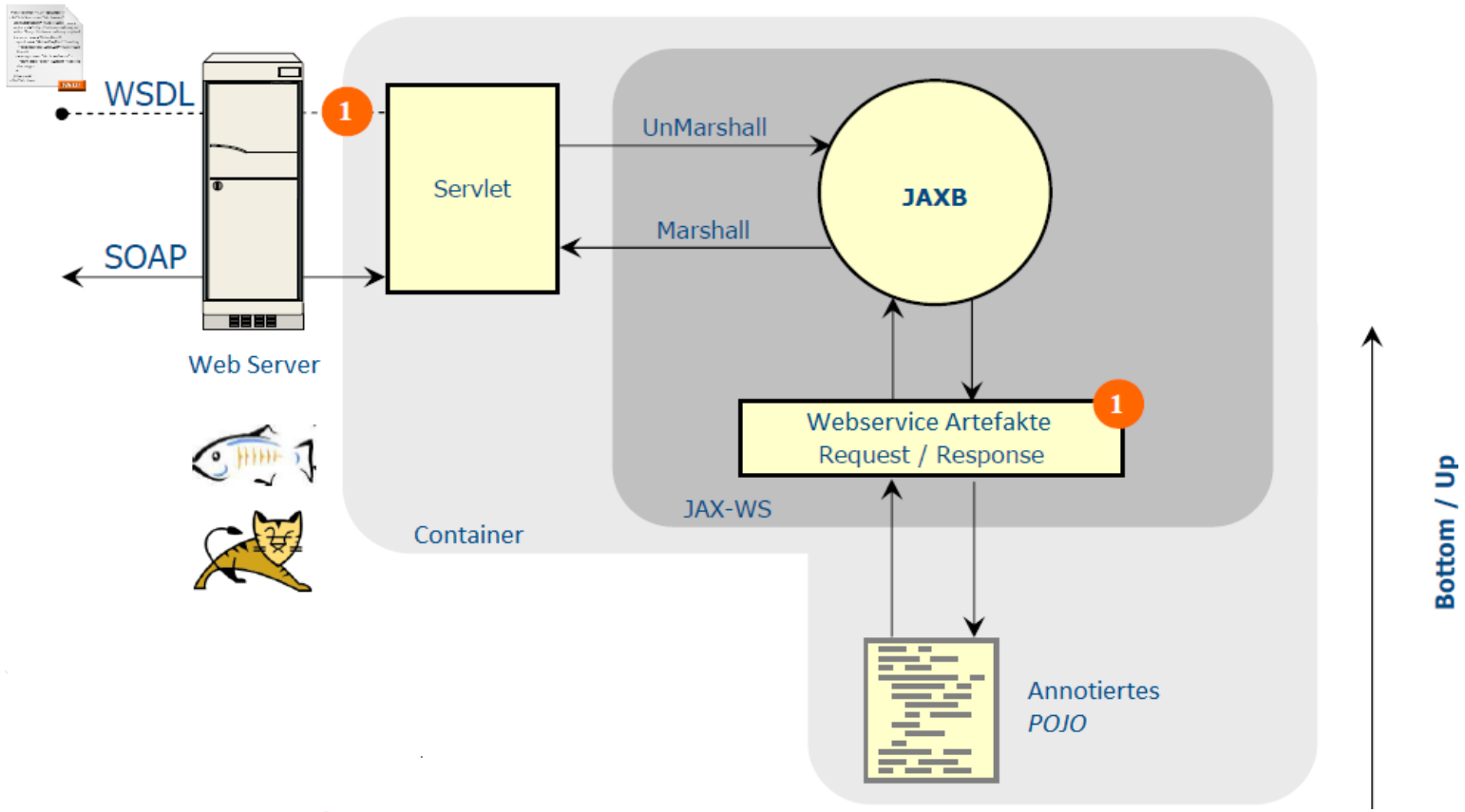
- Ansätze:

1. **top-down (Contract-First)**
   Aus der WSDL, werden mit **wsimport**
   WS-Artefakte (JAXB) generiert
2. **bottom-Up (Code-First)**
   Aus Pojos mit JAXWS-Annotationen, werden mit **wsgen**
   WSDL/XSD generiert

Serverseitig: Contract-First/Code-First



Code-First Ansatz



1 WSDL automatisch mit *wsgen* generiert

Code-First

- Die Entwicklung von Java Webservices basiert auf **POJOS**.
- Einsatz von **@JAX-WS Annotations** für die Basis Entwicklung erforderlich.

- Keine Deployment Descriptoren mehr!

1. Ausnahme *Tomcat (web.xml + sun-jaxws.xml)*

Beispiel

```java
/**
 * Simple POJO with JAXWS Annotations as Webservice
 * @author ngj
 */
@WebService(
        name = "Calculator",
        serviceName = "CalculatorWS",
        portName = "CalculatorIF",
        targetNamespace = "http://ws.jaxws.edu")
public class CalculatorPojo {

    /**
     * Web service operation add2IntegerNumbers
     * @param firstNumber
     * @param secondNumber
     * @return
     */
    @WebMethod(operationName = "add")
    public BigInteger add2IntegerNumbers(@NotNull Integer firstNumber, @NotNull Integer secondNumber) {
        int firstIntValue = firstNumber;
        int secondIntValue = secondNumber;
        int sum = firstIntValue + secondIntValue;
        return new BigInteger(String.valueOf(sum));
    }

}
```

Code-First - Pojo

- **POJO/POJI** mit **@WebService**

1. Alle public- nicht static-Methoden sind automatisch Webservice Operationen!

2. Ausnahmen: **@WebMethod(exclude=true/false)**
3. Überladung von Methoden **NICHT** zulässig!

- Anschließend Webservice deployen/publizieren (JRE > 5 oder JEE Container)

JAX-WS Runtime im JEE Container

- Wildfly 13

```
11:02:02,256 INFO  [org.jboss.ws.cxf.metadata] (MSC service thread 1-1) JBWS024061: Adding service endpoint metadata: id=edu.ja
 address=http://localhost:8080/ws-bottomup/CalculatorWS
 implementor=edu.javaee.ws.bottom.up.CalculatorPojo
 serviceName={http://ws.jaxws.edu}CalculatorWS
 portName={http://ws.jaxws.edu}CalculatorIF
 annotationWsdlLocation=null
 wsdlLocationOverride=null
 mtomEnabled=false
```

JAX-WS Runtime in JRE (>5)

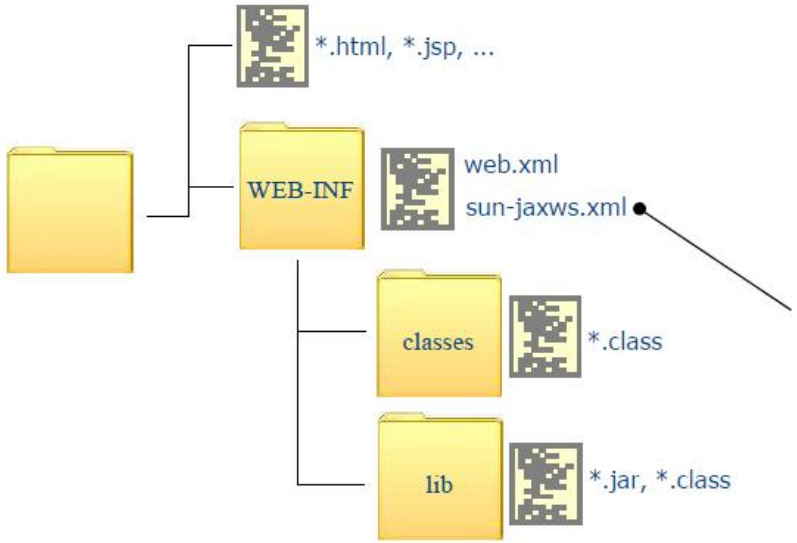
```java
/**
 * Endpoint Publisher for Webservice CalculatorPojo
 * @author ngj
 */
public class Main {

    final static String CALCULATOR_WS_URI = "http://localhost:9090/ws-bottomup/CalculatorWS";

    public static void main(String[] args){
        CalculatorPojo myCalculatorWS = new CalculatorPojo();
        Endpoint endpointCalculatorWS = Endpoint.publish(CALCULATOR_WS_URI, myCalculatorWS);
        if(endpointCalculatorWS.isPublished()){
            System.out.printf("--- CalculatorWS with JRE published: %s" ,
                    endpointCalculatorWS.getEndpointReference().toString());
        }
    }

}
```

```
--- exec-maven-plugin:1.5.0:exec (default-cli) @ bottom-up ---
 --- CalculatorWS with JRE published: <?xml version="1.0" encoding="UTF-8" standalone="yes"?><EndpointReference xmlns="http://www.
```

JAX-WS Runtime in Tomcat

Konfigurationsdateien

- web.xml
- sun-jaxws.xml



- **web.xml**

1. Dispatcher Servlet
   com.sun.xml.ws.transport.http.servlet.WSServlet
2. Listener für Servlet Context Events

   com.sun.xml.ws.transport.http.servlet.WSServletContextListener

- **sun-jaxws.xml**
  (DD für JAX-WS Webservices in WARs)

1. Assoziiert Webservice Port mit Web-Kontext URL-Pattern)

Beispiel: Tomcat

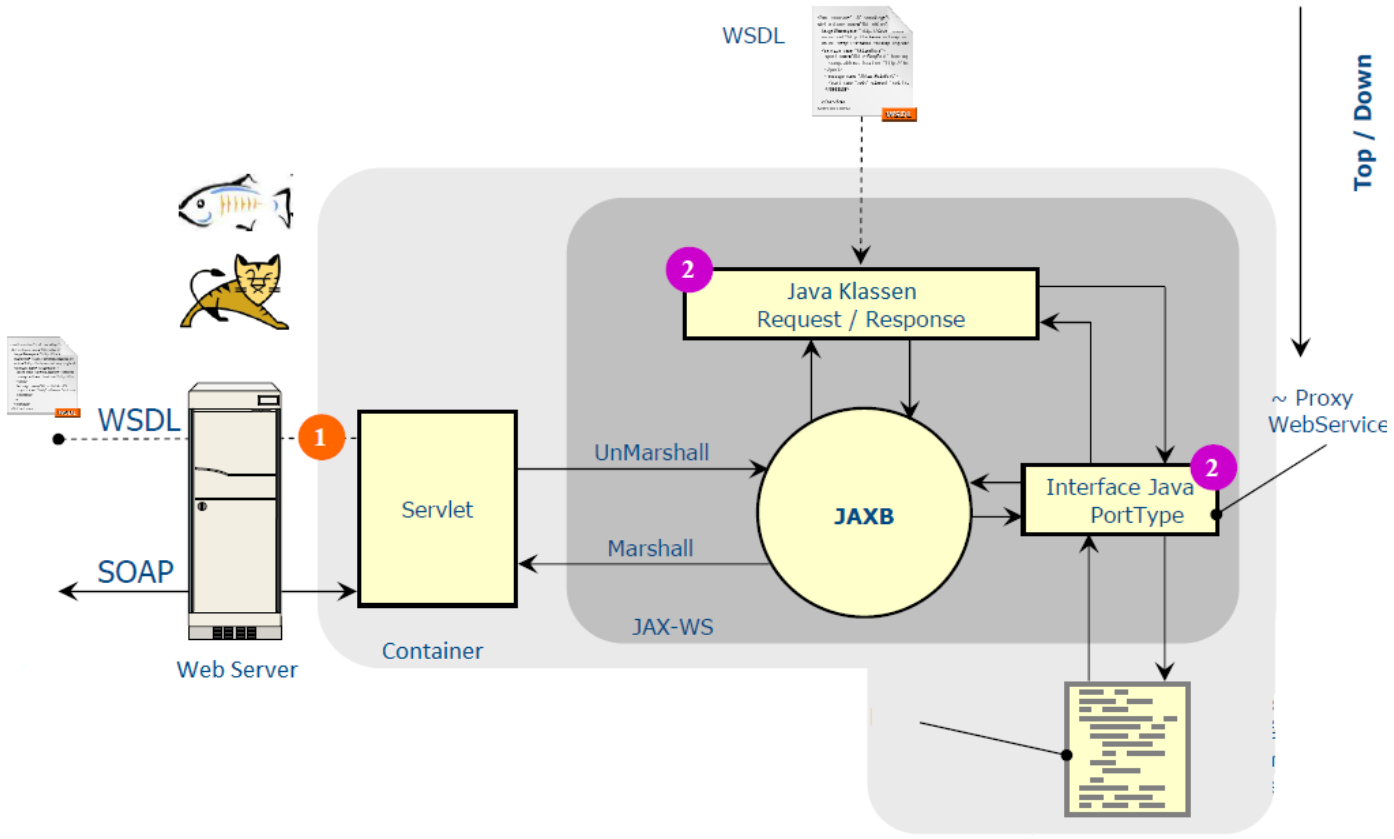
```
<!-- Tomcat WS Artefacts: Listener + servlet class -->
<listener>
    <listener-class>com.sun.xml.ws.transport.http.servlet.WSServletContextListener</listener-class>
</listener>
<servlet>
    <display-name>CalculatorWS</display-name>
    <servlet-name>calculatorWeb</servlet-name>
    <servlet-class>com.sun.xml.ws.transport.http.servlet.WSServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>calculatorWeb</servlet-name>
    <url-pattern>/calculatorWS</url-pattern>
</servlet-mapping>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoints
  xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime"
  version="2.0">
  <endpoint
      name="CalculatorWS"
      implementation="edu.javaee.ws.bottom.up.CalculatorWeb"
      url-pattern="/calculatorWS"/>
</endpoints>
```

Webservices

End Point	Informationen
Servicename\: {http://ws.jaxws.edu}CalculatorWS Portname\: {http://ws.jaxws.edu}CalculatorIF	Adresse: http://localhost:8080/bottom-up/calculatorWS WSDL: http://localhost:8080/bottom-up/calculatorWS?wsdl Implementierungsklasse: edu.javaee.ws.bottom.up.CalculatorWeb

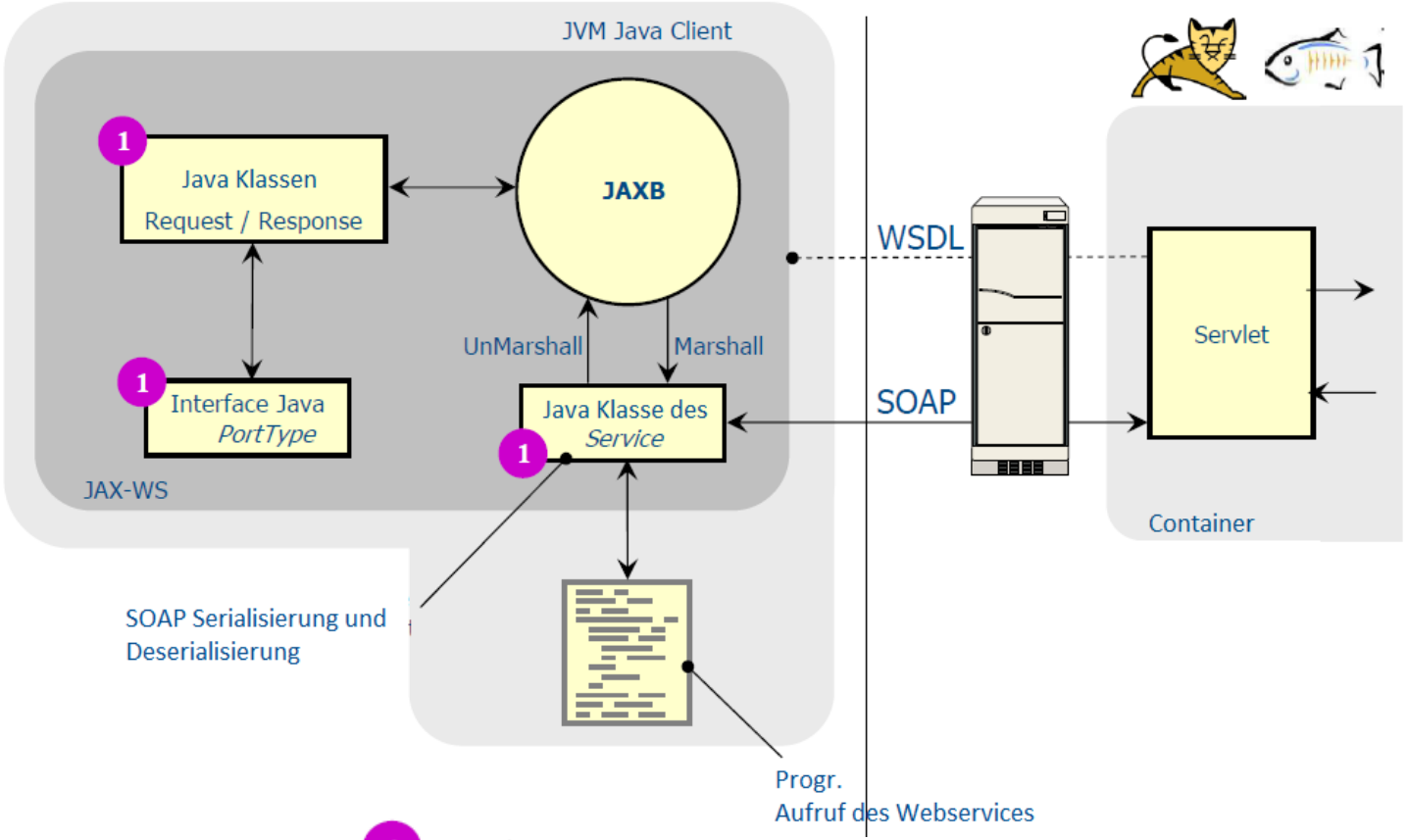
Contract-First Ansatz



1 WSDL implizit mit *wsgen* generiert

2 Explizit mit *wsimport* generiert

Clientseitiges Programmiermodel



Client Schicht 1 Explizit mit *wsimport* generiert

JAX-WS Runtime in JRE 6

- JRE 6 bringt neben der API JAX-WS **Werkzeuge** (wsimport, wsgen):

1. wsgen generiert **JAXB-Klassen (Marshalling, Unmarshalling)** und **WSDL** aus einer JAX-WS Annotierten Klasse!

```
wsgen –cp . <package.Webservice> –keep
wsgen –cp . <package.Webservice> –keep -wsdl
```

2. wsimport generiert **JAXB-Klassen (Marshalling, Unmarshalling)** aus einer WSDL!

```
wsimport –d bin -verbose <WSDL> –keep
wsimport –p <package> <WSDL> –keep
```

JAX-WS Clients Entwickeln

- JAX-WS Clients beliebig: JAVA SE/EE Anwendungen, die

1. Webservice Operationen aufrufen wollen
2. Aufrufe erfolgen entweder synchron oder asynchron!

- Aus der WSDL werden mittels **wsimport** folgende WS-Artefakte benötigt

1. JAXB Klassen
2. PortType und Service-Klasse

- Service-Klasse anschließend instanziieren und Zugriff auf dem Port

Beispiel: JAX-WS Java SE Client (synchron)

```java
/**
 * CalculatorWS Webservice Consumer (JavaSE Client)
 *
 * @author ngj
 */
public class Main {

    public static void main(String[] args) {
        CalculatorWS calculatorWS = new CalculatorWS();
        Calculator port = calculatorWS.getCalculatorIF();
        if (null != port) {
            long sum = port.add(12, 23);
            System.out.printf("--- Sum of (12, 23) is '%d' ---", sum);
        }
    }

}
```

```
------------------------------------------------------------------------
Building jaxws::client 1.0-SNAPSHOT
------------------------------------------------------------------------


--- exec-maven-plugin:1.5.0:exec (default-cli) @ client ---
--- Sum of (12, 23) is '35' ---
```

Asynchrone JAX-WS Clients

- **Schemaanpassungen** (wegen Asynchron) erforderlich!
  JAX-WS Bindungen direkt in der WSDL oder in externer Bindungsdateien.

1. Bindungsdeklaration **enableAsyncMapping** der Bindungsdatei hinzufügen, damit die **Schnittstelle** für den **Dynamic-Proxy-Client** mit **asynchronen Callback- oder Abfragemethoden** generiert wird
2. Synchronen Methoden werden ebenfalls generiert!

- Client muss **wsimport** ein ***Binding Customization File*** mitgeben!

Beispiel 1: Asynchrone Verarbeitung

- *Binding Customization File*

```
<?xml version="1.0" encoding="UTF-8"?>
<bindings
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  wsdlLocation="http://localhost:8080/asynchronesws/personenSucheWS?wsdl"
  xmlns="http://java.sun.com/ns/jaxws"
  version="2.0">
      <enableAsyncMapping>true</enableAsyncMapping>
</bindings>
```

wsimport **-b bindingfile.xml**
http://<host>:<port>/<path>?wsdl

Beispiel 2: Einzelne Bindings

- Sollen nicht alle WS Methoden asynchron aufgerufen werden: ***XPATH***

```
<?xml version="1.0" encoding="UTF-8"?>
<bindings
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 wsdlLocation="http://localhost:8080/asynchronesws/personenSucheWS?wsdl"
 xmlns="http://java.sun.com/ns/jaxws"  version="2.0">
    <node="wsdl:definitions/
              wsdl:portType[@name='<name>']/
              wsdl:operation[@name='<name>']" >
     <enableAsyncMapping>true</enableAsyncMapping>
</bindings>
```

JAX-WS Annotations

FineTuning mit JAXWS

- Verwendete JSR (224, 222, 181 und 250)
- Grundlegende Annotations:

1. @WebService POJO als JAX-WS Webservice
2. @WebMethod Operation parametrisieren
3. @WebParam SOAP-Nachricht parametrisieren
4. @WebResult Ausgangsnachricht parametrisieren
5. @WebFault Fehlernachricht parametrisieren
6. @OneWay Einweg Operation

@Webservice

- Markiert ein Interface oder eine Klasse als Webservice.

```
@Webservice(
        name="Webservice",
        endpointInterface="implementiertes Interface",
        portName="port",
        serviceName="service",
        targetNamespace="Namensraum",
        wsdlLocation="wsdl"
)
```

@SOAPBinding

- Konfiguriert das Schema der Soap Nachricht

```
@SOAPBinding(
        style ="Document oder RPC",     //Bei RPC ist Wrapped zwingend!
        use="Literal oder Encoded",      //Encoded nicht unterstützt!
        parameterStyle="BARE oder WRAPPED"
)
```

Beispiel: @WebService, @SOAPBinding

```
@WebService(
   targetNamespace = "http://duke.example.org", name="AddNumbers")
@SOAPBinding(
  style=SOAPBinding.Style.RPC,
  use=SOAPBinding.Use.LITERAL,
  parameterStyle=ParameterStyle.WRAPPED )
public interface Taschenrechner {  …  }
```

@WebMethod

- Markiert eine Methode als Webservice Operation.
- Methode darf nicht final oder statisch sein
- Muss keine ***java.rmi.RemoteException*** werfen!

```
@WebMethod (
        action="SOAP-Binding Action",
        exclude=true | false,        //Wenn true wird sie nicht gemappt
        operationName="Name"
)
```

@WebParam

- Anpassung von Parametern
- Mode OUT und INOUT erfordern **Holder<E>** Objekte

```
@WebParam (
        name="Name",
        mode="WebParam.IN, OUT, INOUT",   //Default ist WebParam.IN
        partName="wsdl:part Name",     //Nicht für Document/Wrapped
        targetNamespace="Namensraum",
        header=true | false          //Wenn true steht der Parameter im Header
 )
```

Beispiel: @WebParam

```
@WebService
public interface Taschenrechner{
        @WebMethod
        public void addiereUndMultipliziere(
                @WebParam(name="num1") int nummer1,
                @WebParam(name="num2") int nummer2,
                @WebParam(name="add",
                mode = WebParam.Mode.OUT) Holder<Integer> addiert,
                @WebParam(name="multiply",
                mode = WebParam.Mode.OUT) Holder<Integer> multipliziert
        );
}
```

@WebResult

- Anpassung von Rückgabewerten

```
@WebResult (
        name="Name",
        partName="wsdl:part Name",
        targetNamespace="Namensraum",
        header=true | false     //Wenn true steht das Result im Header
)
```

Beispiel: @WebResult

```
@WebService(
targetNamespace = "http://duke.example.org", name="AddNumbers")
@SOAPBinding(style=SOAPBinding.Style.RPC,
use=SOAPBinding.Use.LITERAL)
public interface Taschenrechner{
        @WebMethod(operationName="add", action="urn:addNumbers")
        @WebResult(name="summe")
        public int addiereZahlen(
                @WebParam(name="num1") int nummer1,
                @WebParam(name="num2") int nummer2
        )
}
```

@WebFault

- Weiterreichen von Exceptions

1. Werden im Client als **javax.xml.ws.soap.SOAPFaultException** geworfen
2. RuntimeExceptions und java.rmi.RemoteException werden nicht in der WSDL definiert

```
@WebFault (
        name="Name",
        targetNamespace="Namensraum",
        faultBean="POJO Bean"        //Verwendet zur Kapselung der Exception
)
```

Beispiel: @WebFault (1/2) Serverseitig

- Serverseite:

```
@WebService
public interface Taschenrechner{
        @WebMethod(operationName="add")
        public int addiereZahlen(
                @WebParam(name="num1")int nummer1,
                @WebParam(name="num2")int nummer2)
                throws AddiereZahlenFault;
}
```

```
@WebFault //Annotation auf dem Server optional
public class AddiereZahlenFault extends Exception{
}
```

Beispiel: @WebFault (2/2) Clientseitig

- Exception- und Bean-Klassen werden durch **wsimport** erstellt

```
@WebFault(name="AddiereZahlenFault")
public class AddiereZahlenFault_Exception extends Exception{
    private AddiereZahlenFault faultInfo;
    public AddiereZahlenFault_Exception (String message,
            AddiereZahlenFault faultInfo) {
            super(message);
    }
    …
}
```

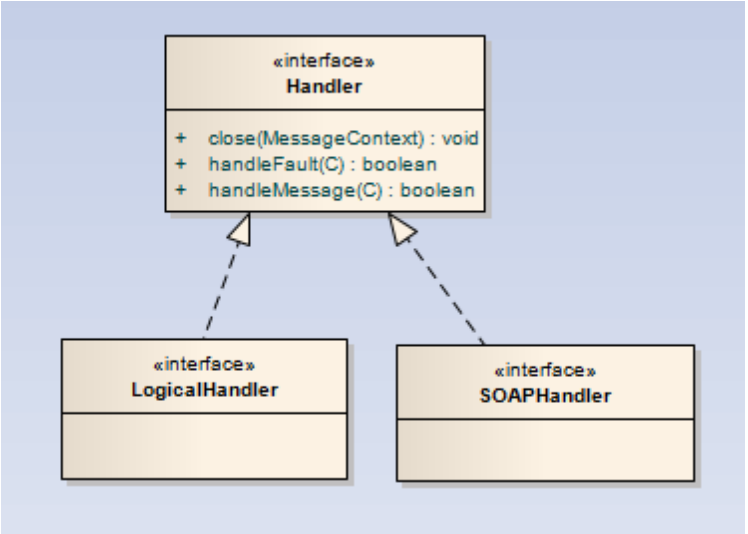
```
public class AddiereZahlenFault {
    protected String message;
    public AddiereZahlenFault() { }
   // Getter + Setter
}
```

Handler

JAX-WS

JAX-WS Handler

- Handler sind **architekturtonische Elemente**, die SOAP Nachrichten auf einer definierten Art und Weise verändern können.
- Je nach Art haben sie Zugriff auf die gesamte Nachricht (Body und/oder Header)



JAX-WS Handler

Handler Framework

SOAP und Logical Handler

- Man unterscheidet zwischen

1. SOAP Handler
   javax.xml.ws.handler.soap.SOAPHandler
   haben über SOAPMessageContext Zugriff auf die gesamte SOAP Nachricht (Header + Body)
2. LogicalHandler
   javax.xml.ws.handler.LogicalHandler
   haben über LogicalMessageContext Zugriff auf die eigentliche Nachricht (Body)

Was sind Handler?

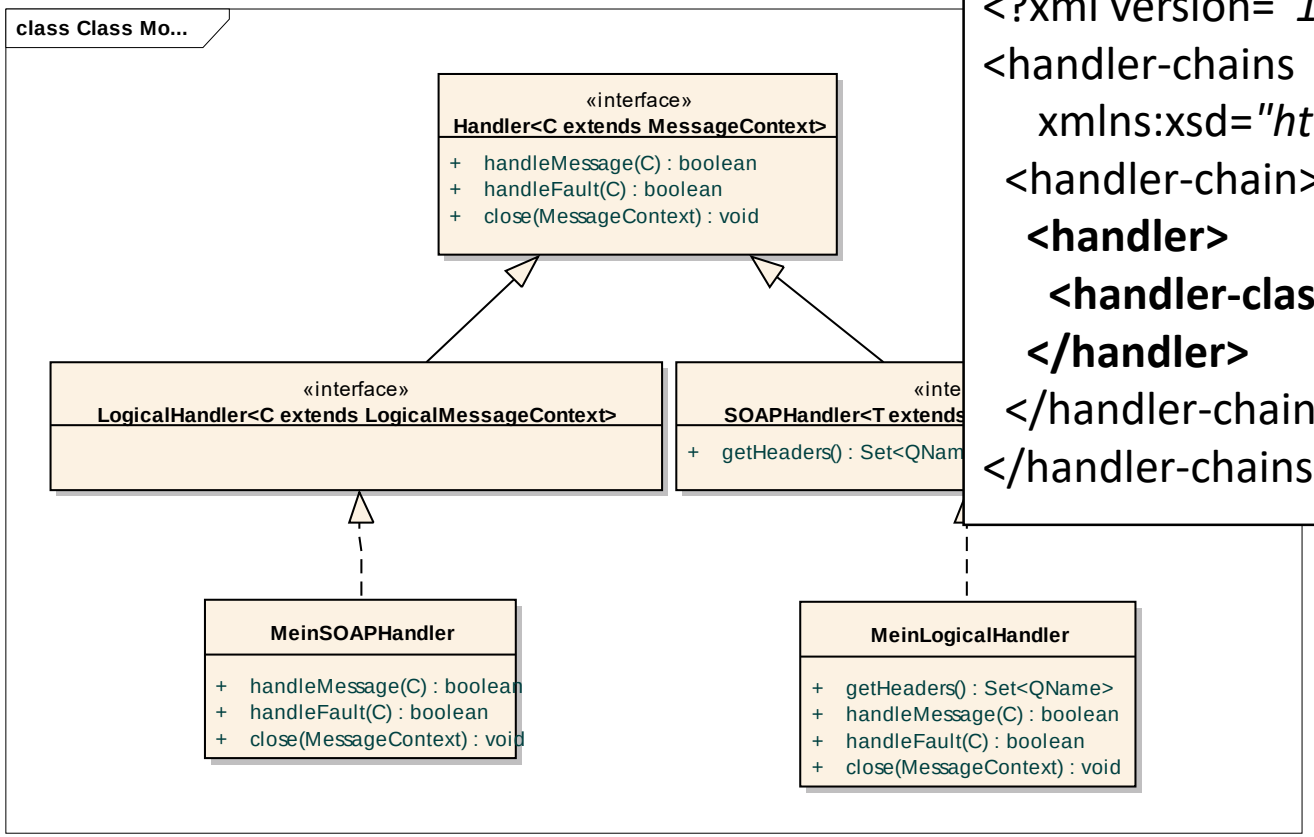
Interzeptoren

- Greifen beim Empfang oder Senden von SOAP-Nachrichten ein!

1. Empfang: Handler werden **vor dem Aufruf** der Webservice Operation aufgerufen.
2. Senden: Handler werden **nach dem Aufruf** der Webservice Operation aufgerufen.

JAX-WS Handler programmieren

Implementierung



handler.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<handler-chains  xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <handler-chain>
  <handler>
   <handler-class>HandlerKlasse</handler-class>
  </handler>
 </handler-chain>
</handler-chains>
```

Beispiel: Logischer Handler

```
public class MeinLogicalHandler implements
        LogicalHandler<LogicalMessageContext> {
    public boolean handleMessage(LogicalMessageContext messageContext) {
            LogicalMessage msg = messageContext.getMessage();
            …
            return true;
    }
}
```

Beispiel: SOAP Handler

```
public class MeinSOAPHandler implements
        SOAPHandler<SOAPMessageContext> {
    public boolean handleMessage(SOAPMessageContext messageContext) {
        SOAPMessage msg = messageContext.getMessage();
        …
        return true;
    }
}
```

JAX-WS Handler im POJO aktivieren

- Serverseitig: @HandlerChain(file= "handler.xml ")
- Clientseitig:

1. @HandlerChain(file= "handler.xml ") und Angabe im wsimport
2. Programmatisch mit BindingProvider

```
@WebService(endpointInterface="... ServerZeitService")
@HandlerChain(file="handler.xml")
public class ServerZeitServiceImpl {
```

```
public class Client {
   public static void main(String[] args) {
   List<Handler> meineHandler = new ArrayList<Handler>();
   meineHandler.add(new MeinSOAPHandler());
    ((BindingProvider)port).getBinding().setHandlerChain(meineHandler);
```

Zusammenfassung 1/2 - Webservice Provider

```
@WebService
public class Calculator {
    public int add (int a, int b) {
        return a+b;
    }
}
```

```
@Stateless @WebService
public class Calculator {
    @Resource WebServiceContext context;
    public int add (int a, int b) {
        return a+b;
    }
}
```

- **JAX-WS Webservice**

1. *POJO (einfacher)*
2. *Stateless Session Bean (Java EE Dienste)*

- Alle **public Methoden** sind Webservice Operationen!
- **Container**

1. *JAVA SE 6 (Embedded Web server)*
2. *Apache Tomcat 6*
3. *Glassfish 3*

- Endpoint.***publish***(String url, Object jaxws)
- *http://server:<port>/<webservice>?wsdl*

Zusammenfassung 2/2 - Webservice Consumer

```
public class CalculatorServlet extends HttpServlet {
   @WebServiceRef(wsdlLocation= "http://localhost:8080/calculator?wsdl ")
   CalculatorService svc;
   protected void doGet(…) throws ServletException, … {
        svc.getCalculatorPort().add(35, 7);
   }
}
```

```
…
CalculatorService svc = new CalculatorService();
Calculator port = svc.getCalculatorPort();
port.add(35, 7);
```

```
@Stateless
public class CalculatorBean {
   @WebServiceRef(CalculatorService.class)
   Calculator port;
   public int mymethod() { return port.add(35, 7);  }
}
```

Ende…

Java REST(HTTP-) Webservices

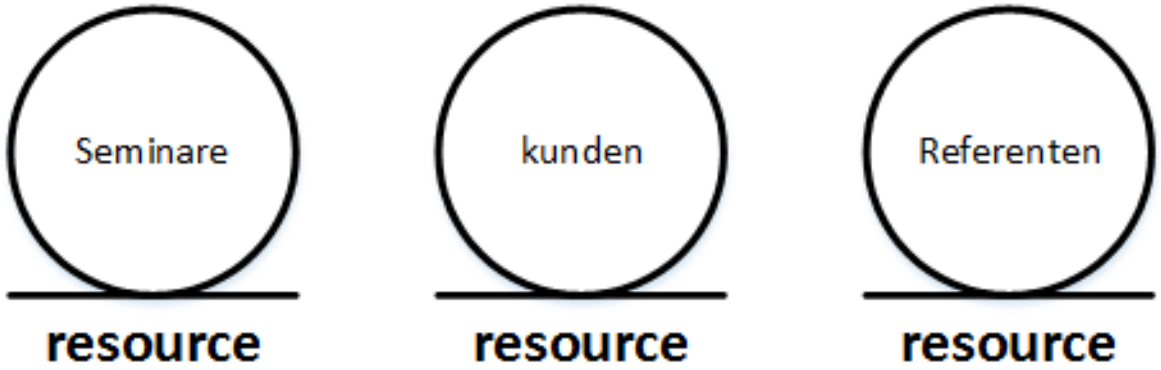
JAXRS

REST?

"Style of software architecture for distributed hypermedia systems"
(Roy T. Fielding)

REST Architekturstil

- Web als System von verteilten Ressourcen
- Jede Ressource eindeutig über eine ID (URI, PK im web) identifizierbar!



- Beispiele:
  http://.../training/**seminare,** .../seminare/**12345**/kunden

(Web-) Ressourcen

- Ressourcen sind über Links erreichbar bzw. manipulierbar

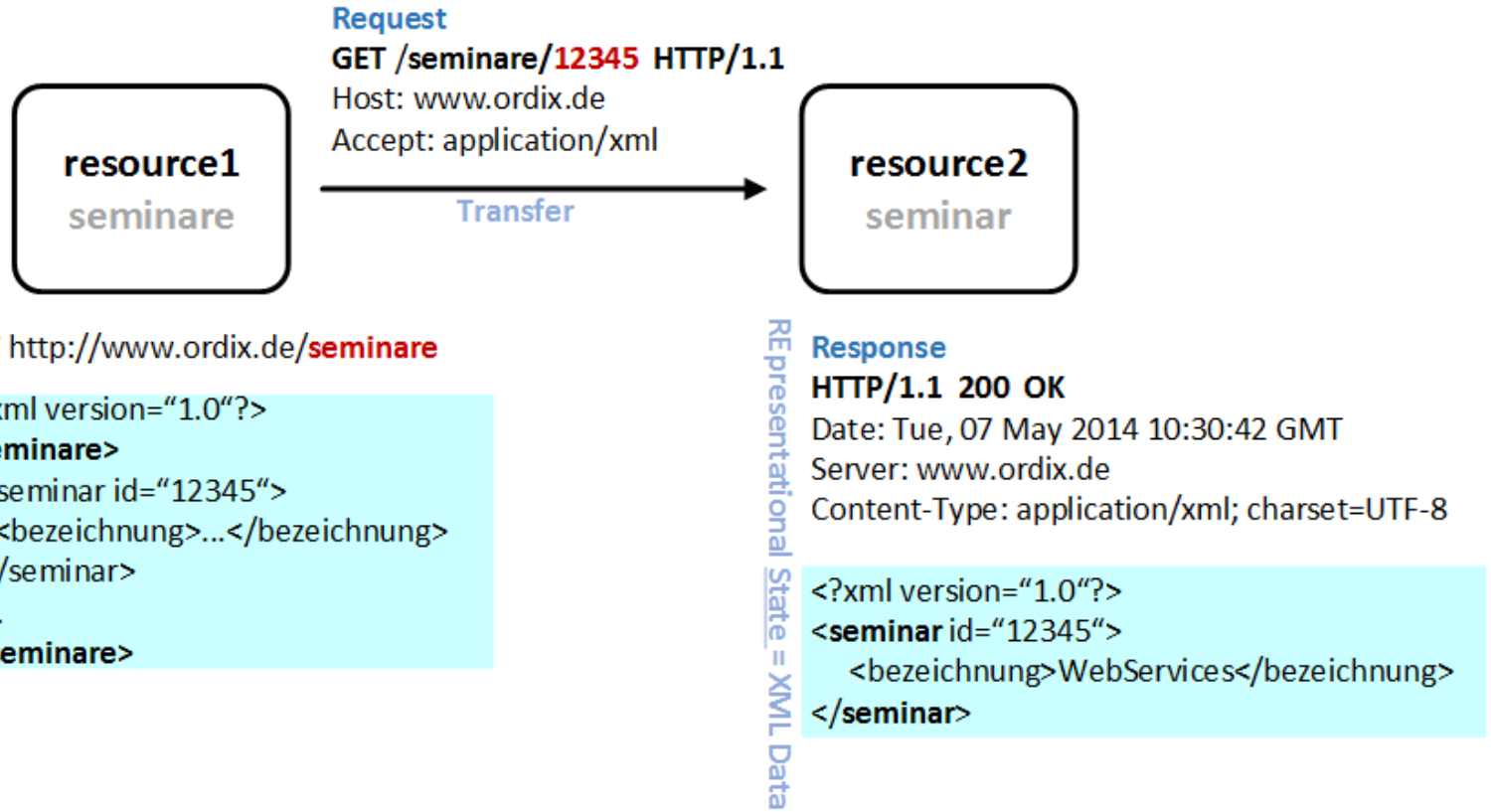
1. Kommunikation erfolgt über Hypertext Transfer Protocol (HTTP)
2. Ressourcen sind Zustandslos
3. Ressourcen verfügen über multiple Repräsentationen

- Link überführt eine Ressource in einem neuen Zustand

1. Ressource erreichen heißt konkret eine **Repräsentation** erhalten!
2. Ressource manipulieren heißt Repräsentation via das **uniform Interface (HTTP-Methoden)** manipulieren!

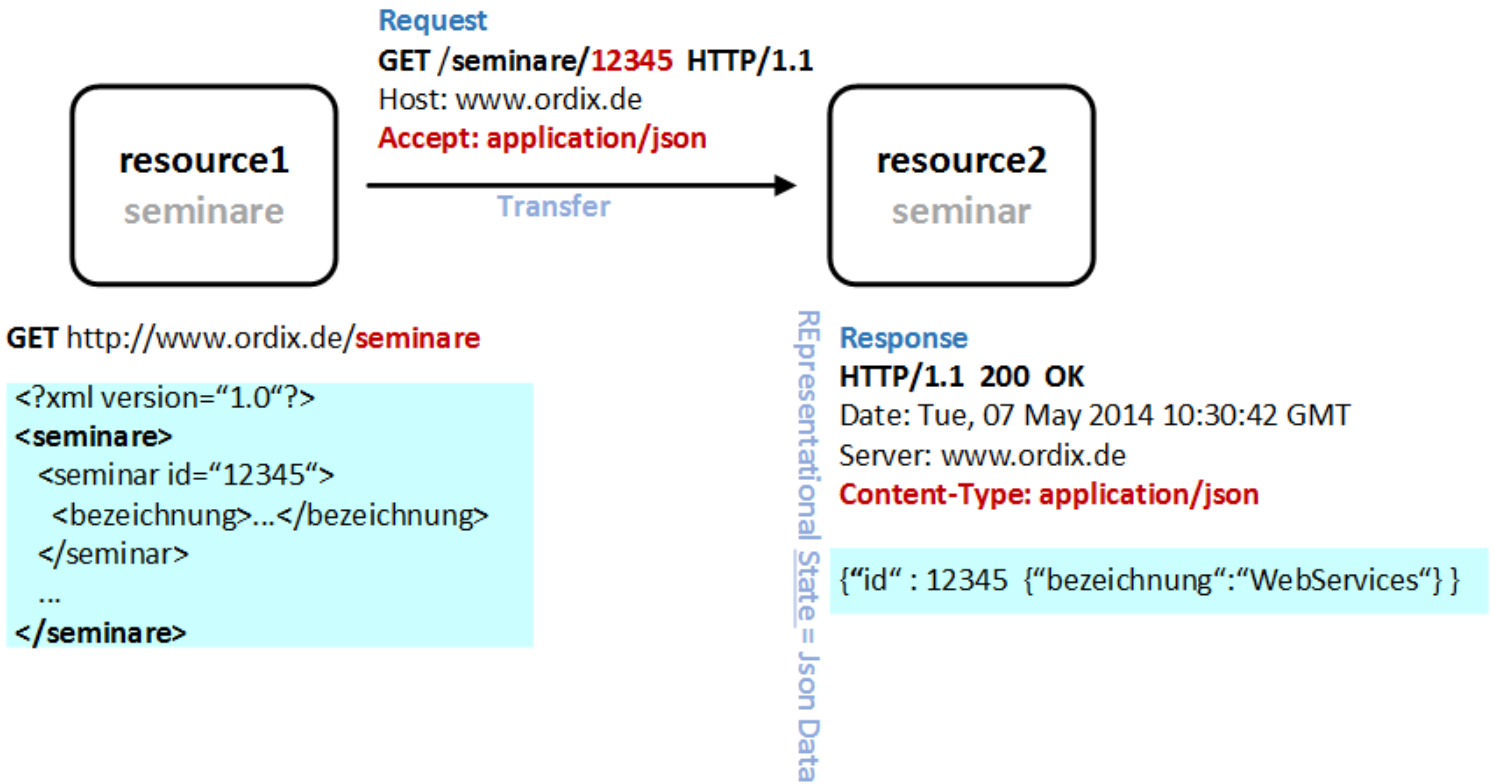
REpresentational State Transfer (REST)

Repräsentation ist der **Zustand** einer Ressource



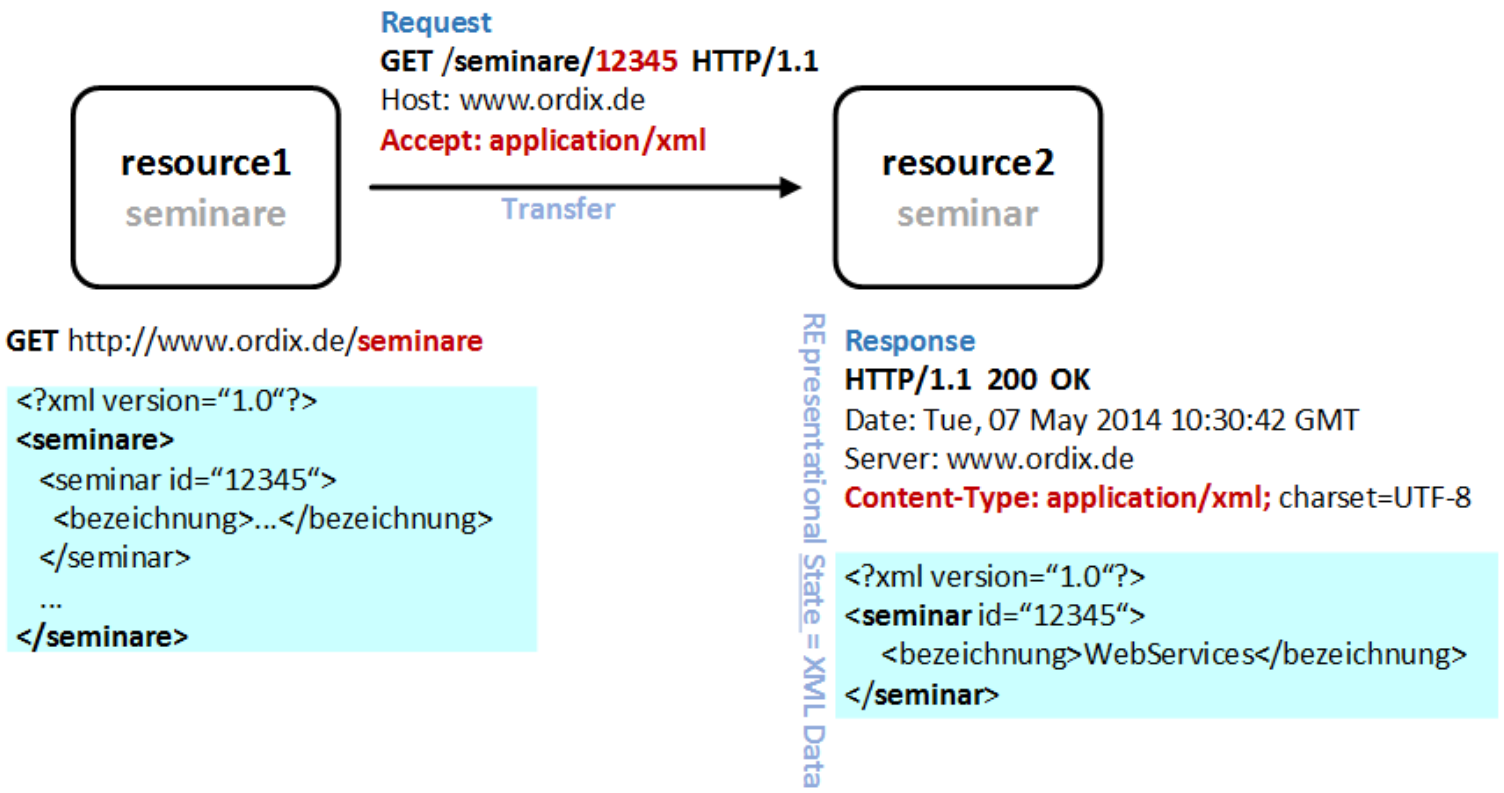
Content Negociation - Json

Multiple Repräsentationen: **Accept Header, Content-Type**



Content Negociation - XML

Multiple Repräsentationen: **Accept Header, Content-Type**



Uniform Interface (CRUD)

- Gemäß Konvention kommt den HTTP-Methoden folgende Bedeutung zu:

HTTP-Methode	Action
GET (Read)	Liefert Ressourcen in einer bestimmten Repräsentation zurück
POST (create)	erzeugt eine Ressource
PUT (update)	aktualisiert eine bereits vorhandene Ressource
DELETE (delete)	löscht eine Ressource

1. **GET** /seminare/12345/Kunden - Kunden im Seminar (ID=12345)
2. **POST** /seminare/12345/Kunden - Kunde zum Seminar hinzufügen

HTTP- Status Code

Successful

HTTP-Methode	Status Code
GET (Read)	**200** - OK Everything is working
POST (create)	**201** - OK New resource has been created
PUT (update)	
DELETE (delete)	**204** - OK resource was successfully deleted

HTTP- Status Code

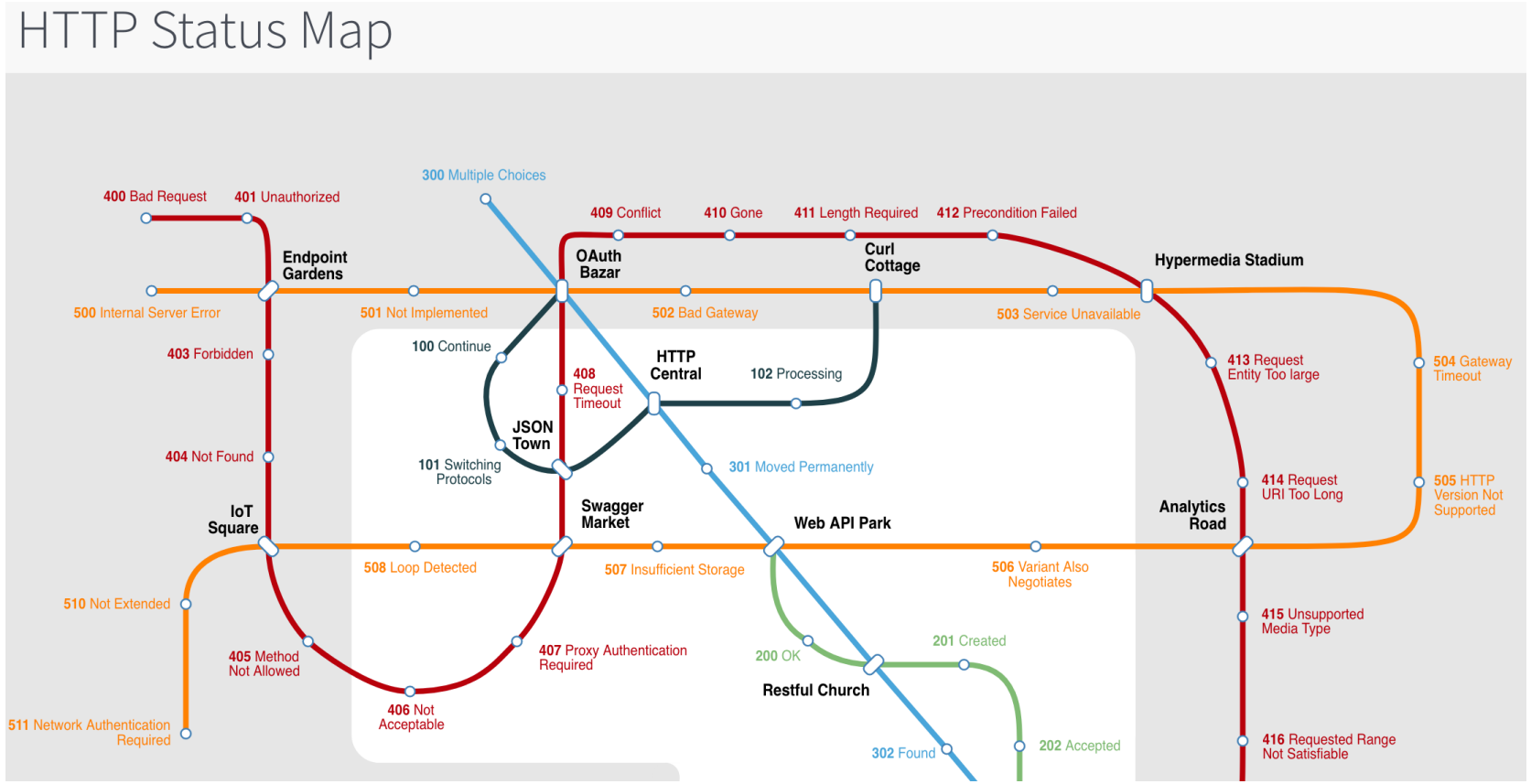
Error

- Status Code statt *Stacktrace* im HTTP-Response

Status Code
400 – Bad Request
401 – Unauthorized
403 – Forbidden
404 – Not found
422 – Unprocessable Entity
500 – Internal Server Error

HTTP Status Codes (Map)

- https://restlet.com/http-status-map/



HATEOAS

Hypermedia **A**s **T**he **E**ngine **O**f **A**pplication **S**tate

- Essentielle **Bedingung** für jede REST-Architektur:
- Hypermedia („Hypertext“ und „Multimedia“), Verallgemeinerung von Text durch Multimedia
- Engine, Zustandsautomat, beschreibt das Verhalten der Anwendung

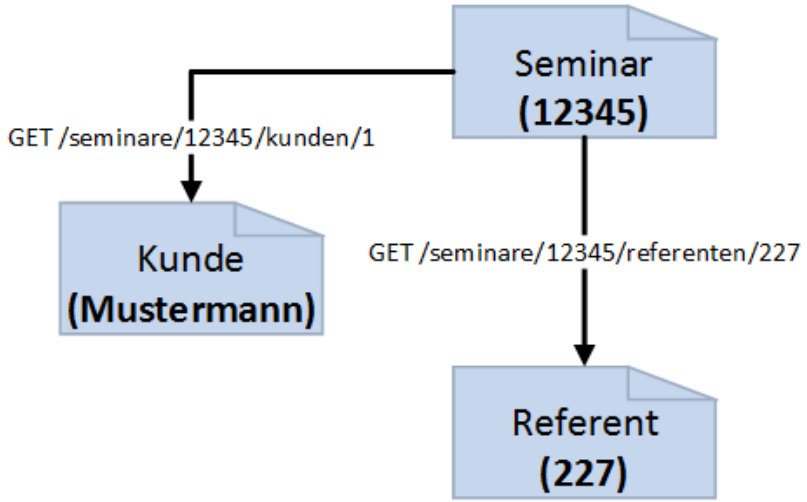
1. Application ist im Restful-Kontext mit einer Ressource gleichzusetzen
2. State ist ein Zustand

- Fazit: Client Aktionen nur durch Folge von **Hyperlinks** möglich!

Beispiel: HATEOAS

- Repräsentationen enthalten Links zu anderen Ressourcen
- Clients verändern den Zustand von Ressourcen über Links

```
<seminar id="12345" self="/seminare/12345">
  <referent ref="/seminare/12345/referenten/227"/>
  <kunden ref="/seminare/12345/kunden" >
     <kunde ref="/seminare/12345/kunden/1" />
</kunden>
</seminar>
```



Kommunikation

Ressourcen-Kommunikation ist zustandslos

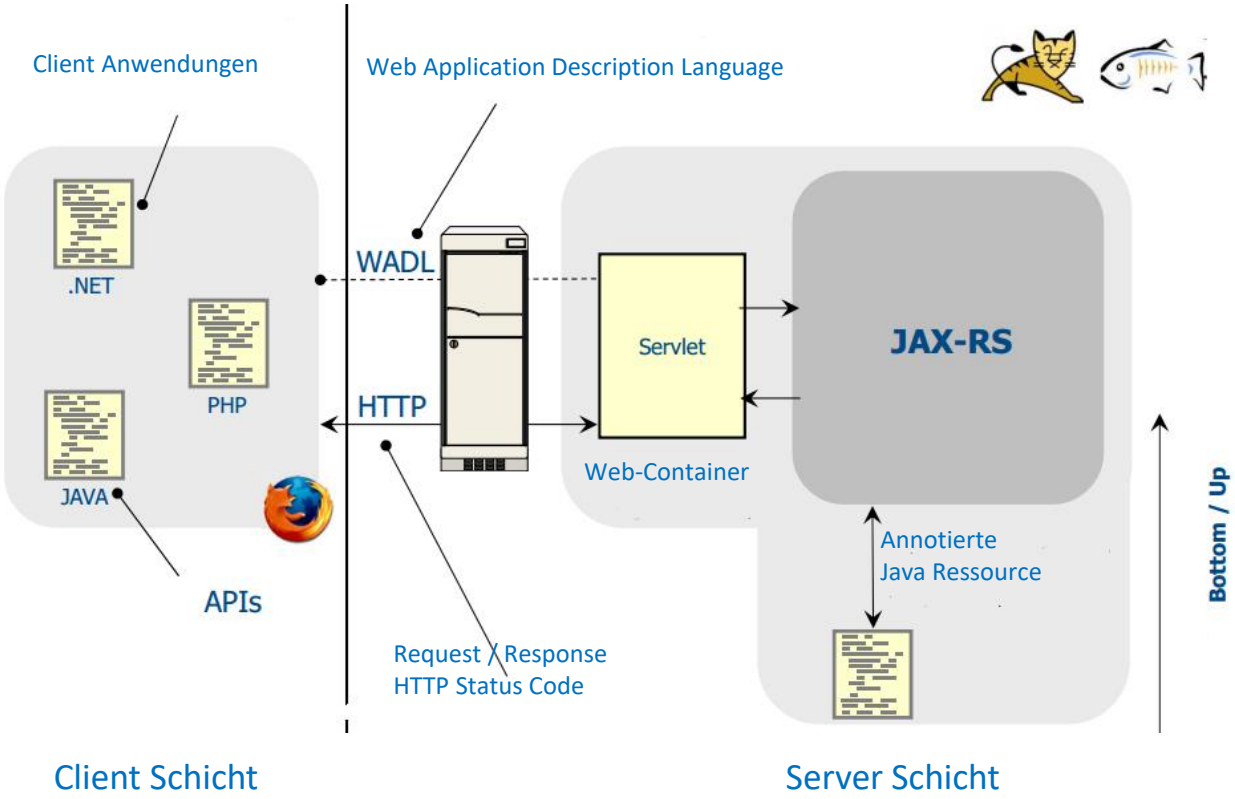
- Basiert auf das **zustandslose HTTP Protokoll**:

1. Request muss immer alle benötigten Daten enthalten
2. Keine Session (Client-Daten) auf dem Server
3. Zustand der Anwendung (Application State) clientseitseitig gehalten
4. Service verantwortlich für die Zustandsänderung einer Ressource

JAX-RS

REST – Jersey Servlet Dispatcher

Web Container (vor JavaEE 6) com.sun.jersey.spi.container.servlet.ServletContainer



Packaging - Servlet 2.x

- Konfiguration der **JAX-RS Servlet**

```
<web-app>
 <servlet>
  <servlet-name>RestServlet</servlet-name>
  <servlet-class>
    com.sun.jersey.spi.container.servlet.ServletContainer
  </servlet-class>
  <init-param>
   <param-name>javax.ws.rs.Application</param-name>
   <param-value>ApplicationSubclassName</param-value>
  </init-param>
 </servlet>
</web-app>
```

web.xml - Servlet 3.x

Packaging Restful Applikation

```
<web-app>
 <servlet>
  <servlet-name>RestServlet</servlet-name>
  <init-param>
   <param-name>javax.ws.rs.Application</param-name>
   <param-value>ApplicationSubclassName</param-value>
  </init-param>
 </servlet>
 <servlet-mapping>
  <servlet-name>RestServlet</servlet-name>
  <url-pattern>/*</url-pattern>
 </servlet-mapping>
</web-app>
```

REST Service - Ressource

- POJO mit @Path, kein Interface benötigt
- *http://HostName/contextPath/servletURI/resourceURI*

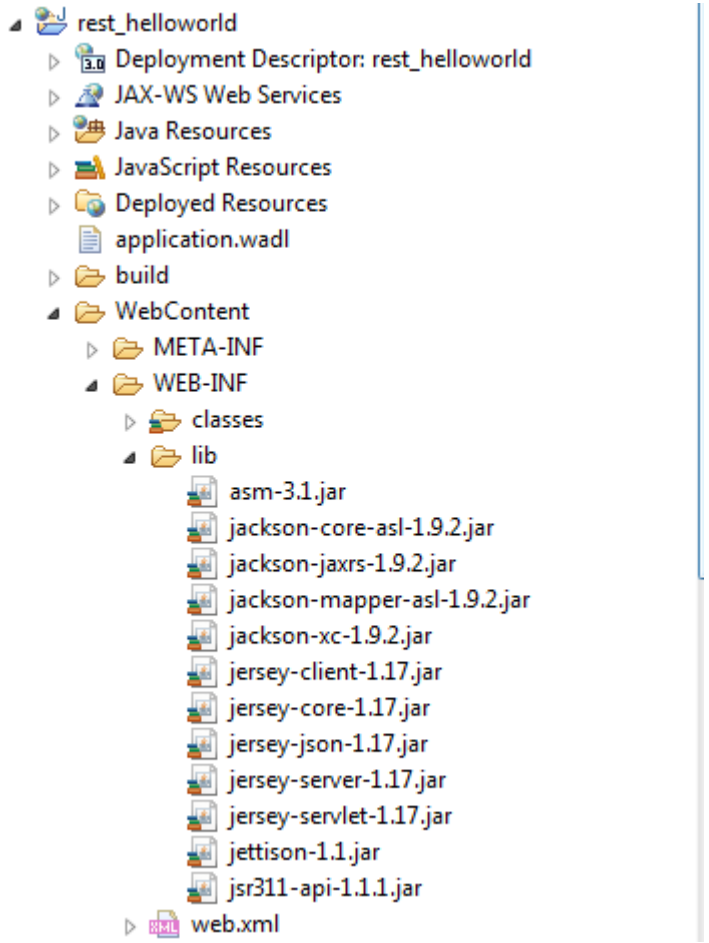
```
@Path("/hello") //path zum RESTful service
public class HelloWorldResource {
  @GET
  @Produces("text/plain")
  public String getHelloWorld() {
    return "Hello World from text/plain";
  }
}
```

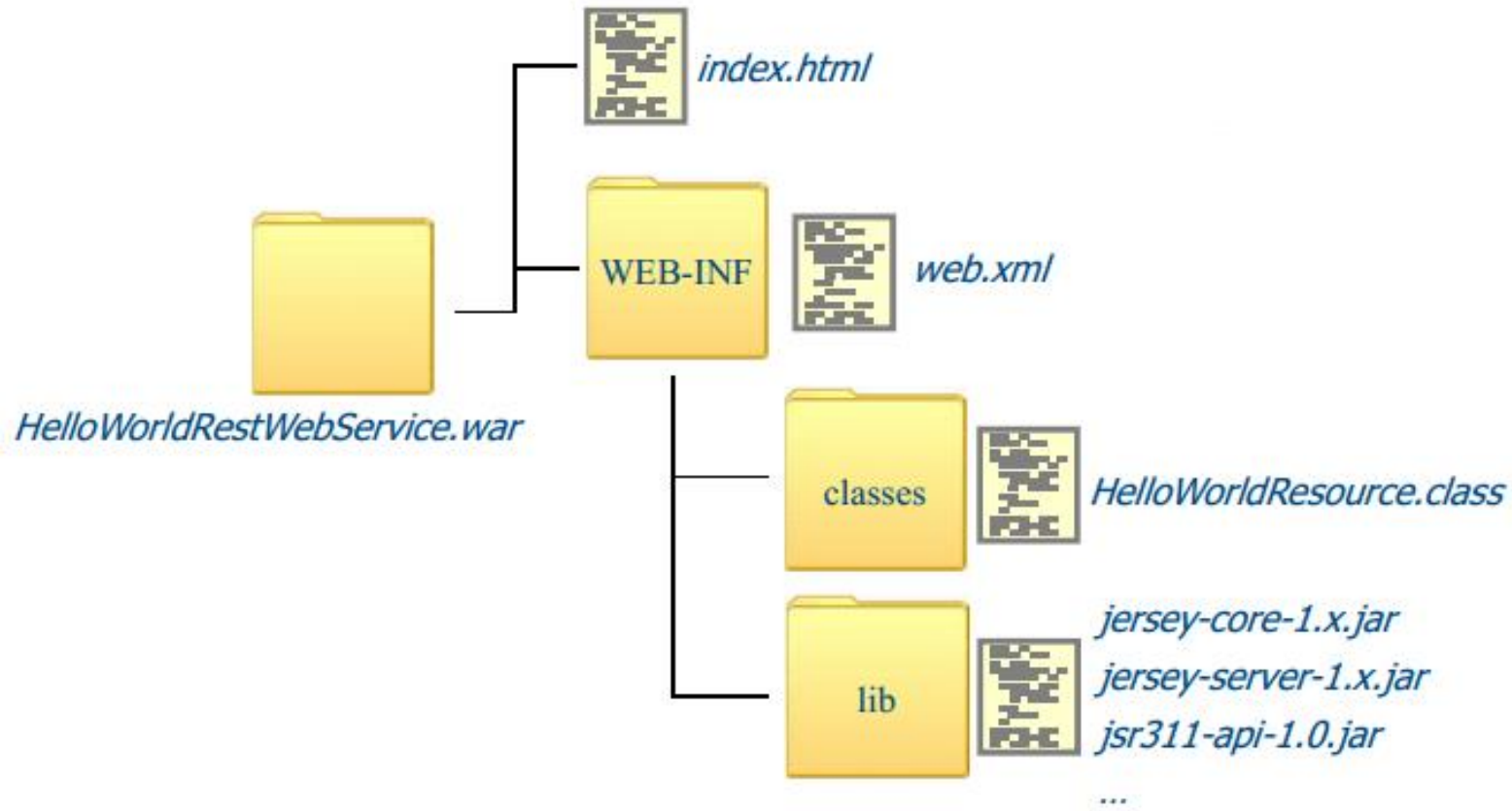
Base URI Pattern

- http://HostName/contextPath/servletURI/**resources/**resourceURI

```
import javax.ws.rs.core.Application;
Import javax.ws.rs.ApplicationPath;
@ApplicationPath("resources")
public class MyApplication extends Application {
  public Set<Class<?>> getClasses() {
    Set<Class<?>> s = new HashSet<Class<?>>();
    s.add(HelloWorldResource.class); //optional – selecting Rest Resources otherwise all
    return s;
  }
}
```

Projektstruktur





Client API - JAX-RS 1.x

- Nicht standardisierte Client API (Jersey)

HelloworldClient.java

```
package de.ordix.schulung.webservices.rest.consumer;

import javax.ws.rs.core.MediaType;

public class HelloworldClient {

    public static void main(String[] args) {

        Client client = new Client();
        WebResource webservice = client.resource("http://localhost:8080/rest_helloworld");

        // String outputText =
        // webservice.path("/hello").accept(MediaType.TEXT_PLAIN).get(String.class);
        String outputText = webservice.path("/hello/html")
                .queryParam("name", "Mustermann").accept(MediaType.TEXT_HTML)
                .get(String.class);
        // String outputText =
        // webservice.path("/hello/Mustermann").accept(MediaType.TEXT_PLAIN).get(String.class);

        System.out.println(outputText);
    }
}
```

Markers | Properties | Servers | Data Source Explorer | Snippets | Problems | Console | Search

```
<terminated> HelloworldClient [Java Application] C:\Program Files (x86)\Java\jdk1.6.0_21\jre\bin\javaw.exe (03.02.2013 16:46:03)
<html><title>HelloWorld</title><body><h2>Html: Hello Mustermann</h2></body></html>
```

Basis Annotationen

API um Request-Parameter zu verwalten

Annotation *javax.ws.rs.**	**Beschreibung**
@PathParam	Werte aus URI template Parameter extrahieren
@MatrixParam	Werte aus URI's matrix Parameter extrahieren
@QueryParam	Werte aus URI's Query Parameter extrahieren
@FormParam	Werte aus Formular POST Data extrahieren
@HeaderParam	Werte aus HTTP request headers extrahieren
@CookieParam	Werte aus HTTP cookies extrahieren (Client gesetzt)
@Context	Werte aus Kontext lesen

- Default Werte können mittels **@DefaultValue** angegeben werden

JAX-RS Annotation - @Path

- POJO muss mit **@Path** annotiert werden

1. Bezeichnet als **Resource** bzw. ***Root Resource Class***
2. URI Pattern: http://<host>:<port>/Context/ApplicationPath/**Resource**

```
@Path("/buecher")
public class BookResource {
  @GET public List<String> alleBuecher(){…}
  @GET @Path("/ausleihe")
  public List<String> ausleihe() {…}
}
```

Template Parameter - @PathParam

Selektion über **Template Parameter**

- @Path begrenzt auf konstante Ausdrücke z.B. http://..../books/borrowed

- Variable und komplexe Ausdrücke sind **Template Parameter**:

1. Template Parameter werden an **@Path** mit Hilfe von **{…}** angegeben
2. Template Parameter nehmen auch reguläre Ausdrücke!
3. Platzhalter Angabe über **@PathParam("{Template}")**

Beispiel 1: Template Parameter

```
@Path("/books")
public class BookResource {
 @GET @Path("{id}")
 public String getBookById(@PathParam("id") int id) {
   return "Rest Service";
 }
 @GET @Path("name-{name}-editor-{editor}")
 public String getBookByNameAndEditor(
   @PathParam("name")String name, @PathParam("editor")String editor)
   return "Rest API(Name:"+ name +" - Editor:"+ editor +")";
 }
}
```

Beispiel 2: Template Parameter

```
@Path("/books")
public class BookResource {
  @Inject BookService service;
  @GET @Path("{id : .+}/editor")
  public String getBookEditorById(@PathParam("id") String id) {
    return service.findBook(id).getTitle();
  }
  @GET @Path("original/{id : .+}")
  public String getOriginalBookById(@PathParam("id")String id) {
    return service.findBook(id).getTitle();
  }
}
```

GET http://..../books/**1234/path1/pathx/editor**

GET http://..../books/**original/1234/path1/pathx**

@QueryParam

- Parameter aus einem Request lesen

GET /books/**queryparameters?name**=harry**&isbn**=1-111111-11**&isExtended**=true

```
@Path("/books")
public class BookResource {
 @GET @Path("queryparameters")
 public String getQueryParameterBook(
   @DefaultValue("all") @QueryParam("name") String name,
   @DefaultValue("?-???????-?") @QueryParam("isbn") String isbn,
   @DefaultValue("false") @QueryParam("isExtended") boolean isExtented){
   return name + " " + isbn + " " + isExtented;
 }
}
```

@FormParam

- Parameter aus **einem Formular** lesen
- **application/x-www-form-urlencoded** um Parameter aus POST zu lesen

```
@Path("/books")
public class BookResource {
  @POST
  @Consumes("application/x-www-form-urlencoded")
  public String createBookFromForm(@FormParam("name") String name){
    System.out.println("BookResource.createBookFromForm()");
    return name;
  }
}
```

@HeaderParam

- Parameter aus dem **Header** eines Request

```
@Path("/books")
public class BookResource {
 @GET
 public String getHeaderParameterBook(
  @DefaultValue("all") @HeaderParam("name") String name,
  @DefaultValue("?-???????-?") @HeaderParam("isbn") String isbn,
  @DefaultValue("false") @HeaderParam("isExtended") boolean isExtented) {
  return name + " " + isbn + " " + isExtented;
 }
}
```

Context Parameter

- Zu **Kontext-Informationen** gehören:

1. **UriInfo** : URIs Metadaten
2. **Request** : Request Metadaten
3. **HttpHeaders** : Header Daten
4. **SecurityContext** : Sicherheitsrelevanten Daten (Principal)

@Context / UriInfo

- **Metadaten** aus einem Request

```
@Path("/infos")
public class InfosResource {
  @GET @Path("{name}")
  public String infos(@Context UriInfo uriInfo,@PathParam("name")String name) {
    String path = uriInfo.getPath();
    List<PathSegment> pathSegments = uriInfo.getPathSegments();
    MultivaluedMap<String, String> pathParameters = uriInfo.getPathParameters();
    MultivaluedMap<String, String> queryParameters = uriInfo.getQueryParameters();
    String requestUri = uriInfo.getRequestUri();
    ...
```

@Context / HttpHeaders

- **Headerinformationen** aus einem Request

```
@Path("/books")
public class BookResource {
  @GET @Path("informationfromhttpheaders/{name}")
  public String headers(@Context HttpHeaders httpheaders) {
     Map<String,Cookie> cookies = httpheaders.getCookies();
     Set<String> cookiesKeySet = cookies.keySet();
     MultivaluedMap<String,String> requestHeaders = httpheaders.getRequestHeaders();
     Set<String> requestHeadersSet = requestHeaders.keySet();
     return "";
   }
}
```

Sub-resource locator

- Ressourcen können über übergeordnete Ressourcen weitergegeben werden!

- Solche Ressource sind **Sub-resource Locator**

1. Sub-resource Locator sind immer **Methoden** mit **@Path** annotiert **aber ohne @GET, @POST, @PUT, @DELETE!**
2. **Rückgabentyp** ist immer eine Resource vom Typ **Object**

- Vorteil ist hier die **Polymorphie**

Beispiel: Sub-resource Locator

```
@Path("/books")
public class BookResource {
  @Path("specific")
  public SpecificBookResource getSpecificBook() {
    return new SpecificBookResource();
  }
}
```

GET http://..../**books/specific/1234**

```
public class SpecificBookResource {
  @Inject BookService service;
  @GET @Path("{id}")
  public String getSpecificBookTitleById(@PathParam("id") int id) {
    return service.findBook(id).getTitle();
  }
}
```

MIME Typ - @Consumes, @Produces

```
@Path("/books")
public class BookResource {
  @GET @Path("details/{id}")
  @Produces(MediaType.TEXT_PLAIN)
  public String getDetailTextBookId(@PathParam("id") String id) {
    return "Ce livre est une introduction sur la vie";
  }
  @PUT @Consumes("application/xml")
  public void updateBookWithJAXBXML(Book current){ //Book = JAXB RootElement
  …
  }
```

Klasse Response

- JAX-RS ermöglicht der Aufbau von komplexen Response mit **Header, Status Code und URI**

HTTP/1.1 **200 OK**
Date: Wed, 14 November 2013 14:44:55 GMT
Server: Jetty(6.1.14)
Content-Type: text/html
…

- JAX-RS bietet dafür die Klasse **Response** mit abstrakten Methoden:

1. Object getEntity() : **Body**
2. int getStatus() : **Status Code**
3. MultivalueMap<String, Object> getMetaData() : **Header Informationen**

ResponseBuilder (1/2)

- *Builder* werden aus dem **Response Objekt** generiert:

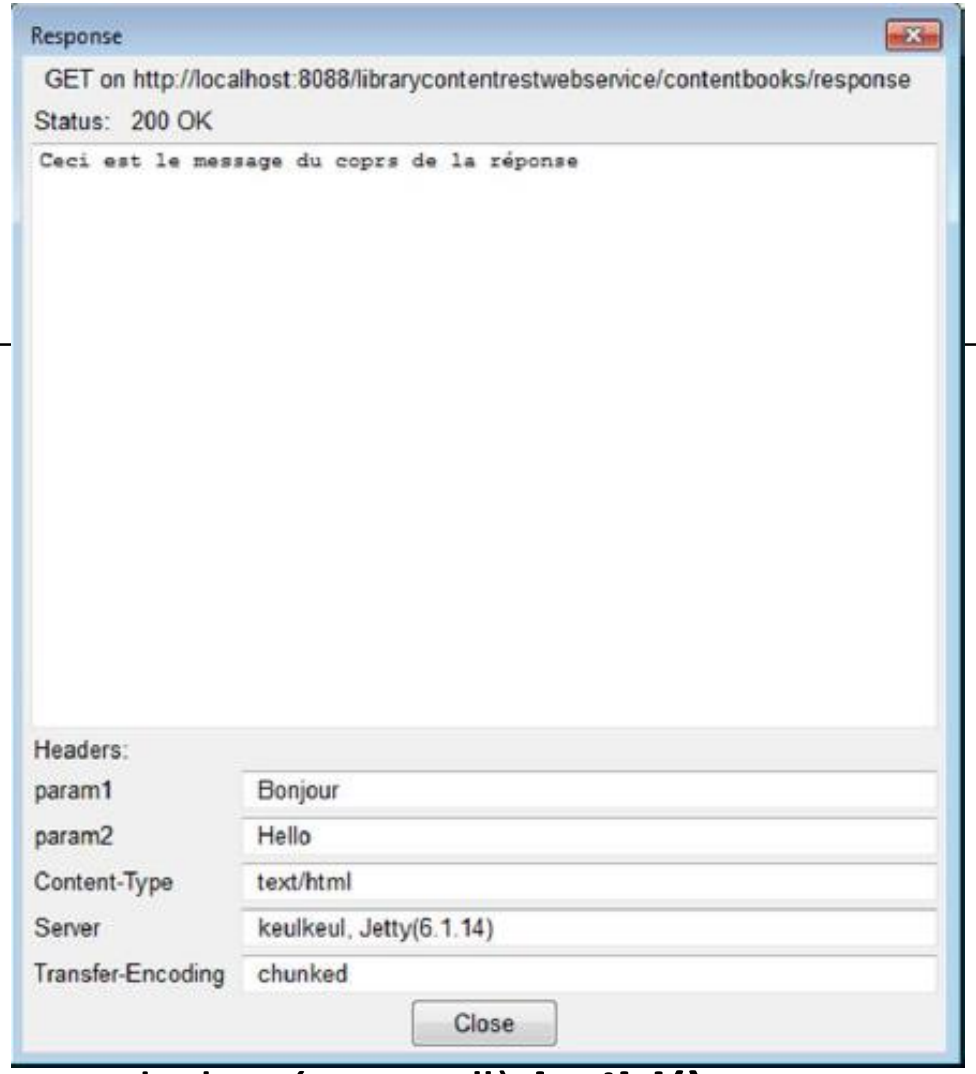
1. ResponseBuilder **created(URI)**
   ändert den Wert der *location* im Header für neu erzeugten Resourcen
2. ResponseBuilder **notModified()** Status auf « Not Modified »
3. ResponseBuilder **ok()** Status auf « Ok »
4. ResponseBuilder **serverError()** Status auf « Server Error »
5. ResponseBuilder **status(Response.Status)** definiert ein Status

ResponseBuilder (2/2)

- Methoden um **Response Objekte zu verändern**:

1. Response **build()** : erzeugt eine neue Instanz
2. ResponseBuilder **entity(Object value)** : ändert den Body
3. ResponseBuilder **header(String, Object)** : ändert ein Header-Parameter

Beispiel 1: Response




```
@Path("contentbooks")
public class BookResource {
  @Path("response")
  @GET
  public Response getBooks(){
   return Response.status(Response.Status.OK)
                  .header("param1", "Bonjour")
                   .header("param2", "Hello")
                  .header("server", "keulkeul")
                  .entity("ceci est le message du corps de la réponse").build();
  }
}
```

Beispiel 2: Response

```
@Path("/contentbooks")
public class BookResource {
  @GET
  @Path("response")
  public Response getBooks() {
          return Response.serverError().build();
  }
}
```

Beispiel 3: Response

```
@Path("/contentbooks")
public class BookResource {
 @POST @Path("response")
 @Consumes("application/xml")
 public Response createBooks(Book newBook, @Context UriInfo uri) {
   URI absolutePath = uri.getAbsolutePath();
   return Response.created(absolutePath).build();
 }
}
```

Client API - JAX-RS 2.0

- **Standardisierte Client API** für REST Services Consumer

```
final String REST_URL = "http://<host>:<port>/restservices/resources";
Client client = ClientBuilder.newClient();
WebTarget target = client.target(REST_URL);
Invocation invocation = target.request(MediaType.TEXT_PLAIN).buildGet();
Response response = invocation.invoke();
//fluent Api
Response response = ClientBuilder.newClient().target(REST_URL)
                                .request(MediaType.TEXT_PLAIN).get();
String body = ClientBuilder.newClient().target(REST_URL).request() .get(String.class);
```

JAX-RS Async Client

- Asynchrone Aufrufe möglich

```
Future<String> future =
  ClientBuilder.newClient()
            .target("http://www.foo.com/book")
            .request()
            .async()
            .get(String.class);
  try {
    String body = future.get(1, TimeUnit.MINUTES);
  } catch (InterruptedException | ExecutionException e) {...}
```

JAX-RS Async Server

- Asynchrone Request Verarbeitung auf dem Server

```
@Path("/async")
public class AsyncResource {
  @GET
  public void asyncGet(@Suspended AsyncResponse asyncResp) {
     new Thread(new Runnable() {
                    public void run() {
                                String result = veryExpensiveOperation();
                                asyncResp.resume(result);
                    }
          }).start();
  }
}
```

JAX-RS Filter

- Filters on client side

1. ClientRequestFilter
2. ClientResponseFilter

- Filters on server side

1. ContainerRequestFilter
2. ContainerResponseFilter

Ende…

Sicherheit

Webservices

Sicherheit?

- Definition

Sicherheit (lateinisch *sēcūritās* zurückgehend auf *sēcūrus* „sorglos“, aus *sēd* „ohne“ und *cūra* „(Für-)Sorge“) bezeichnet einen Zustand, der frei von unvertretbaren Risiken ist oder als gefahrenfrei angesehen wird.
wikipedia

Authentifizierung?

- Definition

Authentifizierung (griechisch authentikós ‚echt‘,) ist der **Nachweis (Verifizierung)** einer behaupteten Eigenschaft einer Entität, die beispielsweise ein Mensch, ein Gerät, ein Dokument oder eine Information sein kann, und die dabei durch ihren Beitrag ihre Authentisierung durchführt.
Das zugehörige Verb lautet **authentifizieren** (englisch: *authenticate*), das für das Bezeugen der Echtheit von etwas steht
wikipedia

Autorisierung?

- Definition

Autorisierung (lateinisch auctorizare, „bestätigen, beglaubigen“) ist im weitesten Sinne eine **Zustimmung oder Erlaubnis**, spezieller die Einräumung von Rechten gegenüber interessierten Rechtssubjekten, gegebenenfalls als Nutzungsrecht gegenüber Dritten.
wikipedia

Fazit: Sicherheit

Authentication:
Verifying that a person is (or at least appears to be) a specific user, since he/she has correctly provided their security credentials (password, answers to security questions, fingerprint scan, etc.).

Authorization:
Confirming that a particular user has access to a specific resource or is granted permission to perform a particular action.

Stated another way, *authentication* is knowing who an entity is, while *authorization* is knowing what a given entity can do.
owasp

Sicherheitsprobleme?

- https://www.toptal.com/security/10-most-common-web-security-vulnerabilities

- **Open Web Application Security Project (OWASP)**

1. Non-Profit-Organisation mit dem Ziel, die Sicherheit von Anwendungen und Diensten im **World Wide Web** zu verbessern
2. Top 10 Angriffe im WEB: https://www.owasp.org/index.php/Top10

Patterns gegen Angriffsmuster

- **Validierung**
  Ursache für viele Sicherheitsprobleme ist meistens die fehlende Eingabevalidierung!
- **HTTP Basic Authentication**

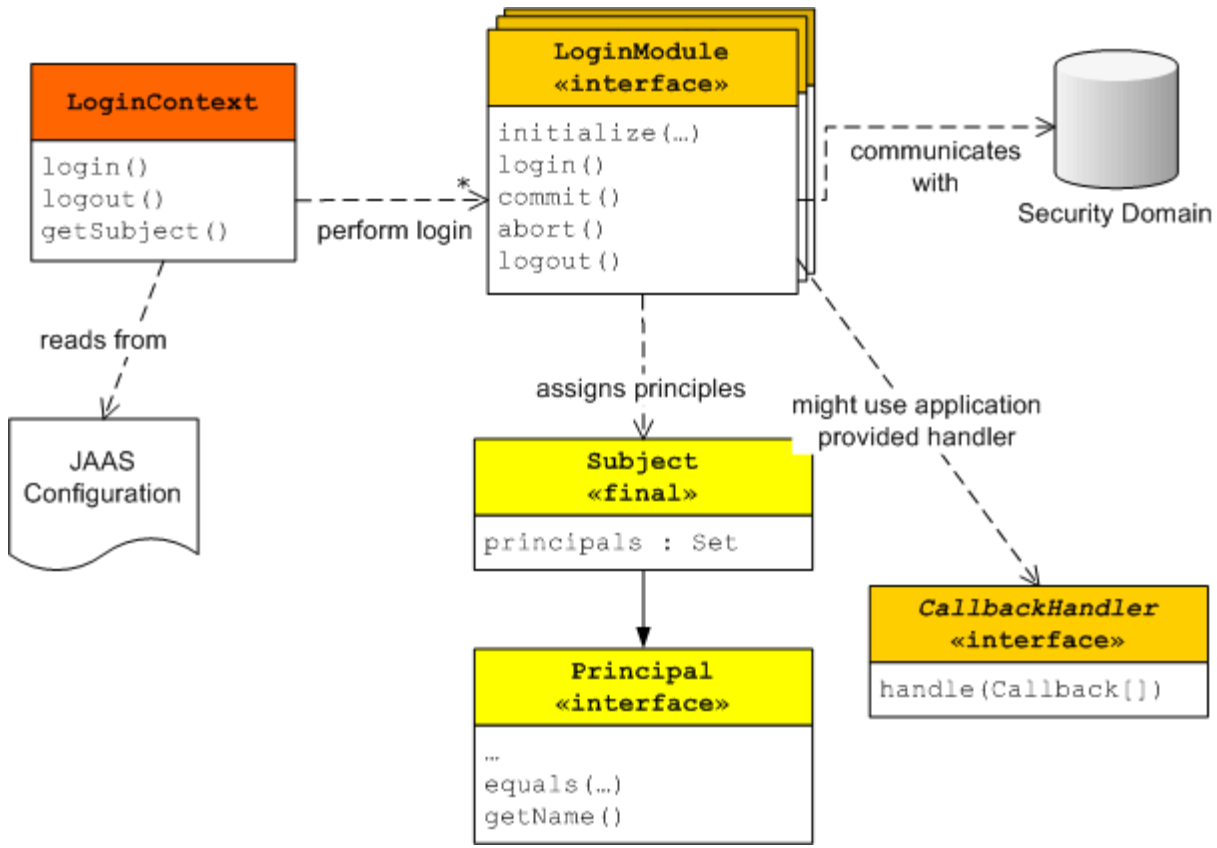
1. Authentifizierung, **Ermittlung der Identität**
2. Autorisierung, Definition von **Sicherheitsrollen**
3. SSL/TLS

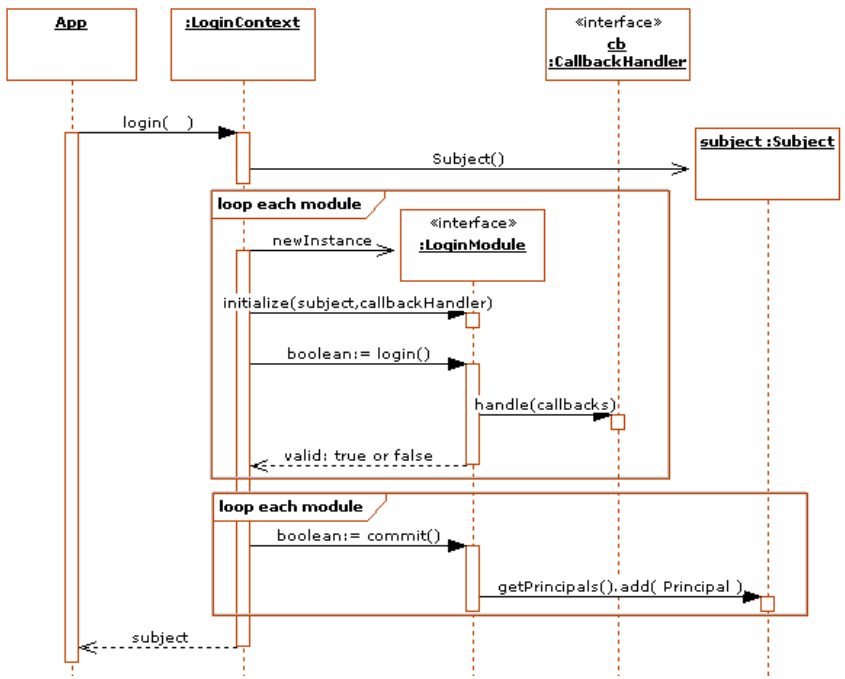
- Gute Ideen, aber sind diese Lösungen noch zeitgemäß (Cloud, …)?

Java Security API - JAAS

Java *Authentication* and *Authorization* Service (**JAAS**)
Java-API, welches es ermöglicht, Dienste zur **Authentifizierung** und **Zugriffsrechte** in Java-Programmen bereitzustellen.
[wikipedia]

JAAS - Authentication





JACC - JAAS in Container

- *Java Authorization Contract for Containers (JACC)*

1. Definiert Sicherheitsrollen als Sammlung von Java Permissions
2. Konzept zur Überprüfung von Sicherheitsrollen

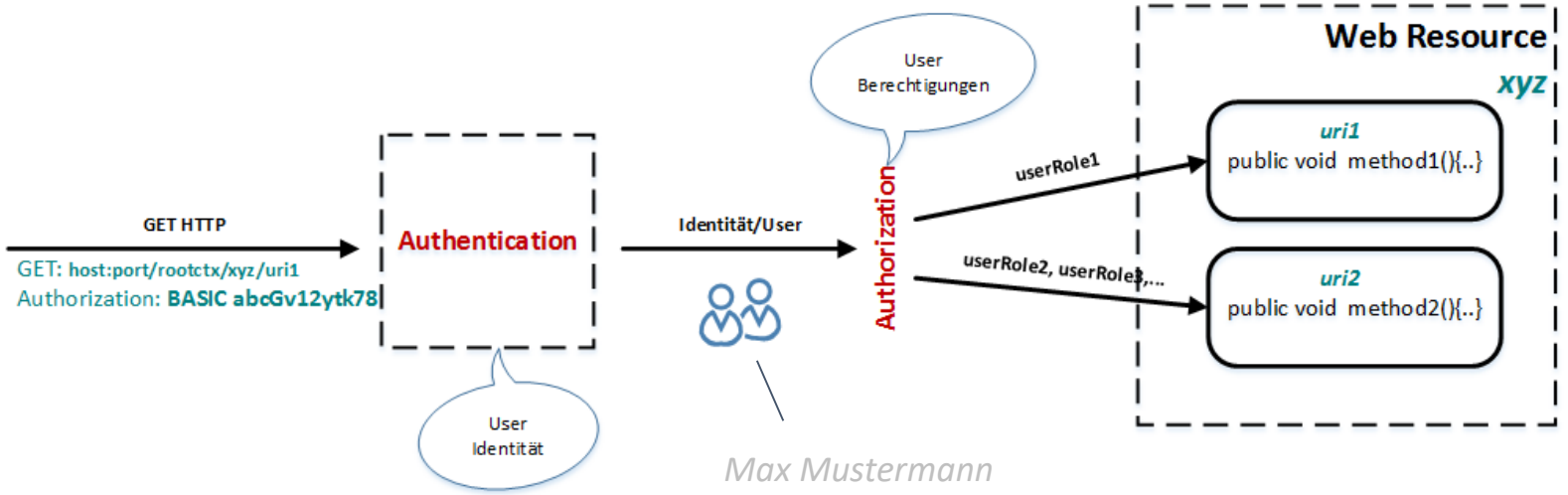
- Standard Verfahren für die Autorisierung in Web- und Applikationscontainer
- **Defacto-Standard** in verwaltete Container

Fazit: (Web-) Sicherheit in Java EE

JAAS (Java-Authentication-and-Authorization)

- **HTTP Basic Authentication (Defacto-Standard)**

1. In den meisten Application Server vorhanden (JACC, JASPI)!
2. Leicht zu implementieren und skalierbar



Zertifikat

```
Marcs-MacBook-Pro:keystores-local ngj$ keytool -genkey -keyalg RSA -alias tomcat
Keystore-Kennwort eingeben:
Neues Kennwort erneut eingeben:
Wie lautet Ihr Vor- und Nachname?
  [Unknown]:  Marc Nguidjol
Wie lautet der Name Ihrer organisatorischen Einheit?
  [Unknown]:
Wie lautet der Name Ihrer Organisation?
  [Unknown]:
Wie lautet der Name Ihrer Stadt oder Gemeinde?
  [Unknown]:  Paderborn
Wie lautet der Name Ihres Bundeslands?
  [Unknown]:  NRW
Wie lautet der Ländercode (zwei Buchstaben) für diese Einheit?
  [Unknown]:  DE
Ist CN=Marc Nguidjol, OU=Unknown, O=Unknown, L=Paderborn, ST=NRW, C=DE richtig?
  [Nein]:  ja

Schlüsselkennwort für <tomcat> eingeben
        (RETURN, wenn identisch mit Keystore-Kennwort):
```

```
-rw-r--r--    1 ngj   staff      2241  9 Okt 23:41 .keystore
```