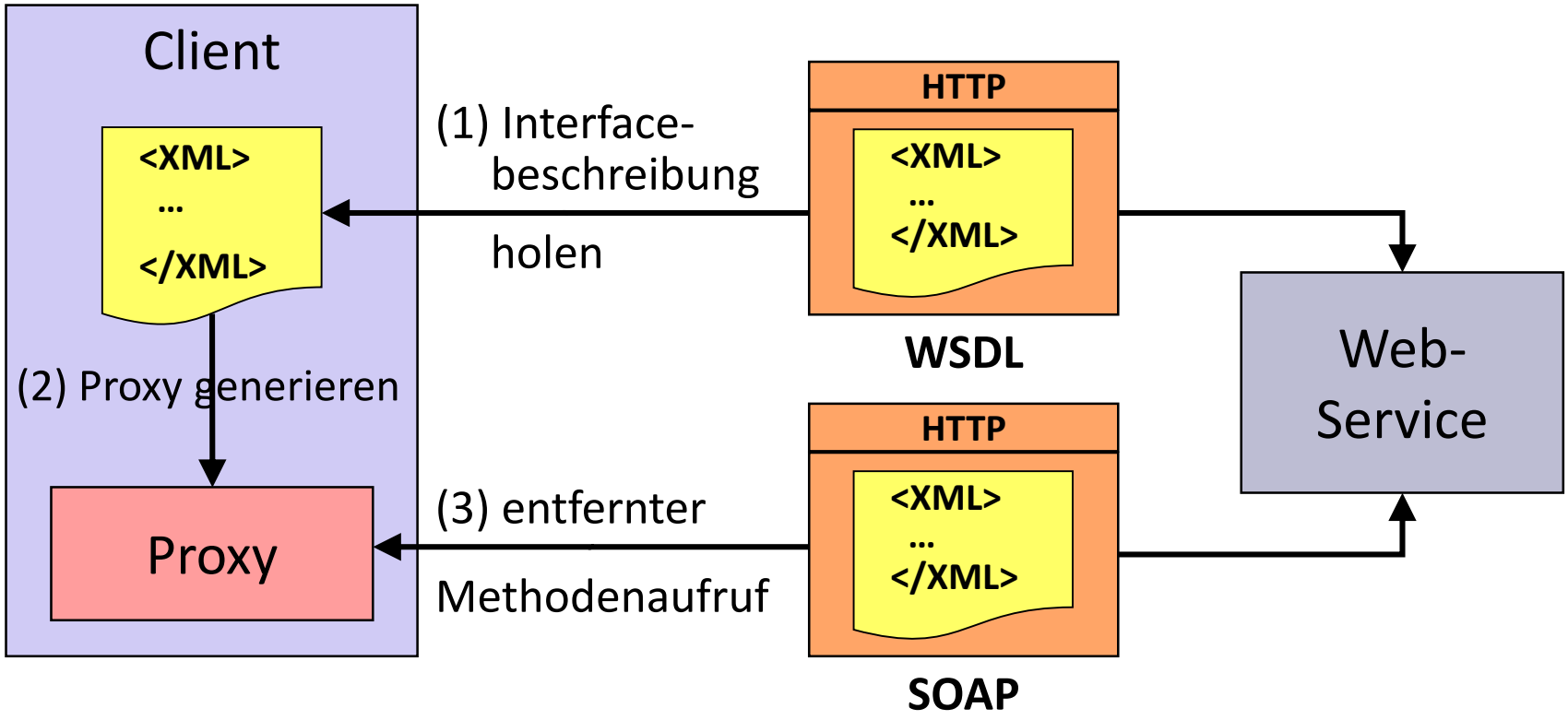
SOAP-basierte Web-Services mit JAX-WS

Version 6.6

Grundlagen

SOAP-basierte Web-Services: „Big Picture“



Kennzeichen von SOAP-basierten Web-Services

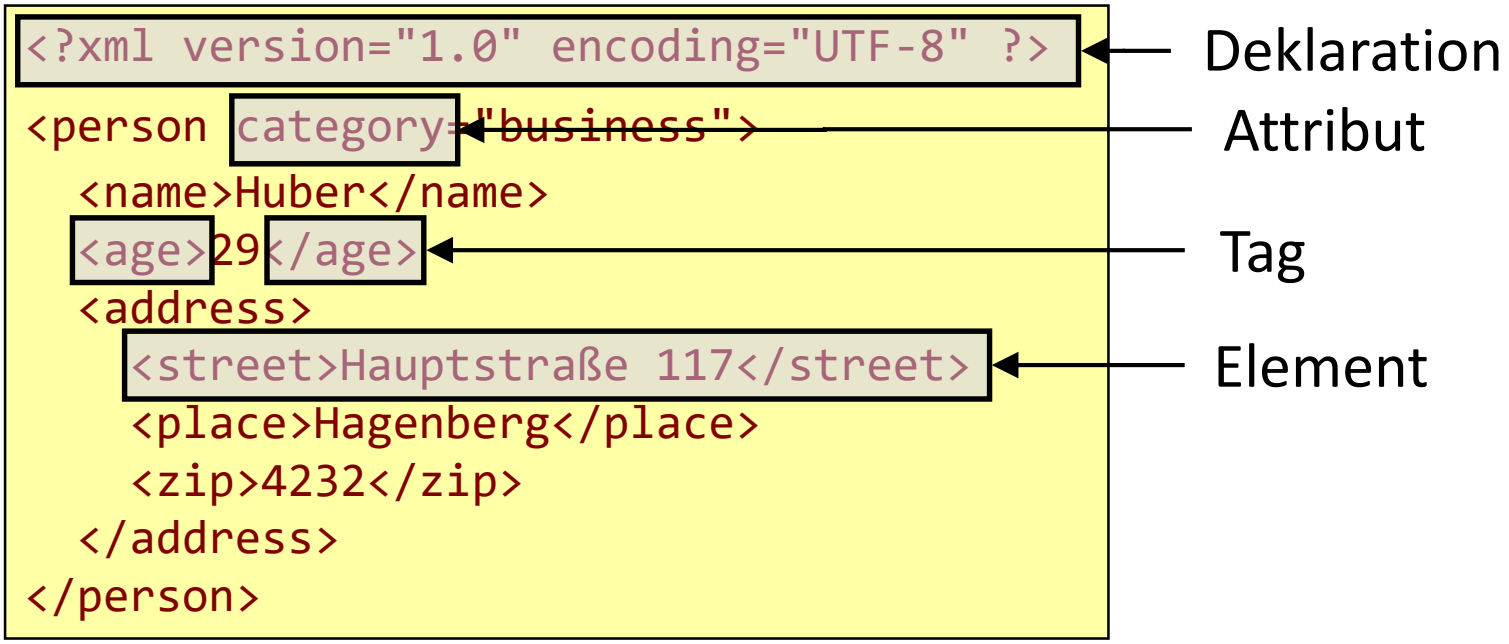
- Operations-Zentriertheit
  - Web-Services bieten ein Interface mit beliebigen Methoden.
  - Daten werden in Form von Eingangs- und Rückgabeparametern zwischen Client und Service ausgetauscht.
- Zustandslosigkeit
- Plattformunabhängigkeit
  - Unterstützung auf vielen Plattformen und für viele Programmier-sprachen.
- Metadaten
  - Maschinenlesbare Metadaten in sprachunabhängigen Format.
- Standardisierung
  - Nachrichtenformat und Metadaten sind standardisiert.
  - Viele weitere Standards im Bereich Security, sichere Nachrichtenübertragung und Transaktionen (WS-*-Standards).

Relevante Standards

- XML: eXtensible Markup Language
  - Strukturierte Darstellung von Daten,
  - Metasprache zur Definition von Sprachen,
  - Anwendung: UDDI, WSDL, SOAP.
- XML-Schema
  - Definition der Grammatik von XML-Sprachen.
- SOAP: ursprünglich Simple Object Access Protocol
  - Standardisierte Darstellung von Daten,
  - Darstellung von Methodenaufrufen und Parametern.
- WSDL: Web Service Description Language
  - Beschreibungssprache für Web-Services.

XML (eXtensible Markup Language)

- Metasprache zur Definition anderer Sprachen.
- XML-Sprachen beschreiben die Struktur von Dokumenten und Daten.
- Begriffsbestimmung:



Namensräume in XML-Dokumenten

- Aufgabe: Gewährleistung der Eindeutigkeit von Tags und Attributen.
- Default-Namenraum:
- Deklaration eines Namenraums:
- Verwendung eines Namenraums:

```
<myTag xmlns="URI" ...>
```

```
<myTag xmlns:myNS="URI" ...>
```

```
<myNS:tag>...</myNS:tag>
```

```
<person category="business"
 xmlns="http://myCompany/person"
 xmlns:addr="http://myCompany/address">
  <name>Huber</name>
  <age>29</age>
  <addr:address>
    <addr:street>Hauptstr.</addr:street>
    <addr:place>Hagenberg</addr:place>
    <addr:zip>4232</addr:zip>
  </addr:address>
</person>
```

Deklaration des **Default-Namenraums**

Deklaration des **Namenraums addr**

Qualifizierter Name (*QName*)

XML-Schema

- Ein *XML-Schema* ist eine XML-Sprache zur Beschreibung von XML-Sprachen.
- Aufbau eines XML-Schema-Dokuments (*.xsd):

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://myCompany/person"
 xmlns:tns="http://myCompany/person">
  <element name="person" type="tns:personType"/>
  <complexType name="personType">
    ...
  </complexType>
  <complexType name="addressType">
    ...
  </complexType>
</schema>
```

  - *targetNamespace* legt den Namenraum der definierten Elemente und Typen fest.

Verbindung Schema/Schema-Instanz

- Durch globales Element wird Wurzelelement eines XML-Dokuments definiert.

```
<element name="person" type="tns:personType"/>
<complexType name="personType">
  ...
</complexType>
```

- *xsi:schemaLocation* referenziert das Schema-Dokument im XML-Dokument.

```
<pns:person category="business"
 xmlns:pns="http://myCompany/person"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://myCompany/person person.xsd">
   <name>Huber</name>
   ...
</pns:person>
```

SOAP

- Merkmale
  - SOAP ist eine XML-Sprache mit einem XML-Schema.
  - SOAP-Nachrichten werden über Transportprotokolle übertragen (*tunneling*): HTTP, SMTP, TCP/IP.
  - SOAP ist sprach- und plattformunabhängig.
  - SOAP ist unabhängig von Messaging-Protokoll:
    - synchron/asynchron,
    - unidirektional (one-way) bzw. bidirektional (request/response).
  - SOAP ist das Basisprotokoll für Web-Services.
- Anwendung: A2A-Kommunikation
  - Enterprise Application Integration
  - B2B-Kommunikation: ähnlich EDI-Standards.

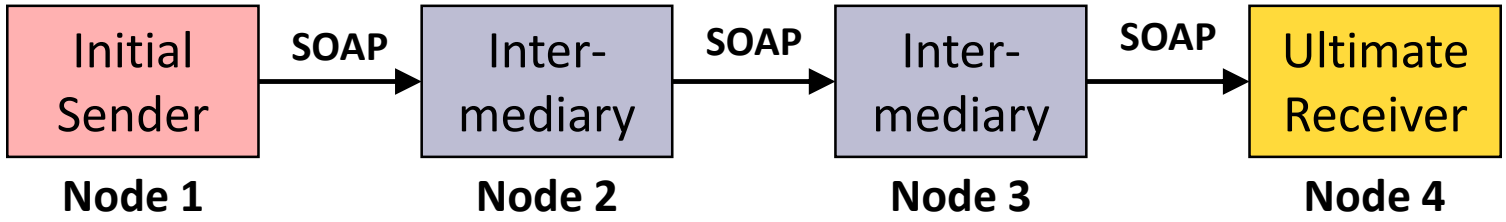
Struktur eines SOAP-Dokuments

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelop
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelop/"
  <soap:Header>
    ...
  </soap:Header>
  <soap:Body>
    ...
  </soap:Body>
</soap:Envelop>
```

- Header (optional)
  - Infos über die Nachricht
    - Security-Tokens,
    - Transaktions-Informationen,
    - Routing-Anweisungen.
- Body
  - Nachricht im XML-Format.

SOAP-Header

- *Message Path:* Eine SOAP-Nachricht durchläuft mehrere Knoten (*Nodes*) auf ihrem Weg vom Sender zum Empfänger.



- Durch das *actor/role*-Attribut im Header wird die Nachricht bestimmten Rollen zugeordnet.

```
<soap:Header>
  <ns:myMessage soap:actor="http://myRole" soap:mustUnderstand="1">
    ...
  </ns:myMessage>
</soap:Header>
```

- „Identifiziert“ sich ein Knoten mit einer bestimmten Rolle, muss er die Nachricht verarbeiten.
- Zwischenknoten dürfen den Header verändern (Elemente löschen), aber nicht den Nachrichtenkörper.

SOAP-Body

- Der SOAP-Body muss ein wohl-geformtes XML-Dokument sein.
- Der SOAP-Body wird gegen das WSDL-Dokument (XML-Schema) validiert
- Body enthält *Daten* oder *Parameter eines entfernten Methodenaufrufs.*

Request: Eingangsparameter

```
<soap:Envelop>
  <soap:Body>
    <ns:getAge>
      <ns:name>Huber</ns:name>
    </ns:getAge>
  </soap:Body>
</soap:Envelop>
```

Response: Rückgabeparameter

```
<soap:Envelop>
  <soap:Body>
    <ns:getAgeResponse>
      <result>29</result>
    </ns:getAge>
  </soap:Body>
</soap:Envelop>
```

SOAP-Faults

- Fehler-Nachrichten (*soap faults*) werden an den Vorgänger-knoten geschickt.
- Struktur einer Fehlernachricht:

```
<soap:Body>
 <soap:Fault>
   <faultcode>soap:Client</faultcode>
   <faultString>Invalid ID</faultString>
   <faultActor>http://myActor</faultActor>
   <detail>XML document fragment</detail>
 </soap:Fault>
</soap:Body>
```

  - Fehlercodes:
    - *soap:Client:* Falsche Parameter.
    - *soap:Server:* Fehler auf Serverseite.
    - *soap:Mustunderstand:* Unbekanntes obligatorisches Header-Element.
    - *soap:VersionMismatch:* Falsche SOAP-Version.

SOAP over HTTP (HTTP-tunnelling)

- SOAP ist unabhängig von Transportprotokoll.
- Am häufigsten wird aber HTTP/HTTPS verwendet.
  - Vorteil: Keine Probleme mit Firewalls (derzeit).

```
POST /URL HTTP/1.1
Host: host-address
Content-Type: text/xml
Content-Length: nnn
SOAPAction: "URL/getAge"

<?xml version="1.0 …>
<soap:Envelop>
 <soap:Body>
  <ns:getAge>
   <ns:name>Huber</ns:name>
  </ns:getAge>
 </soap:Body>
</soap:Envelop>
```

HTTP-Request

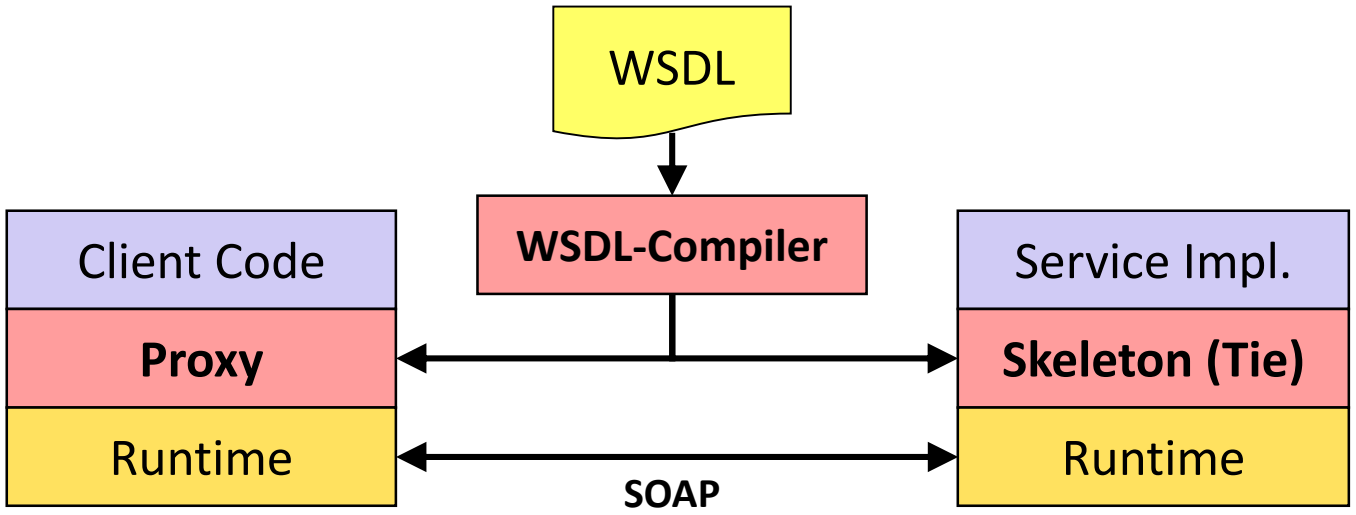
```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: nnn

<?xml version="1.0 …>
<soap:Envelop>
 <soap:Body>
  <ns:getAgeResponse>
   <result>29</result>
  </ns:getAge>
 </soap:Body>
</soap:Envelop>
```

HTTP-Response

WSDL: Web Service Description Langugage

- Ein WSDL-Dokument definiert für ein Web-Service:
  - das Interface (Methoden und Parameter),
  - das Nachrichten-Format (Document/Literal, RPC/Literal, …),
  - das zu verwendende Transportprotokoll (HTTP, SMTP, TCP/IP, …),
  - die Adresse (URL).
- Anwendung: Generierung von Tie- (Skeleton-)/Proxy-Code:



Struktur eines WSDL-Dokuments

<definitions name = *"MyWebService"*
targetNamespace = *"http://MyCompany/MyWS"*
xmlns:tns = "http://MyCompany/MyWS"
xmlns = "http://schmas.xmlsoap.org/wsdl" >

<types> … </types>

<message> … </message>

<portType> … </portType>

<binding> … </binding>

<service> … </service>

</definitions>

WSDL: *types*

- Definition von benutzerdefinierten einfachen und komplexen Typen.
- Typen werden für die Definition von Nachrichten verwendet.

```
<types>
  <xsd:schema targetNamespace="http://MyCompany/MyWS">
    <xsd:complexType name="ArrayOfString">
      <xsd:sequence>
        <xsd:element name="string" type="xsd:string"
                     minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:schema>
</types>
```

WSDL: *message*

- *message* definiert den Inhalt einer SOAP-Nachricht.
- Für jede eingehende und jede ausgehende Nachricht wird jeweils ein *message*-Element definiert.
- Bei der Nachrichtenart *RPC* wird jeder Parameter durch ein *part*-Element beschrieben.
- Bei der Nachrichtenart *Document* werden alle Parameter zu einem *part*-Element zusammengefasst, und dieses mit einem XML-Schema-Element beschrieben.

```
<message name="GetGradesRequest">
  <part name="studentID" type="xsd:string"/>
  <part name="year" type="xsd:int"/>
</message>

<message name="GetGradesResponse">
  <part name="grades" type="tns:ArrayOfInt"/>
</message>
```

WSDL: *portType*

- *portType* definiert das Interface eines Web-Service.
- Das Interface wird durch ein Folge von Operationen (*operation*) definiert.
- Jede Operation besteht aus
  - einer ausgehenden Nachricht,
  - einer eingehenden Nachricht (optional) und
  - einer Fehlernachricht (optional):

```
<portType name="Student">
  <operation name="GetGrades"/>
    <input name="GetGradesRequest"/>
    <output name="GetGradesResponse"/>
    <fault name="InvalidParams"
           message="tns:InvalidParams"/>
  </operation>
</portType>
```

WSDL: *binding*

- *binding* definiert, wie die Interface-Methoden und -Parameter auf SOAP abgebildet und übertragen werden:
  - legt Nachrichtenart (*style*) fest: *RPC* oder *document*,
  - legt Darstellungsform (*encoding*) fest: *literal* oder *encoded*,
  - definiert das Transportprotokoll: HTTP, SMTP, …

```
<binding name="Student_Binding type="tns:Student">
  <soap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetGrades">
    <soap:operation soapAction="http://.../GetGrades"/>
    <input message="tns:GetGradesRequest">
      <soap:body use="literal" namespace=".../MyWS"/>
    </input>
    <output message="tns:GetGradesResponse"> ... </output>
  </operation>
</binding>
```

WSDL: *service*

- Jedem *service*-Element können mehrere Ports (*port*) zugeordnet sein.
- Ein Port ordnet einer Bindung (*binding*) eine Internet-Adresse zugeordnet.

```
<service name="StudentService">
  <port name="StudentPort" binding="tns:Student_Binding">
    <soapbind:address
      location="http://.../StudentService"/>
  </port>
</service>
```

Nachrichtenmodi

- SOAP und WSDL unterscheiden zwischen folgenden vier Nachrichtenmodi (*messaging modes*).

messaging style / encoding	**Document**	**RPC**
Literal	Document/Literal	RPC/Literal
Encoded	Document/Encoded	RPC/Encoded

Nachrichtenmodus Dokument/Literal

- Merkmale
  - Body enthält ein Fragment eines XML-Dokuments.
  - Methodenname umgibt den Nachrichtentyp *(wrapped)*.
  - Nachricht kann gegen ein XML-Schema in WSDL validiert werden.
  - Standardmodus in .NET

- SOAP

```
<soap:Body>
  <ns:add xmlns:ns="swk5">
    <arg0>10.0</arg0>
    <arg1>20.0</arg1>
  </ns:add>
</soap:Body>
```

- WDSL

```
<types>
 <xs:element name="add"
             type="tns:add"/>
 <xs:complexType name="add">
  <xs:sequence>
   <xs:element name="arg0"
               type="xs:double"/>
   <xs:element name="arg1"
               type="xs:double"/>
  </xs:sequence>
 </xs:complexType>
</types>

<message name="add">
 <part name="parameters"
       element="tns:add"/>
</message>

<portType name="Calculator">
 <operation name="add">
  <input message="tns:add"/>
 </operation>
</portType>
```

Nachrichtenmodus RPC/Literal

- Merkmale:
  - Body enthält Aktualparameter des Methodenaufrufs.
  - Jeder Parametertyp wird in WSDL einzeln beschrieben.
  - Standardmodus bei Java-WS.
- SOAP

```
<soap:Body>
  <ns:add xmlns:ns="swk5">
    <arg0>10.0</arg0>
    <arg1>20.0</arg1>
  </ns:add>
</soap:Body>
```

- WDSL

```
<message name="add">
  <part name="arg0"
        type="xsd:double" />
  <part name="arg1"
        type="xsd:double" />
</message>
<portType name="Calculator">
  <operation name="add"
      parameterOrder="arg0 arg1">
    <input message="tns:add" />
    <output message=… />
  </operation>
</portType>
```

Nachrichtenart *Encoded*

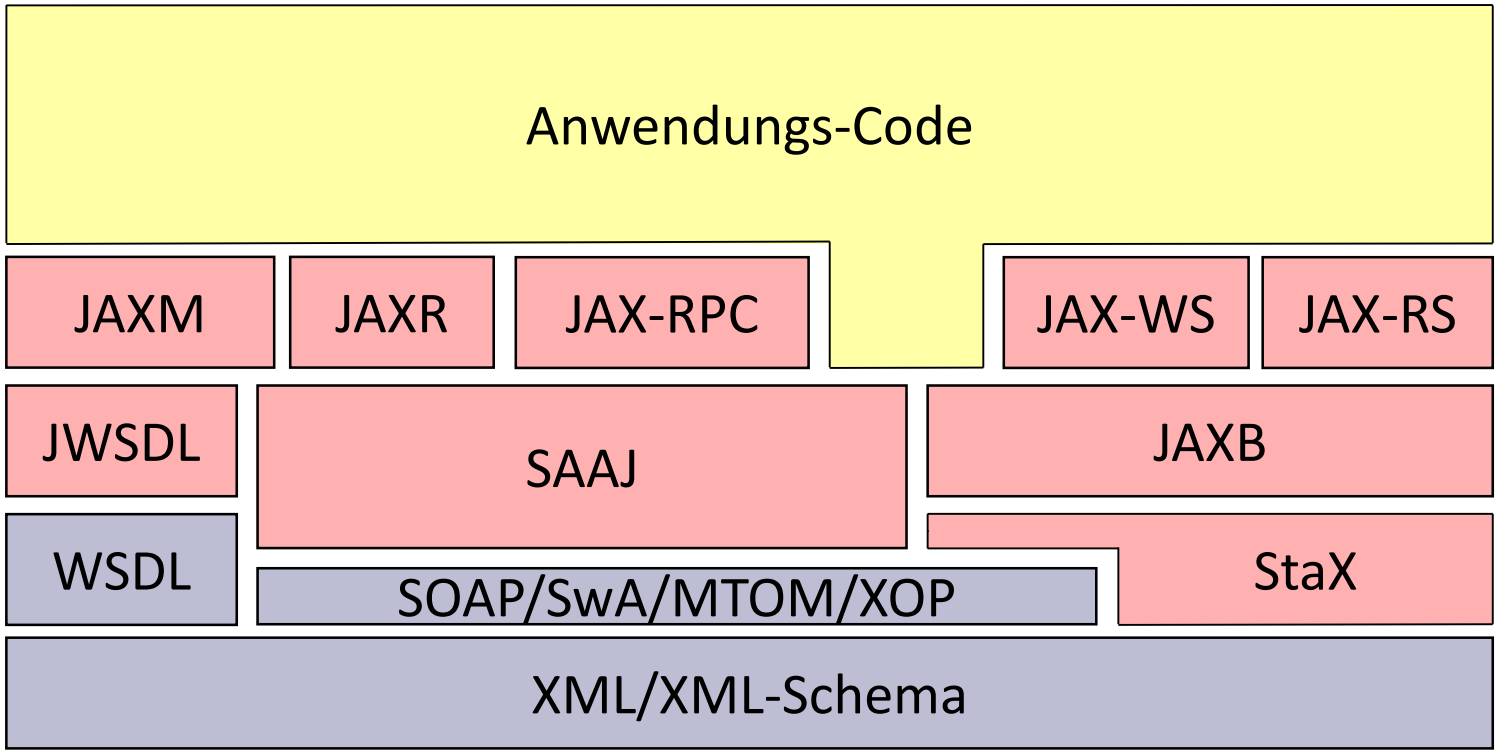
- RPC/Encoded
  - Definiert Abbildung von Datentypen auf XML-Schema.
  - Ermöglicht Repräsentation von zyklischen Objektgraphen.
  - Verwendung von in SOAP definierten Datentypen (*enc:short*, *enc:Array*).
  - Interoperabilitätsprobleme wegen vielfältiger Darstellungsmöglichkeiten.
  - Nicht Basic-Profile-konform

```
<soap:body>
  <myMethod>
    <x xsi:type="xsd:int">10</x>
    <y xsi:type="xsd:float">20.0</y>
  </myMethod>
</soap:body>
```

- Document/Encoded
  - Nicht Basic-Profile-konform. Wird in der Praxis nicht eingesetzt.

Implementierung von SOAP-basierten Web-Services mit JAX-WS

Java-APIs für Web-Services (1)



Java-APIs für Web-Services (2)

- JWSDL: *Java APIs for WSDL*
  - Lesen, Generieren und Manipulieren von WSDL-Dokumenten.
- SAAJ: *SOAP with Attachments API for Java*
  - Lesen, Generieren und Manipulieren von SOAP-Nachrichten,
  - Versenden/Empfangen von SOAP-Nachrichten (synchron).
  - SwA ist nicht interoperabel und wird daher durch
    - MTOM: *Message Transmission Optimisation Mechanism* bzw.
    - XOP: *XML-binary Optimized Packaging*

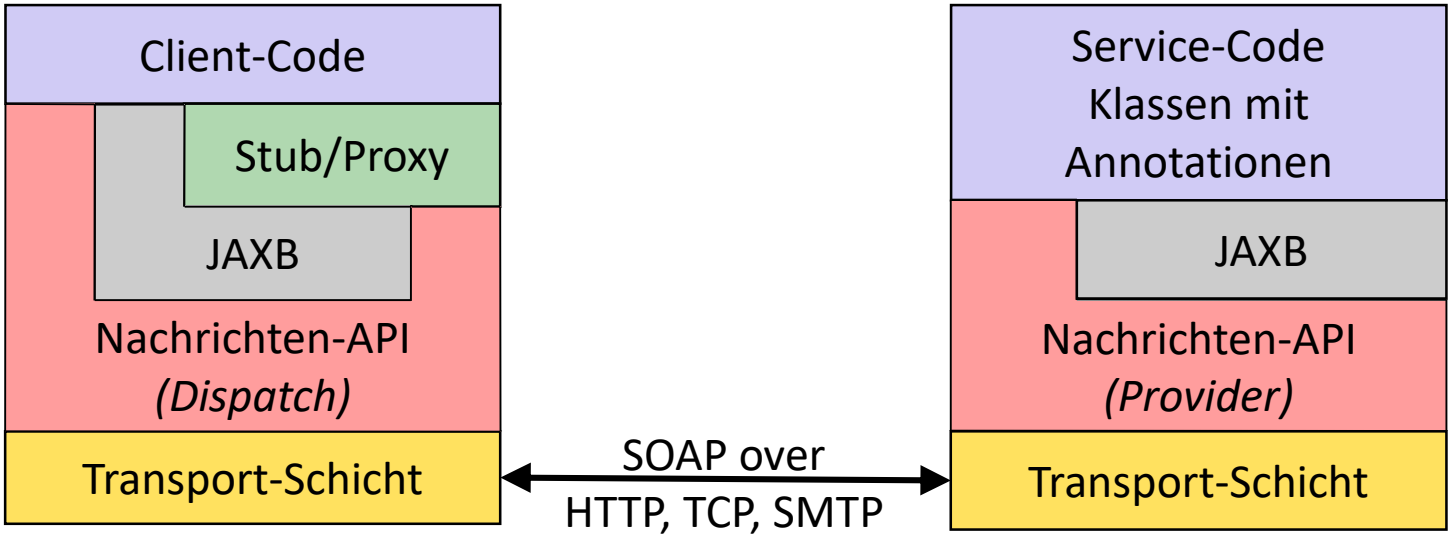
    abgelöst.
- JAXM: *Java API für XML Messaging*
  - Asynchroner Nachrichtenaustausch,
  - sichere Nachrichtenübertragung,
  - hat keine Bedeutung mehr.

Java-APIs für Web-Services (3)

- JAX-RPC: *Java-API for XML-based RPC*
  - API zur Implementierung von Web-Services und zur Erstellung von RPC-basierten Clients für Web-Services.
- JAX-WS: *Java-API for XML Web Services*
  - Nachfolgetechnologie von JAX-RPC (≥ Java 5)
- JAX-RS: Java API for RESTful Web Services
- JAXB: *Java Architecture for XML Binding*
  - Klassen ↔ XML-Schema, Objekte ↔ XML-Dokument
  - Wird bei JAX-WS eingesetzt.
- JAXR: *Java-API for XML-based Registries*
  - Interface für den Zugriff auf UDDI-Verzeichnisse.

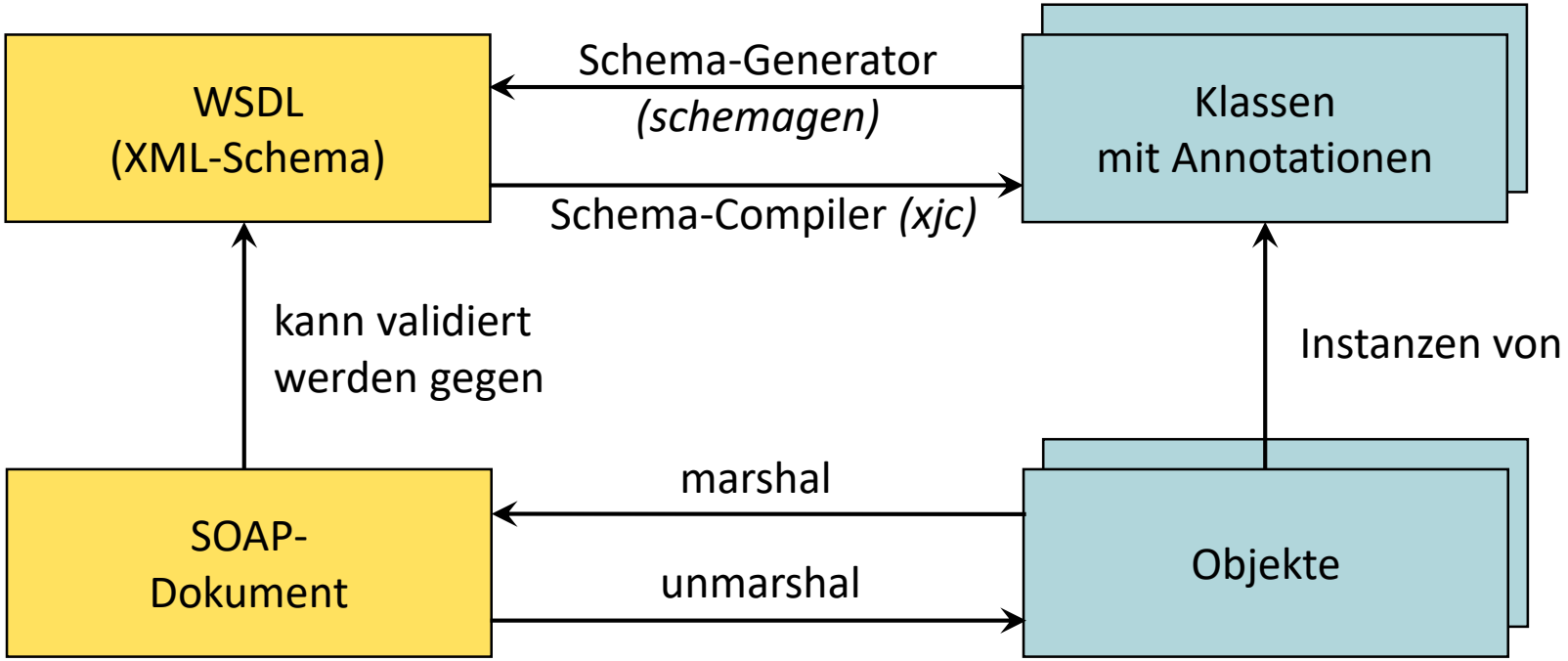
JAX-WS

- API zur Implementierung von Web-Services und zur Erstellung von Clients für Web-Services.
  - RPC-basiertes Programmiermodell (ähnlich zu RMI).
  - Services (Komponenten) sind jedoch zustandslos.
  - Protokollschicht wird vollständig gekapselt.



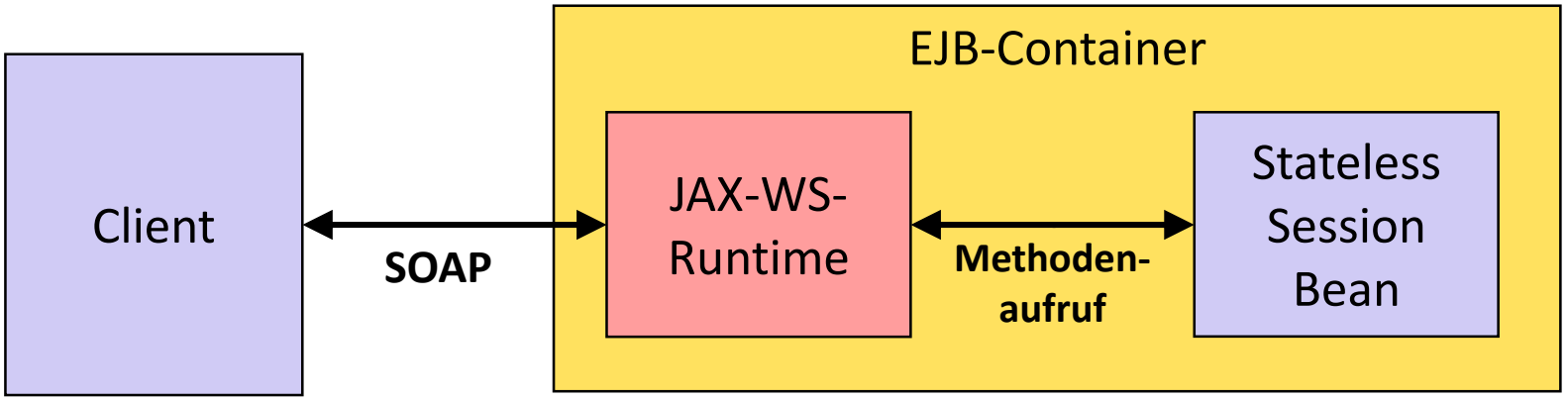
JAVA Architecture for XML-Binding (JAXB)

- JAXB ist eine API, mit der Daten effizient zwischen einer XML-Repräsentation und einer Repräsentation als Objektgraph konvertiert werden können.



EJB-Endpunkt (Java EE)

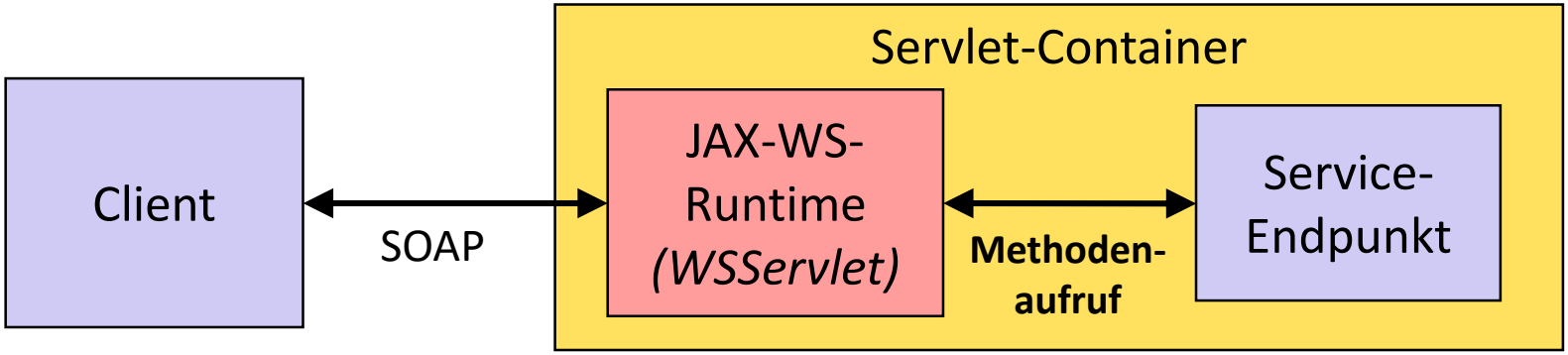
- Stateless Session Beans können als Web-Service-Endpunkte exportiert werden.



  - Session-Bean implementiert das Endpunkt-Interface und stellt damit die Implementierung des Web-Service dar.
  - Die Session-Bean-Klasse und -Methoden werden mit JAX-WS-Annotationen versehen.
  - Der EJB-Container ist für die Instanzenverwaltung zuständig.

Servlet-Endpunkt (Java SE)

- Architektur



  - Servlet-Endpunkte werden in einem Servlet-Container installiert.
  - Im Servlet-Container müssen JAX-WS-spezifische Erweiterungen installiert werden (jaxws-api.jar, jaxb.jar).
  - Aufrufe von Clients werden von einem zentralen Servlet, das Bestandteil der JAX-WS-Laufzeitumgebung ist, übernommen und an die Endpunkte verteilt.
  - JAX-WS stellt eine minimale Web-Server-Infrastruktur zur Verfügung → Web-Service kann auch in Standalone-Anwendung gehostet werden.

Servlet-Endpunkt: Konfiguration

web.xml

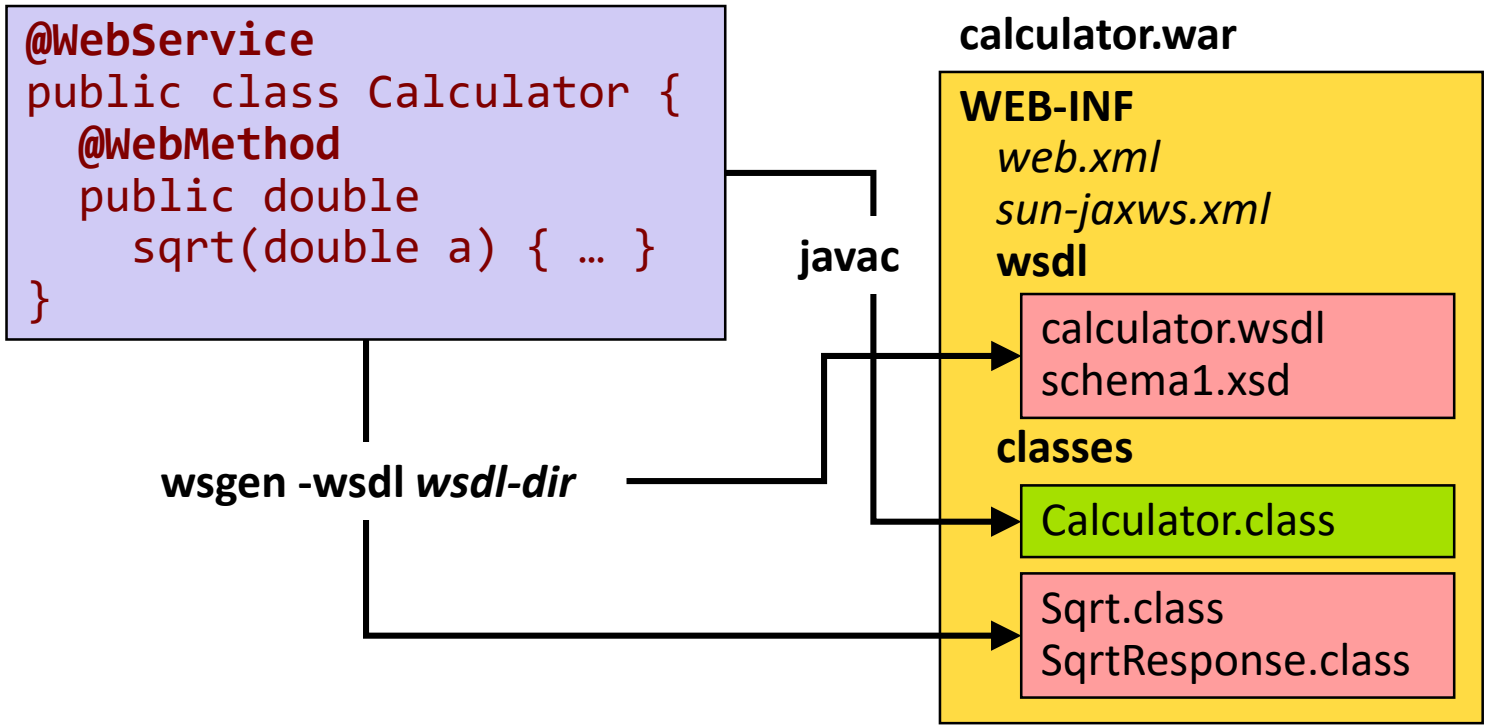
```
<servlet>
  <servlet-name>WSServlet</servlet-name>
  <servlet-class>
    com.sun.xml.ws.transport.http.servlet.WSServlet
  </servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>WSServlet</servlet-name>
  <url-pattern>/*</url-pattern>
</servlet-mapping>
```

sun-jaxws.xml

```
<endpoints xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime" version="2.0">
  <endpoint
    name="calculator"
    implementation="service.Calculator"
    url-pattern="/calc" />
</endpoints>
```

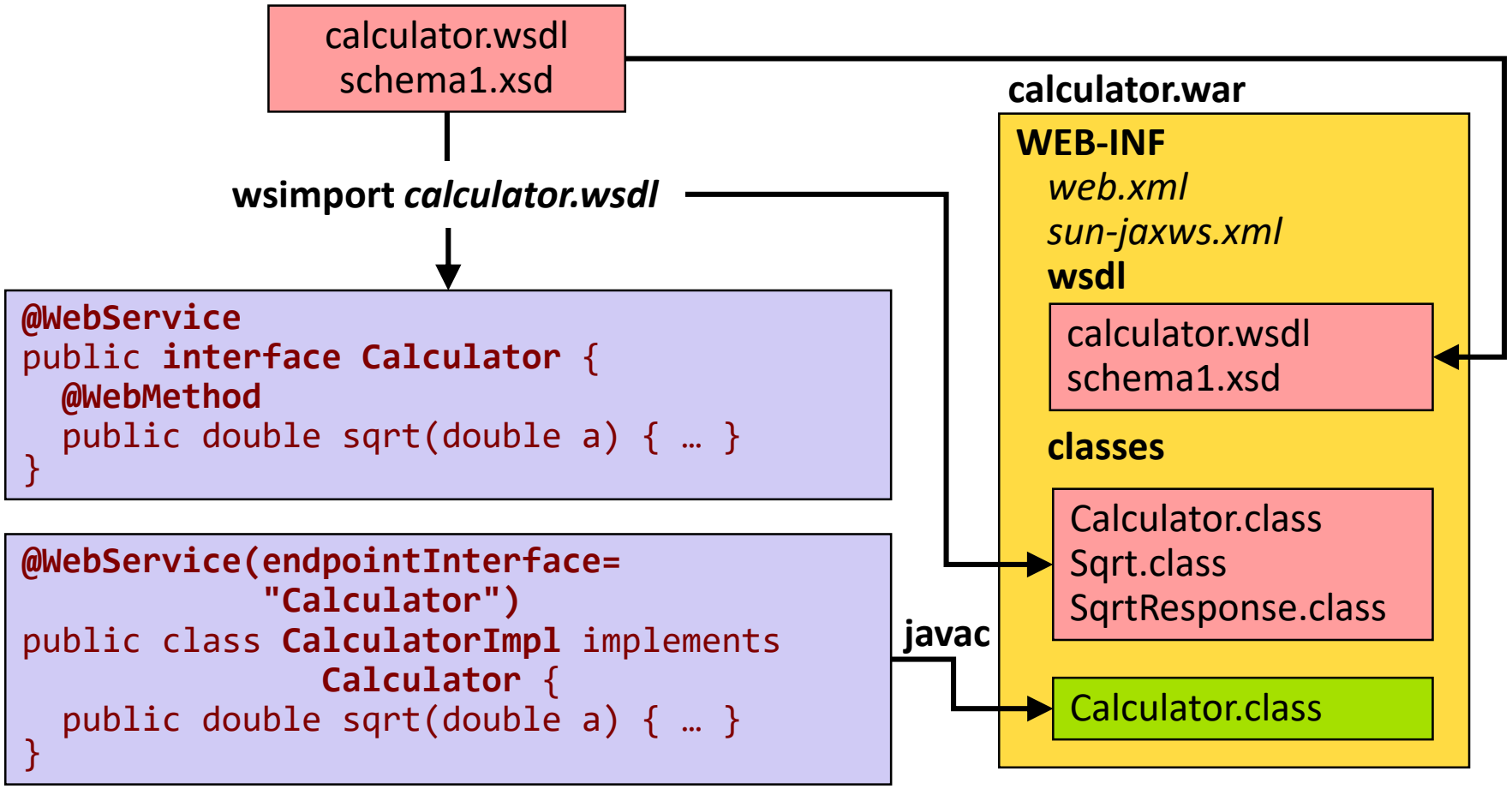
Implementierung eines Servlet-Endpunkts (1)

- Ausgehend von der Implementierung des Web-Service („implementation first“)



Implementierung eines Servlet-Endpunkts (2)

- Ausgehend vom WSDL-Dokument („contract first“)



JAX-WS-Annotationen

- Über Annotationen kann definiert werden, wie Service-Operationen auf SOAP abgebildet werden.

```
@WebService(name="Calculator",
            serviceName="CalculatorService",
            targetNamespace="http://swk5")
@SOAPBinding(style=Style.DOCUMENT,
             use=Use.LITERAL,
             parameterStyle=ParameterStyle.WRAPPED)
public class Calculator {
  @WebMethod(operationName="SquareRoot")
  @WebResult(name="SquareRootResult")
  public double sqrt(
      @WebParam(name="dblValue", mode=Mode.IN) double a) {
    return Math.sqrt(a);
  }
}
```

Wichtige Annotationen in JAX-WS

Annotation	Aufgabe
@WebService	Definition der Klasse, die das Web-Service implementiert.
@WebMethod	Festlegung der Methoden, die vom Web-Service exportiert werden sollen.
@SoapBinding	Legt fest, welcher Nachrichtenmodus (Document/Literal, RPC/Literal, …) zur Formatierung der SOAP-Nachrichten verwendet werden soll.
@WebParam	Definition der Methodenparameter (Name, Ein/Ausgangs/Übergangsparameter).
@WebResult	Definition des Rückgabewertes einer Methode.
@OneWay	Methodenaufruf soll asynchron durchgeführt werden.
@XmlType (JAXB)	Festlegen der Eigenschaften von komplexen Typen.
@XmlElement (JAXB)	Festlegen der Eigenschaften von Datenkomponenten

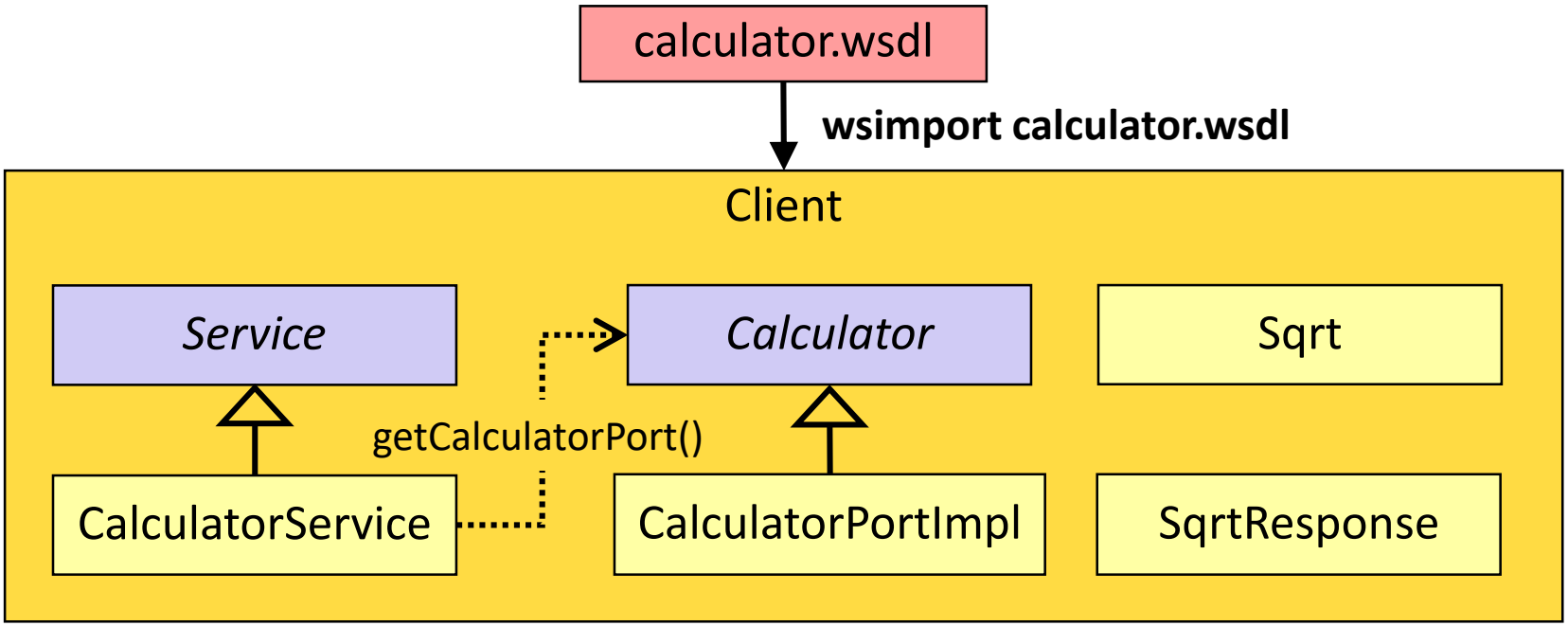
Deployment: Installation des Web-Service

- Variante 1: Standalone-Anwendung
  - JAX-WS stellt die Klasse *Enpoint* zur Verfügung, mit der Endpunkte erzeugt und veröffentlicht werden könne:

```
public static void main(String[] args) {
  Calculator calc = new Calculator();
  Endpoint.publish("http://localhost:8080/calculator", calc);
}
```

- Variante 2: Installation in einem Servlet-Container (z.B. Tomcat)
  - Verpacken der Metadaten und der Class-Dateien in ein Web-Archiv (*.war) und Deployment im Servlet-Container.
  - Vorteile:
    - Robuste Infrastruktur,
    - Besser Skalierbarkeit wegen ausgereifter Instanzenverwaltung.

Clients (1): Static Stubs



CalculatorClient:

```
CalculatorService calcService = new CalculatorService();
Calculator calcProxy = calcService.getCalculatorPort();
double result = calcProxy.sqrt(2.0);
```

Clients (2): Dynamic Proxies

- Dynamic Proxy: Proxy-Code wird zur Laufzeit erzeugt.
- Der Proxy-Code wird aus dem WSDL-Dokument abgeleitet.

```
Service calcService =
  Service.create(new URL("http://host:port/calculator?wsdl"),
                 new QName("http://mycompany.serivces/,
                           "CalculatorService"));

// get the dynamic proxy
Calculator calcProxy = calcService.getPort(Calculator.class);
```

Clients (3): Verwendung der Nachrichten-API

- *Dispatch<T>* definiert Methoden zum Versenden und Empfangen von Nachrichten: *T invoke<T msg>.*
- Nachrichten können auf verschiedenen Ebenen bearbeitet werden:
  - *Dispatch<Source>* → XML-Nachricht,
  - *Dispatch<SOAPMessage>* → SOAP-Nachricht,
  - *Dispatch<Object>* → JAXB-Nachricht
- Beispiel:

```
Service service = new CalculatorService();
SOAPMessage message = createSOAPMessage("…");

Dispatch<SOAPMessage> soapDispatch =
  service.createDispatch(new QName("http://swk5","CalculatorPort"),
                         SOAPMessage.class,
                         Service.Mode.MESSAGE);

SOAPMessage response = soapDispatch.invoke(message);
```

Clients (4): Asynchrone Aufrufe

- In der Binding-Konfiguration kann man festlegen, dass in der Proxy-Klasse auch Methoden für asynchrone Aufrufe generiert werden sollen.

```
class SqrtResultHandler implements AsyncHandler<SqrtResponse> {
  public void handleResponse(Response<SqrtResponse> resp) {
    try {
      SqrtResponse response = resp.get();
      double result = response.getReturn();
    }
    catch (Exception e) {}
  }
}

CalculatorService calcService = new CalculatorService();
Calculator calcProxy = calcService.getCalculatorPort();
Future<?> future = calcProxy.sqrtAsync(2.0,
                        new SqrtResultHandler());
```

Unterstützte Datentypen (1)

- Einfache Typen: *boolean, byte, short, int, long, float, double*.
- Wrapper-Klassen: *Short, Integer, Long, Float, Double, …*
- Andere Klassen: *String, Date, Calendar, BigDecimal, BigInteger.*
- Arrays: Elementtyp muss unterstützt sein.
- Strukturen *(value classes)*
  - nur Properties (Setter+Getter) werden serialisiert.

```
@XmlType(name="Rational")
class Rational {
  private long num, denom;
  public void setNum(
    long) { … }
  public long getNum() { … }
  …
}
```

```
@WebService
class Calculator {
  @WebMethod
  double toDouble(
    Rational r);
}
```

Unterstützte Datentypen (2)

- Holder-Klassen
  - Notwendig für Übergangs- und Ausgangsparameter.

```
class Holder<T> {
  public T value;
  public Holder(T val) { this.value = val; }
}
```

- Definition einer Service-Methode mit einem Übergangsparameter

```
@WebMethod
void twice(@WebParam(mode=WebParam.Mode.INOUT)
           Holder<Double> a) {
  a.value *= 2;
}
```

- Implementierung des zugehörigen Clients

```
Calculator calc = …;
Holder<Double> inout = new Holder<Double>(2.0);
calc.twice(inout);
System.out.println(inout.value); // → 4.0
```

Einschränkungen von Web-Services

- Das Programmiermodell von JAX-WS ist ähnlich zu jenem von RMI und verwandten Technologien.
- Wesentliche Einschränkung:
  - Entfernte Objekte leben nur für die Dauer eines Requests (Objekte sind *zustandslos*).
  - Web-Services können keine Referenzen auf entfernte Objekte übergeben.
    - Parametertypen dürfen nicht *java.rmi.Remote* implementieren.
  - Serialisierung von Methodenaufrufen erfolgt nicht mit dem Standard-Serialisierungsmechanismus von Java.
    - Nur Properties – definiert durch Setter- und Getter-Methoden – und öffentliche Datenkomponenten werden serialisiert.
    - private/geschützte Datenkomponenten werden nicht übertragen.

SOAP-basierte vs. RESTful Web-Services

- Vorteile von RESTful Web-Services
  - REST-basierte Frameworks sind einfacher zu implementieren.
  - Für Clients ist nur eine HTTP-Bibliothek erforderlich.
  - Ressourcen können in verschiedenen Repräsentationen ausgeliefert werden (Client kann passende Repräsentation wählen).
  - HTTP-Antworten können zwischengespeichert werden (Caching).
- Nachteile von RESTful Web-Services
  - Keine Standardisierung der Datenrepräsentation.
  - Kein Unterstützung von Standards auf Nachrichtenebene (WS-Security, WS-ReliableMessaging, WS-Transaction, …).
  - Parsen und Generieren der Ressourcenrepräsentationen kann aufwändig sein (service- und clientseitig).
- Welchen Ansatz man wählt, hängt stark vom Anwendungs-gebiet ab.