XML Vorlesung ETHZ, Sommersemester 2006

Web Services

Erik Wilde

27.6.2006

http://dret.net/lectures/xml-ss06/

Übersicht

- XML als Präsentationsschicht
  - Vergleich mit dem OSI Referenzmodell
  - *Abstract Syntax Notation One (ASN.1)*
- XML für RPC-Mechanismen
  - XML-RPC
  - *Simple Object Access Protocol (SOAP)*
- Web Services
  - SOAP, WSDL, UDDI, ...

OSI vs. Internet

- Open Sytems Interconnection (OSI)
  - ISO-Standardisierung um 1990 herum
  - führte ein Schichtenmodell ein
    - sieben Schichten für die modulare Modellierung
    - Modellierung von Diensten auf jeder Schicht
    - jede Schicht nutzt den Dienst der darunterliegenden Schicht
  - gedacht als komplette Ablösung des Internet
- Phase der Unsicherheit Anfang der 90er
  - Entscheidung OSI/Internet war unklar
  - letztlich Erfolg des Internet
    - einfache, bekannte und erprobte Technologien
    - stark zunehmender Erfolg durch Email und WWW

OSI Presentation Layer (OSI Layer 6)

- die sechste Schicht des OSI-Modells
  - Abstraktion von verschiedenen Datenmodellen
  - Lehre aus der "Network Byte Order" des Internet
    - Big Endian oder Little Endian?
    - das Internet benutzt Big Endian
  - Applikationen tauschen Daten abstrakt aus
  - der OSI-Stack sorgt für Kompatibilität
- OSI-Dienstschnittstellen benutzen ASN.1
  - Datenaustausch über ASN.1
  - Konvertierung in die Transfersyntax im OSI Stack
  - Aushandlung der Transfersyntax

Beispiel für ASN.1

```
PersonnelRecord ::= [APPLICATION 0] SET {
   name Name,
   title VisibleString,
   number EmployeeNumber,
   dateOfHire Date,
   nameOfSpouse Name,
   children SEQUENCE OF ChildInformation DEFAULT {} }
ChildInformation ::= SET {
   name Name,
   dateOfBirth Date }
Name ::= [APPLICATION 1] SEQUENCE {
   givenName VisibleString,
   initial VisibleString,
   familyName VisibleString }
EmployeeNumber ::= [APPLICATION 2] INTEGER
Date ::= [APPLICATION 3] VisibleString -- YYYY MMDD
```


ASN.1 Encoding Rules

- ASN.1 definiert eine abstrakte Syntax
  - aber wie werden konkrete Daten codiert?
  - dies ist die Aufgabe von Encoding Rules
    - Abbildung ASN.1 abstrakte Syntax ↔ Octet Stream
- *Basic Encoding Rules (BER)*
  - einfachste Codierung: Tag/Length/Value
- *Packed Encoding Rules (PER)*
  - effiziente Codierung am Datentyp orientiert
  - u.U. Ausnutzung von jedem Bit (aligned vs. unaligned)
- *Distinguished Encoding Rules (DER)*
  - Untermenge von BER für eindeutige Codierung

Beispiel für XML Instanz

```
<?xml version="1.0" encoding="UTF-8"?>
<PersonnelRecord>
     <name>
          <givenName>John</givenName>
          <initial>P</initial>
          <familyName>Smith</familyName>
     </name>
     <title>Director</title>
     <number>51</number>
     <dateOfHire>19710917</dateOfHire>
     <nameOfSpouse>
          <givenName>Mary</givenName>
          <initial>T</initial>
          <familyName>Smith</familyName>
     </nameOfSpouse>
     <children>
          <ChildInformation>
               <name>
                    <givenName>Ralph</givenName>
                    <initial>T</initial>
                    <familyName>Smith</familyName>
               </name>
               <dateOfBirth>19571111</dateOfBirth>
          </ChildInformation>
          <ChildInformation>
               <name>
                    <givenName>Susan</givenName>
                    <initial>B</initial>
                    <familyName>Jones</familyName>
               </name>
               <dateOfBirth>19590717</dateOfBirth>
          </ChildInformation>
     </children>
</PersonnelRecord>
```


ASN.1 XML Encoding Rules (XER)

- klassische ASN.1 Codierungen sind binär
  - Kommunikation zwischen OSI Layers
  - binäre Codierungen sind effizienter
- ASN.1 definiert eine Schemasprache
  - XML definiert mehrere Sprachen
    - DTDs für Schemas (langsam abgelöst durch XML Schema)
    - XML als Sprache für Instanzen (d.h. Dokumente)
  - Abbildung von ASN.1 auf XML ist möglich
- XER als Schnittstelle zwischen ASN.1 und XML
  - Codierung von ASN.1 Instanzen als XML Dokument
  - unglücklicherweise nicht kompatibel mit XML Schema

XML ASN.1 Encoding Rules

- XML definiert Schema und Instanz
  - DTDs für das Schema (oder auch XML Schema)
  - XML Dokumente für die Instanzen
- XML Dokumente sind extrem voluminös
  - Relikt aus der Dokumentenverarbeitung mit SGML
  - Tags verursachen sehr hohe Redundanz
- alternative Codierungen wären möglich
  - im einfachsten Fall Zip von einem XML Dokument
    - Nachteil: die Struktur ist nicht mehr sichtbar
  - XML-spezifische Kompression
  - Abbildung auf ASN.1 und dann PER anwenden

Vergleich ASN.1 und XML Encoding

Unaligned PER	84 Bytes
Aligned PER	94 Bytes
BER with definitive length	136 Bytes
BER with indefinitive length	161 Bytes
XER	1420 Bytes

XML vs. ASN.1 (Extremfall)

```
Simple-example ::= SEQUENCE {
   firstfield INTEGER (0..7),
   secondfield BOOLEAN,
   thirdfield INTEGER (8..11),
   fourthfield SEQUENCE {
     fourA BOOLEAN,
     fourB BOOLEAN } }
```

PER	1 Byte
BER	19 Bytes
XER	183 Bytes

```
<Simple-example>
   <firstfield>5</firstfield>
   <secondfield>false</secondfield>
   <thirdfield>10</thirdfield>
   <fourthfield>
       <fourA>false</fourA>
       <fourB>false</fourB>
   </fourthfield>
</Simple-example>
```


Verhältnis von XML und ASN.1

- konkurrierende Schema-Sprachen
  - XML mit DTDs oder XML Schema
  - ASN.1 mit der ASN.1 Specification Language
- Vorteile von XML
  - wesentlich populärer
  - viel verbreitete und freie Software
    - beginnt sich als Standard-Software zu etablieren
- Vorteile von ASN.1
  - Kompatibilität mit XML über XER
  - kompaktere Codierungen mit BER oder PER

Microsoft's .NET und XML

- Datenmodell von Programmiersprachen
  - oftmals Datenaustausch über den Stack
  - Datenformat durch Compiler implementiert
- Datenmodell fest in der Sprache verankert
  - einfache Datentypen (Integer, Boolean, ...)
  - komplexe Datentypen (sprachspezifisch)
- .NET definiert ein gemeinsamen Datenmodell
  - mehrere Sprachen, ein Datenmodell (XML)
  - dynamisches Zusammenspiel vieler Sprachen
    - aus einer Java-Routine eine C Prozedur aufrufen
    - diese später mit einer neueren C# Implementierung ersetzen

XML-RPC

- extrem einfacher RPC Mechanismus
  - Codierung von Datentypen in XML
    - vordefinierte Typenmenge
  - Austausch über XML und HTTP
- keine weitergehende Infrastruktur
  - Aufrufsemantik unklar
  - Frage der Serviceauffindung unklar
- eher Fingerübung als komplette Infrastruktur
  - sehr einfach zu verwenden
  - sehr klare Limitierungen für ernsthafte Anwendungen

XML-RPC Request

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 177

<?xml version="1.0"?>
<methodCall>
  <methodName>examples.getState</methodName>
  <params>
    <param>
      <value><i4>41</i4></value>
    </param>
  </params>
</methodCall>
```


XML für Web Services

- was sind Web Services?
  - Dienste basierend auf Web-Technologien
  - nicht unbedingt inhaltlich auf das Web bezogen
    - viele Anwendungen in geschlossenen Benutzergruppen
  - XML als Datenformat
- Web Services für B2B Anwendungen
  - allgemein verfügbare Software
  - allgemein angewendete Technologie
  - allgemein vorhandenes Know-How
  - allgemein akzeptiertes Vorgehen

SOAP 1.1

- *Simple Object Access Protocol (SOAP)*
- Message Modell
  - Header
  - Body
  - Attachments
- Unabhängig vom Transport-Protokoll
  - Bindings für HTTP
  - SMTP, RMI, JMS etc. denkbar aber nicht in SOAP definiert
- RPC oder one way (Messaging)

SOAP 1.2

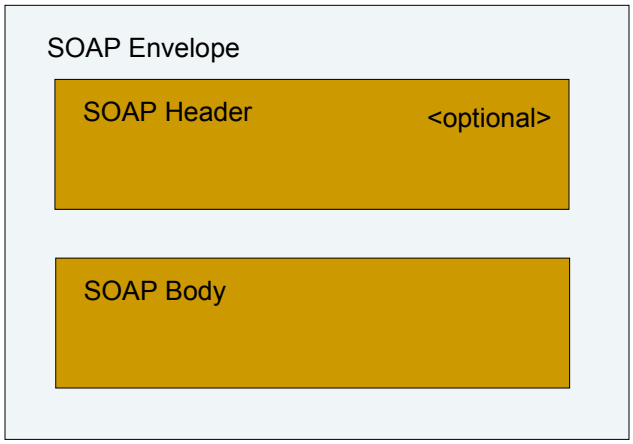
- aktuelle Version von SOAP
- nicht überall implementiert
- anderes Datenmodell als SOAP 1.1
  - SOAP 1.1 basiert auf XML 1.0
  - SOAP 1.2 basiert auf dem Infoset
- stärkere Betonung von Messaging
  - *Representational State Transfer (REST)* vs. RPC
- bessere Integration von binären Daten
  - Transport als Attachments von SOAP Messages
  - transparente Integration in das Infoset

SOAP Request

```
POST /EchoService/EchoService.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: 369
SOAPAction: "http://bsquare.ws/echoservice/Echo"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <Echo xmlns="http://bsquare.ws/echoservice/">
      <echoString>hello</echoString>
    </Echo>
  </soap:Body>
</soap:Envelope>
```


SOAP Aufbau



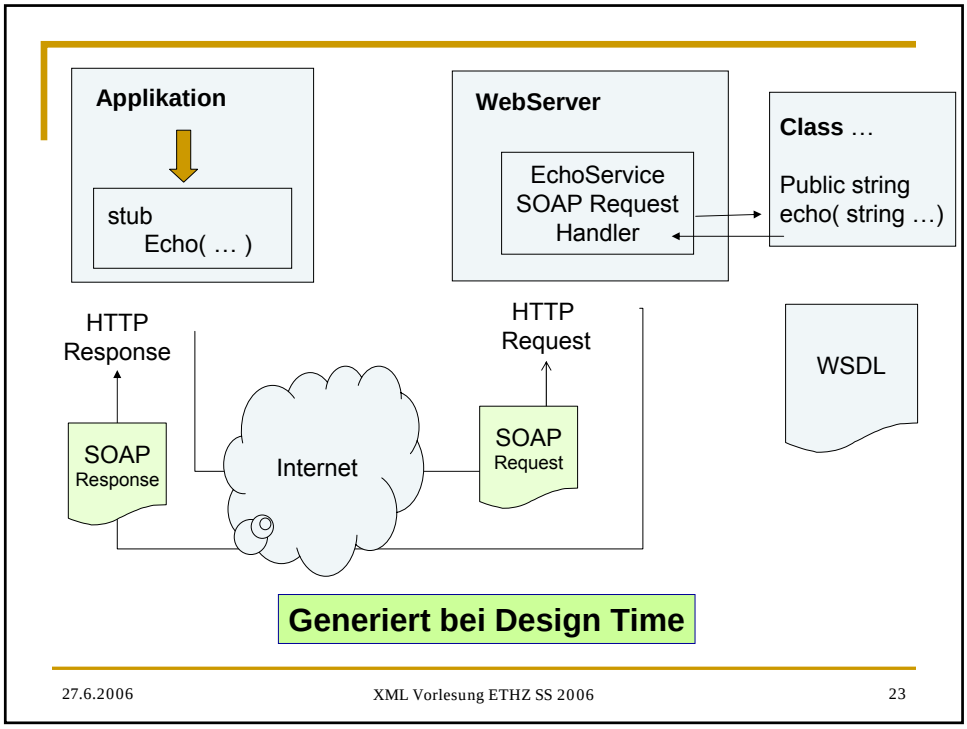
SOAP Header und Body

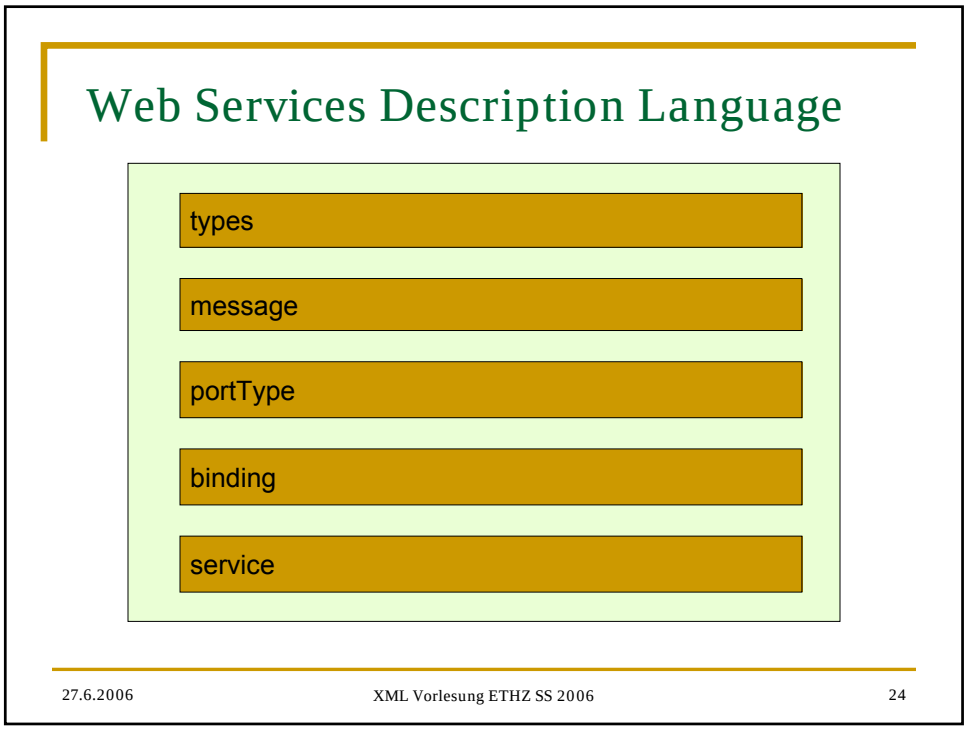
- SOAP Envelope umfasst alles
- SOAP Header ist Information auf dem Umschlag
  - optional (Umschlag kann leer sein)
  - Routing-Information (Absender, Pfad, ...)
- SOAP Body sind die Nutzdaten
  - muss existieren
  - eigentlicher Inhalt der SOAP Message
  - kann beliebig komplex strukturiert sein
    - ist immer eine XML-Struktur
    - kann auf binäre Daten verweisen (SOAP Attachments)

SOAP Response

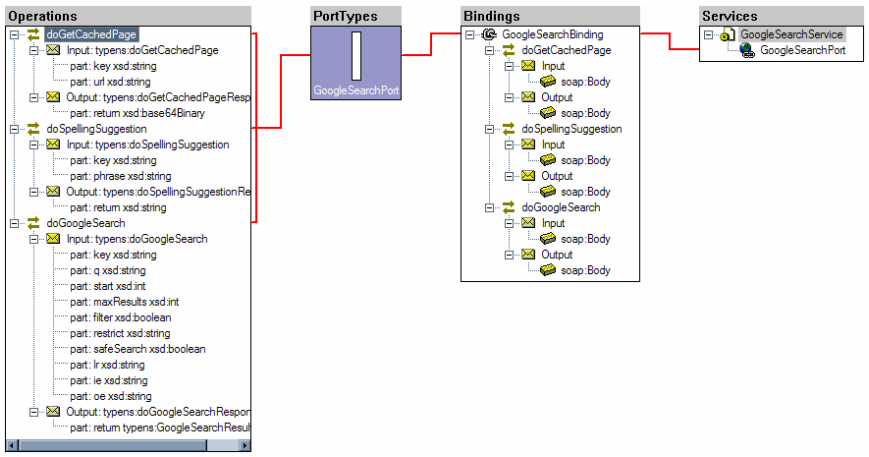
```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Fri, 01 Feb 2002 16:21:09 GMT
Cache-Control: private, max-age=0
Content-Type: text/xml; charset=utf-8
Content-Length: 358
```

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <EchoResponse xmlns="http://bsquare.ws/echoservice/">
    <EchoResult>Echo [17:21] hello</EchoResult>
    </EchoResponse>
  </soap:Body>
</soap:Envelope>
```


Applikation
stub
Echo(…)
WebServer
EchoService
SOAP Request
Handler
Class …
Public string
echo(string …)
HTTP
Response
HTTP
Request
WSDL
SOAP
Response
Internet
SOAP
Request
Generiert bei Design Time
27.6.2006
XML Vorlesung ETHZ SS 2006
23


Web Services Description Language
types
message
portType
binding
service
27.6.2006
XML Vorlesung ETHZ SS 2006
24

WSDL Visualisierung (Google WSDL)



Beispiel WSDL Types (Google)

```
<wsdl:types>
 <xsd:schema targetNamespace="urn:GoogleSearch">
  <xsd:complexType name="GoogleSearchResult">
   <xsd:all>
    <xsd:element name="documentFiltering" type="xsd:boolean"/>
    <xsd:element name="searchComments" type="xsd:string"/>
    <xsd:element name="estimatedTotalResultsCount" type="xsd:int"/>
    <xsd:element name="estimateIsExact" type="xsd:boolean"/>
    <xsd:element name="resultElements" type="typens:ResultElementArray"/>
    <xsd:element name="searchQuery" type="xsd:string"/>
    <xsd:element name="startIndex" type="xsd:int"/>
    <xsd:element name="endIndex" type="xsd:int"/>
    <xsd:element name="searchTips" type="xsd:string"/>
    <xsd:element name="directoryCategories"
                 type="typens:DirectoryCategoryArray"/>
    <xsd:element name="searchTime" type="xsd:double"/>
   </xsd:all>
  </xsd:complexType>
  ...
```

Beispiel WSDL Message (Google)

```
<message name="doGoogleSearch">
  <part name="key" type="xsd:string"/>
  <part name="q" type="xsd:string"/>
  <part name="start" type="xsd:int"/>
  <part name="maxResults" type="xsd:int"/>
  <part name="filter" type="xsd:boolean"/>
  <part name="restrict" type="xsd:string"/>
  <part name="safeSearch" type="xsd:boolean"/>
  <part name="lr" type="xsd:string"/>
  <part name="ie" type="xsd:string"/>
  <part name="oe" type="xsd:string"/>
</message>
<message name="doGoogleSearchResponse">
  <part name="return" type="typens:GoogleSearchResult"/>
</message>
...
```


Beispiel WSDL PortType (Google)

```
<portType name="GoogleSearchPort">
 <operation name="doGetCachedPage">
  <input message="typens:doGetCachedPage"/>
  <output message="typens:doGetCachedPageResponse"/>
 </operation>
 <operation name="doSpellingSuggestion">
  <input message="typens:doSpellingSuggestion"/>
  <output message="typens:doSpellingSuggestionResponse"/>
 </operation>
 <operation name="doGoogleSearch">
  <input message="typens:doGoogleSearch"/>
  <output message="typens:doGoogleSearchResponse"/>
 </operation>
</portType>
```

Beispiel WSDL Binding (Google)

```
<binding name="GoogleSearchBinding"
         type="typens:GoogleSearchPort">
 <soap:binding style="rpc"
               transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="doGetCachedPage">
  <soap:operation soapAction="urn:GoogleSearchAction"/>
  <input>
   <soap:body use="encoded"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="urn:GoogleSearch"/>
  </input>
  <output>
   <soap:body use="encoded"
         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
         namespace="urn:GoogleSearch"/>
  </output>
 </operation>
 ...
```

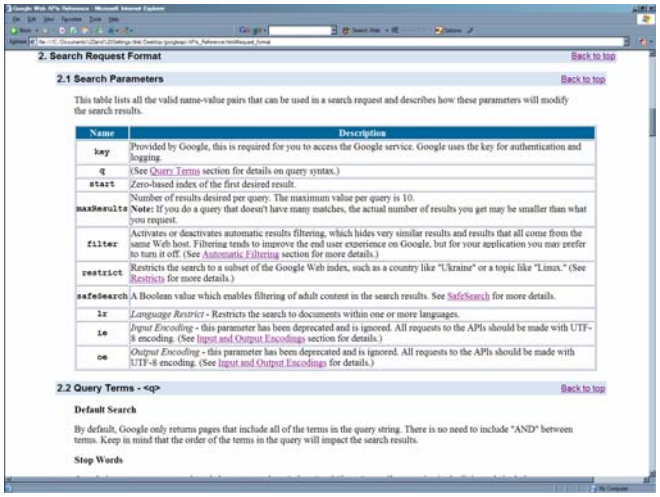

Beispiel WSDL Service (Google)

```
<service name="GoogleSearchService">
 <port name="GoogleSearchPort"
       binding="typens:GoogleSearchBinding">
  <soap:address location="http://api.google.com/search/beta2"/>
 </port>
</service>
```

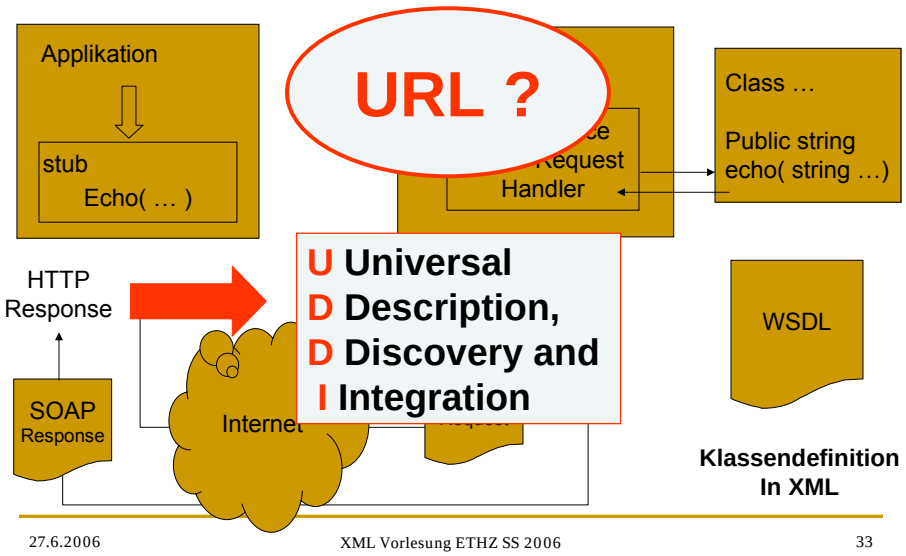
Google SOAP Request

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <SOAP-ENV:Body>
        <m:doGoogleSearch xmlns:m="urn:GoogleSearch"
        SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
                <key xsi:type="xsd:string">String</key>
                <q xsi:type="xsd:string">String</q>
                <start xsi:type="xsd:int">0</start>
                <maxResults xsi:type="xsd:int">0</maxResults>
                <filter xsi:type="xsd:boolean">1</filter>
                <restrict xsi:type="xsd:string">String</restrict>
                <safeSearch xsi:type="xsd:boolean">1</safeSearch>
                <lr xsi:type="xsd:string">String</lr>
                <ie xsi:type="xsd:string">String</ie>
                <oe xsi:type="xsd:string">String</oe>
        </m:doGoogleSearch>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```


Semantik des Google SOAP Request

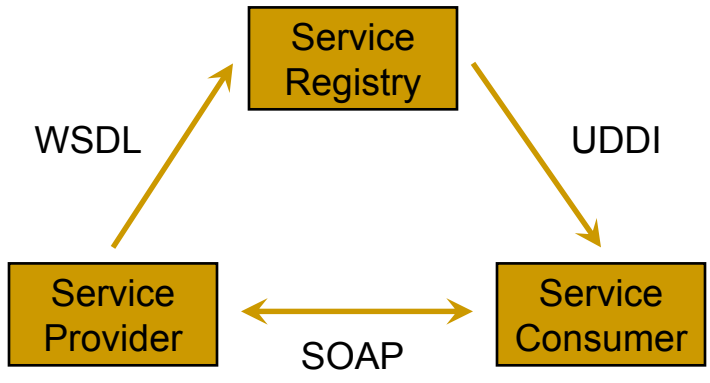


Web Service Architektur



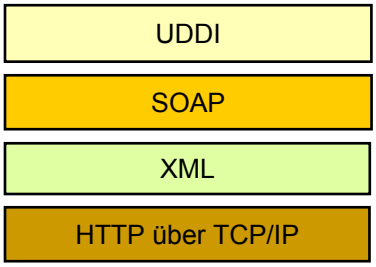
27.6.2006 XML Vorlesung ETHZ SS 2006 33

Teilnehmer an einem B2B Szenario



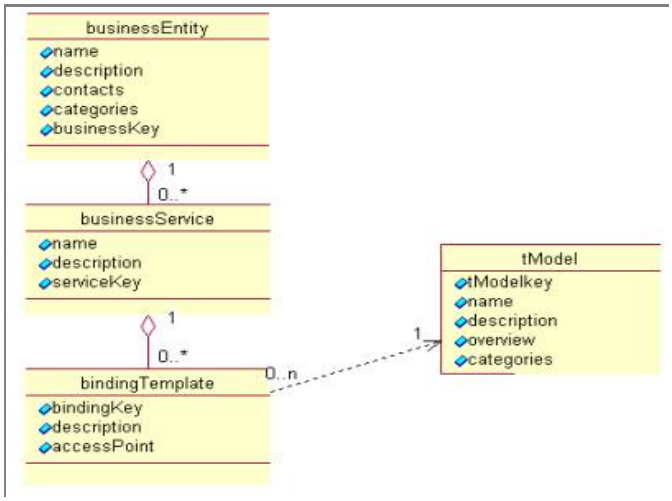
27.6.2006 XML Vorlesung ETHZ SS 2006 34

UDDI



- Universal Description, Discovery and Integration
- Verzeichnis und Klassifikation von Web Services
- selbst ein Web Service

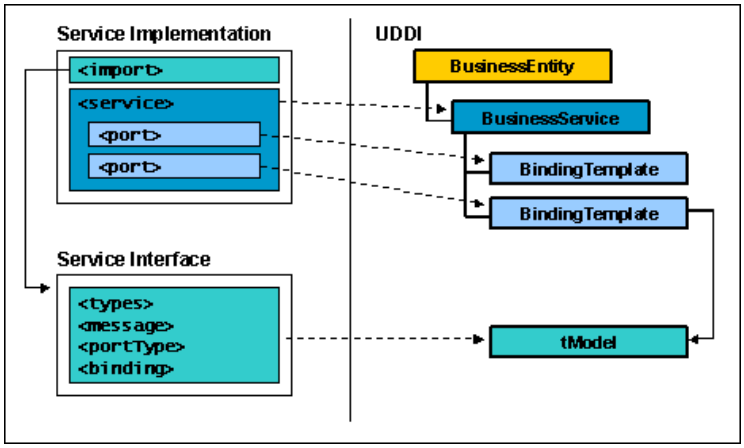
UDDI Datenstruktur



UDDI Komponenten

- businessEntity
  - Firma und/oder Person, die Dienste anbietet
- businessService
  - logische Klassifizierung von Diensten
- bindingTemplate
  - verschiedene Arten, einen Dienst zu benutzen
- tModel
  - Implementierung eines Dienstes
  - u.U. Verweis auf Beschreibung eines Dienstes
  - ermöglicht das Auffinden von bekannten Diensten

Abbildung WSDL → UDDI



aus: http://www-106.ibm.com/developerworks/webservices/library/ws-wsdl/

Zusammenfassung

- XML als Präsentationsschicht des Internet
  - eher gewachsen als designt
  - für viele Anwendungen mässig geeignet
    - Datenvolumen in B2B-Szenarien ist problematisch
- ASN.1 als 15 Jahre alte akademische Lösung
  - in kleinen Bereichen erfolgreich
  - als allgemeine Präsentationsschicht ein Misserfolg
- XML und ASN.1 als verschiedene Ansätze
  - XML als der Zufallserfolg aus der Dokumentenwelt
  - ASN.1 als durchgeplant, aber zu kompliziert