Web-Services
Grundlagen

© J. Heinzelreiter

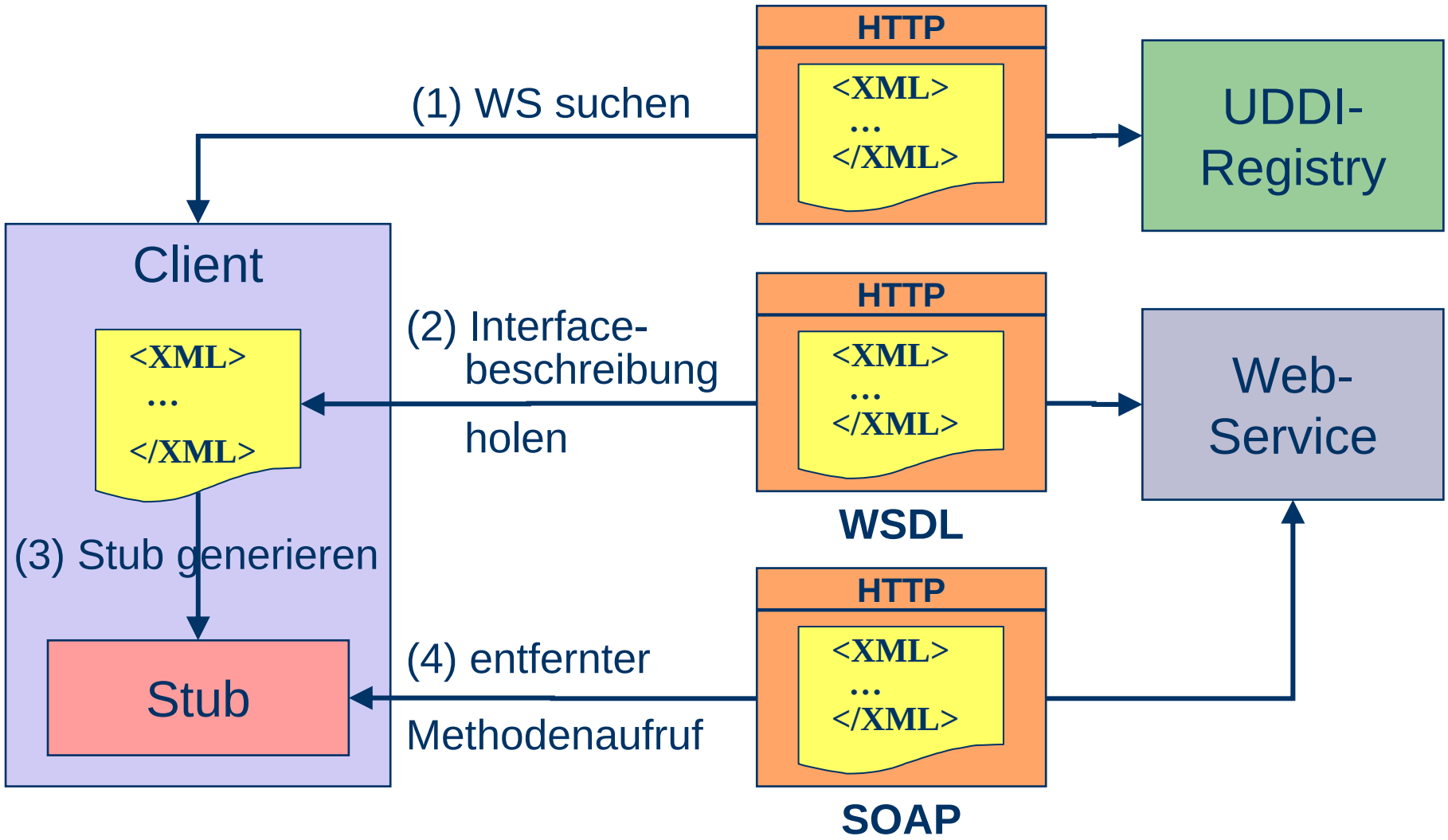
WS 2004/05

Web-Services: Definition

"*A **web service** is a software system **identified by a URI**, whose public **interfaces** and bindings are defined and **described using XML**. Its **definition can be discovered by other software systems**. These systems may then **interact with the web service** in a manner prescribed by its definition, **using XML-based messages conveyed by Internet protocols**.*"

(Web Services Architecture document, W3C)

Web-Services: "Big Picture"



Web-Services: Merkmale

- Web-Services sind *verteilte zustandslose Komponenten.*
- Breite Unterstützung der Softwareindustrie: Microsoft, Sun, IBM, ...
- Kommunikation erfolgt über *SOAP*.
- Web-Services haben *typsicheres Interface* (WSDL).
- Kategorisierung/Suchmöglichkeiten (UDDI).
- Merkmale:
  - Sprachunabhängigkeit,
  - Plattformunabhängigkeit,
  - basieren auf gängigen Internet-Standards (HTML, XML),
  - nicht an bestimmte Komponentenarchitektur gebunden.

Web-Services – Abgrenzung

- Probleme bei bestehenden Technologien für verteilte Anwendungen:
  - proprietäre Protokolle: RMI, DCOM, CORBA (teilweise),
  - verbindungsorientiert: RMI, DCOM, CORBA,
  - zusätzliche Software notwendig: CORBA,
  - Plattformabhängigkeit: DCOM,
  - Sprachabhängigkeit: RMI.
- Abgrenzung zu besehenden Technologien:
  - RMI, DCOM: für Intranet-Lösungen (sichere Verbindung),
  - CORBA: vollständige verteilte Komponente, aber mangelnde Akzeptanz.
- Nachteil von Web-Services: schlechte Performance.

Web-Services – Anwendungsgebiete

- Frei verfügbare Dienste
  - Daten werden angeboten, damit sie in möglichst viele bestehende Services integriert werden können.
  - Beispiele: Veranstaltungs-, Tourismus-, Wetterdaten.
- Gebührenpflichtige Dienste
  - Zukauf von Ressourcen (Speicher, Rechenleistung, ...).
  - Zukauf von Daten: GIS-Daten.
- Enterprise Application Integration (EAI)
  - Verteilung der Business-Logik auf mehrere Standorte.
- B2B Datenaustausch
  - Ablöse bestehender Standards (EDIFACT, ...)

Relevante Standards

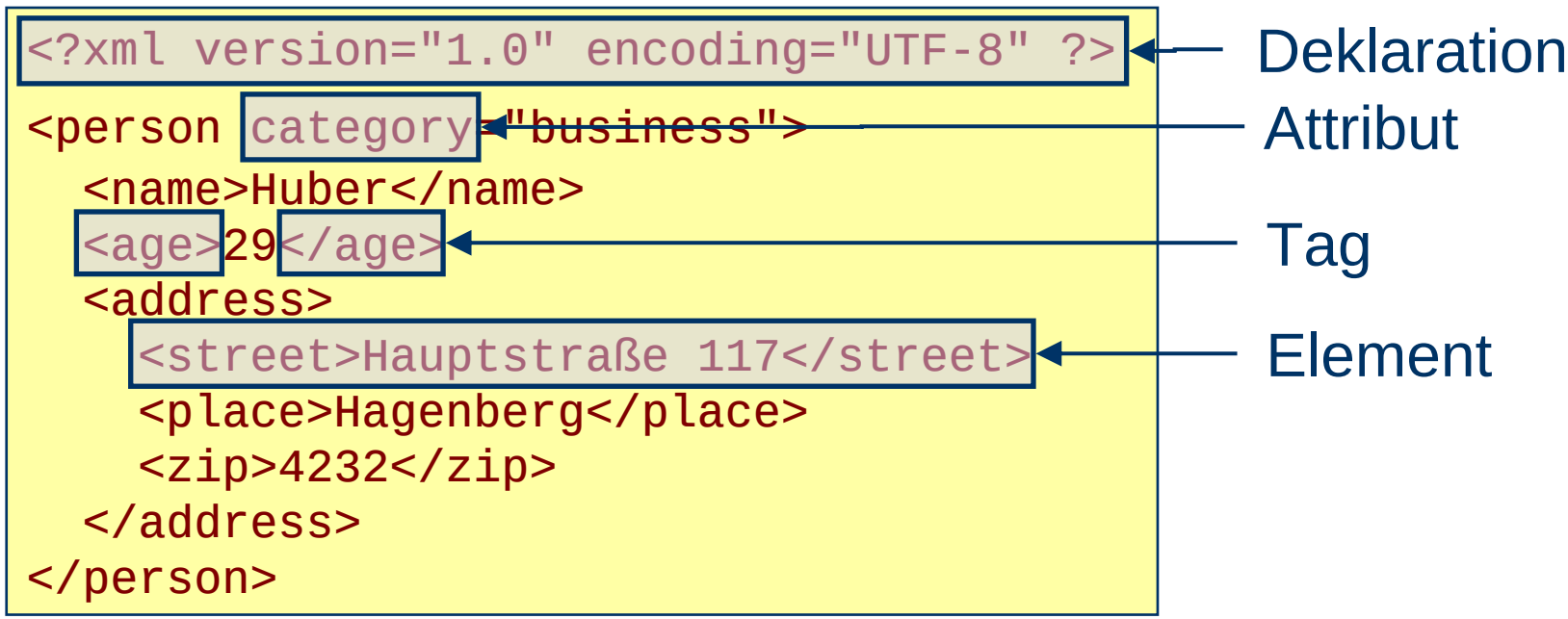
- XML: eXtensible Markup Language
  - Strukturierte Darstellung von Daten,
  - Metasprache zur Definition von Sprachen,
  - Anwendung: UDDI, WSDL, SOAP.
- XML-Schema
  - Definition der Grammatik von XML-Sprachen.
- SOAP: Simple Object Access Protocol
  - Standardisierte Darstellung von Daten,
  - Darstellung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

WS-I Basic Profile

- WS-I (= Web Services Interoperability Organization)
  - Vereinigung von Anbietern und Benutzern von Web-Services-Plattformen: Micorosoft, Sun, IBM, BEA, …
  - Aufgaben:
    - Definition von „Profilen“,
    - Erstellung von Bespielszenarien und –code für Web-Services,
    - Erstellung von Werkzeugen für Konformitätstests.
- WS-I Basic Profile 1.0, 1.1
  - Spezifikationen (WSDL, SOAP, …) sind sehr umfassend und oft nicht eindeutig.
  - Basic Profile schränkt Spezifikationen ein.
  - Ziel: Interoperabilität zwischen allen Herstellen.

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*)

Verarbeitung von XML-Dokumenten

- Eigenschafen eines XML-Dokuments
  - Wohlgeformtheit (*well-formedness*):
    - Dokument entspricht den Regeln der XML-Spezifikation.
  - Validität (*validity*).
    - Dokumentstruktur entspricht einer vorgegebenen Beschreibung (DTD oder XML-Schema).
- Arten von Parsern:
  - DOM: Parser generiert eine baumartige Repräsentation.
  - SAX: Parser generiert bei Abarbeitung Ereignisse.
- Parser-Bibliotheken für Java:
  - Xerces-J: Apache,
  - Crimson: ab JDK 1.4.
- JAXP: Wrapper über Parser-Bibliotheken.

Beschreibung von XML-Dokumenten

- Möglichkeit 1: Document Type Definition (DTD)

```
<!ELEMENT person (name, age?, address+)>
<!ELEMENT age (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (street, place?, zip)>
<!ELEMENT place (#PCDATA)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT zip (#PCDATA)>
<!ATTLIST person category CDATA #REQUIRED>
```

person.dtd

```
<!DOCTYPE person SYSTEM "person.dtd">
<person category=“business”> ... </person>
```

person.xml

  - Nachteile:
    - Es kann lediglich festgelegt werden, dass Elemente andere Elemente, Text oder nichts enthalten dürfen.
    - Der Datentyp von Blättern kann nicht definiert werden.
- Möglichkeit 2: XML-Schema

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

Einfache Typen

- Ein XML-Schema enthält die Definition von
  - einfachen und
  - komplexen Typen
- Einfache Typen:
  - Definition eines Elements mit einem einfachen Type:

```
<element name="street" type="string">
```

  - XML:Schema definiert 44 Standardtypen (*"built-in types"*)
    - string
    - short
    - int
    - long
    - double
    - float
    - date
    - time
    - unsignedInt
    - decimal
    - base64Binary
    - …

Komplexe Typen: Sequenzen

- Komplexe Typen sind aus anderen (einfachen und komplexen) Typen zusammengesetzt.
- Sequenzen:
  - Fixe Anordnung von Elementen mit verschiedenem Typ.
  - Multiplizität der Elemente kann definiert werden.

```
<complexType name="personType">
  <sequence>
    <element name="name" type="string"/>
    <element name="age" type="unsignedShort"
      minOccurs="0" maxOccurs="1"/>
    <element name="address"
      type="tns:addressType" maxOccurs="unbounded"/>
  </sequence>
</complexType>
```

Komplexe Typen: *all*-Elemente/Attribute

- All-Elemente:
  - Anordnung von Elementen mit verschiedenem Typ, wobei Reihenfolge nicht vorgegeben wird.
  - Multiplizität: Elemente können höchstens einmal vorkommen.

```
<complexType name="addressType">
  <all>
    <element name="street" type="string"/>
    <element name="place" type="string" minOccurs="0"/>
    <element name="zip" type="unsignedShort"/>
  </all>
</complexType>
```

- Attribute:

```
<complexType name="personType">
  ...
  <attribute name="category" type="string" use="required"/>
</complexType>
```

Vererbung

- Erweiterung (*extension*): Hinzufügen von Elementen zum Basistyp.

```
<complexType name="studentType">
 <complexContent>
   <extension base="personType">
     <element name="id" type="StudentID">
   </extension>
 </complexContent>
</complexType>
```

- Einschränkung (*restriction*): Modifikation bzw. Weglassen von Elementen des Basistyps.

```
<simpleType name="ZipCode">
  <restriction base="int">
    minInclusive value="1000"
    maxExclusive value="10000"
  </restriction>
<simpleType>
```

```
<simpleType name="StudentID">
  <restriction base="string">
    pattern value=
              "se[mb]?[0-9]{5}"
</restriction>
<simpleType>
```

- Anwendung: Polymorphismus

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: Simple Object Access Protocol

- 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
  - EAI: ähnlich CORBA.
  - B2B-Kommunikation: ähnlich EDI-Standards.

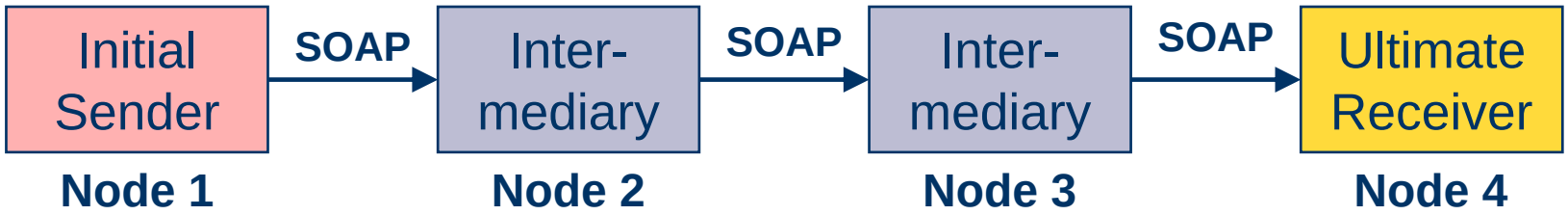
Struktur eines SOAP-Dokuments

```
<?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*-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 Body.

SOAP-Body

- Der SOAP-Body muss ein wohl-geformtes XML-Dokument sein.
- Body enthält *Daten* oder *Parameter eines entfernten Methodenaufrufs.*
- SOAP unterstützt 4 Nachrichten-Modi (*messaging modes*).

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

Nachrichten-Art *Literal*

- Document/Literal:
  - Zur Übertragung von Daten.
  - Body enthält ein Fragment eines XML-Dokuments.
  - Kann auf für RPC verwendet werden (Format für .NET Web-Services).

```
<soap:Body>
  <ns:person>
    <ns:name>Huber</ns:name>
    ...
  </ns:person>
</soap:Body>
```

- RPC/Literal:
  - Zur Darstellung entfernter Methodenaufrufe.
  - Body enthält Methodennamen und Methodenparameter.

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

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

Nachrichten-Art *Encoded*

- RPC/Encoded und Document/Encoded
  - Zur Darstellung entfernter Methodenaufrufe.
  - Definiert Abbildung von Datentypen auf XML-Schema.
  - Ermöglicht Repräsentierung von Objektgraphen.
  - Interoperabilitätsprobleme wegen vielfältiger Darstellungsmöglichkeiten.

```
<soap:Body>
  <ns:getGrades soap:encodingStyle="…/soap/encoding">
    <ns:id xsi:type="xsd:string>Streber</ns:id>
  </ns:getAge>
</soap:Body>
```

```
<soap:Body xmlns:enc="…/soap/encoding">
  <ns:getGradesResponse soap:encodingStyle="…/soap/encoding">
    <enc:Array enc:arrayType="xsd:short[2]">
      <enc:short>1</enc:short>
      <enc:short>2</enc:short>
    </enc:Array>
  </ns:getGradesResponse>
</soap:Body>
```

SOAP-Faults

- Fehler-Nachrichten (*soap faults*) werden an den Vorgängerknoten 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 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

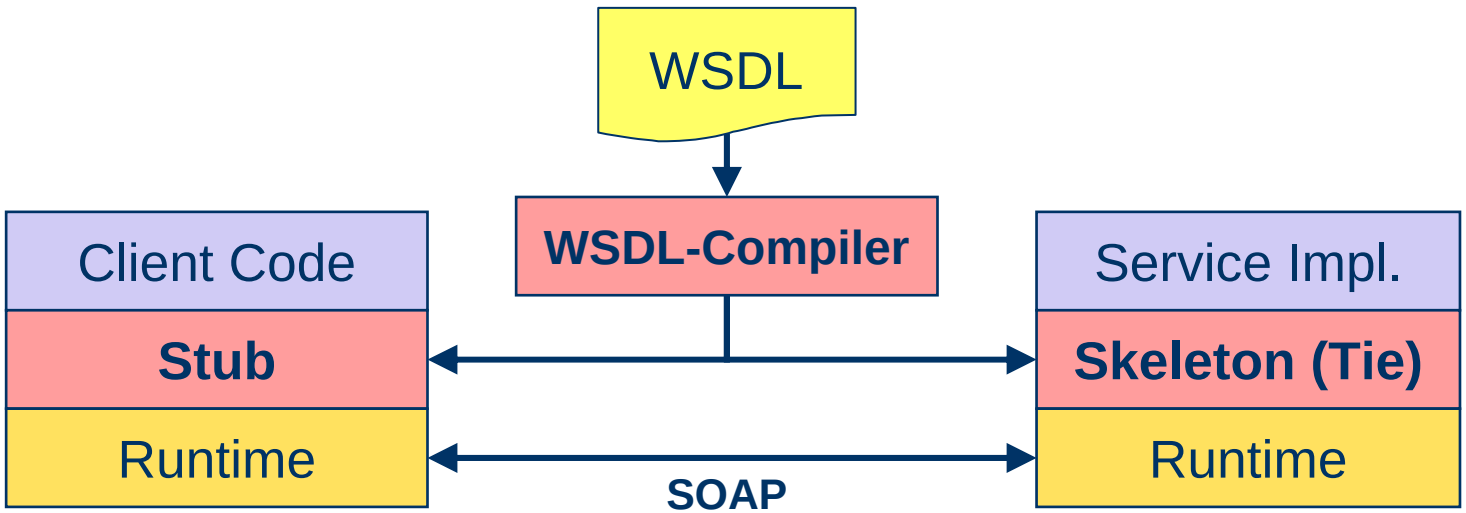
```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: nnn

<?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-)/Stub-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 eine *message*-Element definiert.

```
<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>
    <input name="id" message="tns:GetGradesRequest"/>
    <output name="grades" message="tns: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">
  <soapbind:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetGrades">
    <soapbind:operation soapAction="http://.../GetGrades"/>
    <input name="GetGradesRequest">
      <soapbind:body use="encoded" namespace=".../MyWS"/>
    </input>
    <output name="GetGradesResponse"> ... </output>
  </operation>
</binding>
```

WSDL: *service*

- Jedem *service*-Element können ein oder mehrer 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>
```