# *7363 - Web-basierte Anwendungen*

Eine Vertiefungsveranstaltung
mit Schwerpunkt auf XML-Technologien

# *WSDL*

*Web Services Description Language*

# WSDL

- Warum eine Beschreibungssprache für Web Services?
  - Dokumentation der Schnittstelle
    - Systematischere, strukturierte Entwicklung und Pflege von WS
    - Analogien: C-Headerdateien, IDL, *reflection, type libraries*
  - Grundlage für Code-Generierung
    - Entlastung der Client-Entwicklung
  - Grundlage für flexiblen Einsatz
    - Schnittstellenänderungen lassen sich z.T. automatisch einarbeiten
    - Neue WS lassen sich schnell und leicht verwenden
    - Zusammenspiel mit UDDI: Finden, konfigurieren, nutzen

# WSDL

- Entwicklung von WSDL
  - IBM: NASSL, 2000
    - *Network Accessibility Service Specification Language*
    - Verwendete XML zur Beschreibung von WS *Interfaces*
    - Verwendet XML Schema zur Beschreibung von Datentypen
  - Microsoft: SCL, 2000
    - *Service Contract Language*
    - Verwendete XML zur Beschreibung von WS *Interfaces*
    - Verwendet XDR (Microsoft's "XML *Data Reduced*") zur Beschreibung von Datentypen
    - Nachfolger: SDL, zusammen mit Visual Studio.NET
  - Problem dabei:
    - Interoperabilität auf der Beschreibungsebene??
  - Lösung: WSDL

# WSDL

- Entwicklung von WSDL
  - Microsoft, IBM, Ariba: Erste WSDL-Version
    - Nutzung von XML Schema (bzw. Vorläufer), SOAP, MIME
  - WSDL 1.1, 15. März 2001
    - W3C Note: http://www.w3.org/TR/wsdl
  - WSDL 2.0
    - Status: Candidate Recommendation
      (0: *Primer*, 1: *Core Language*, 2: *Adjuncts*)
    - Zuletzt aktualisiert: 27. März 2006 (siehe http://www.w3.org/2002/ws/desc/)
    - Neu: Vererbungskonzept, per Attribut "extends"
    - Verändert: Begriff / Element "port" wird ersetzt durch "interface"

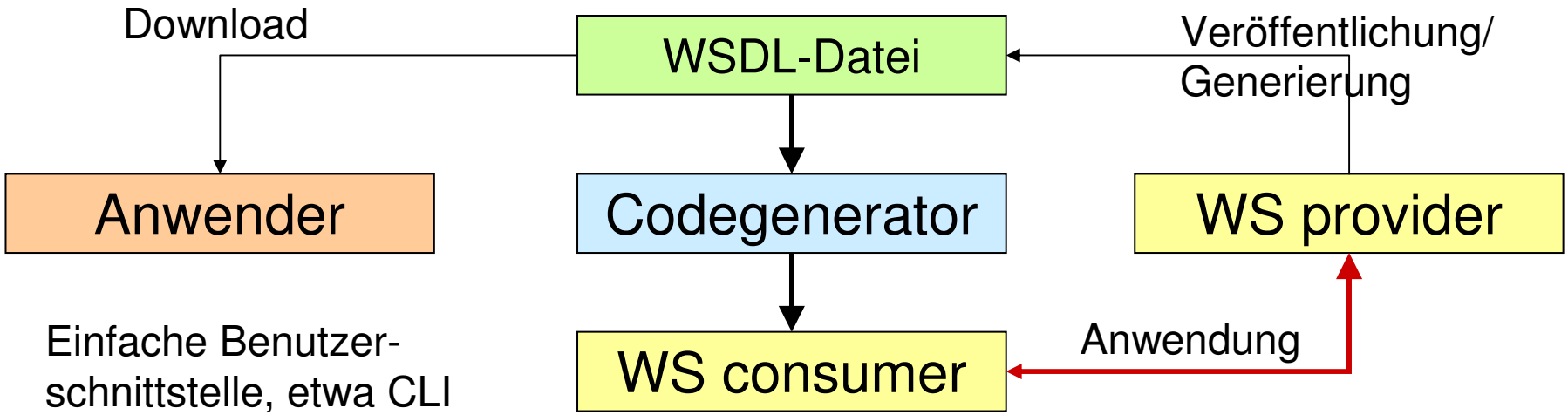
# WSDL

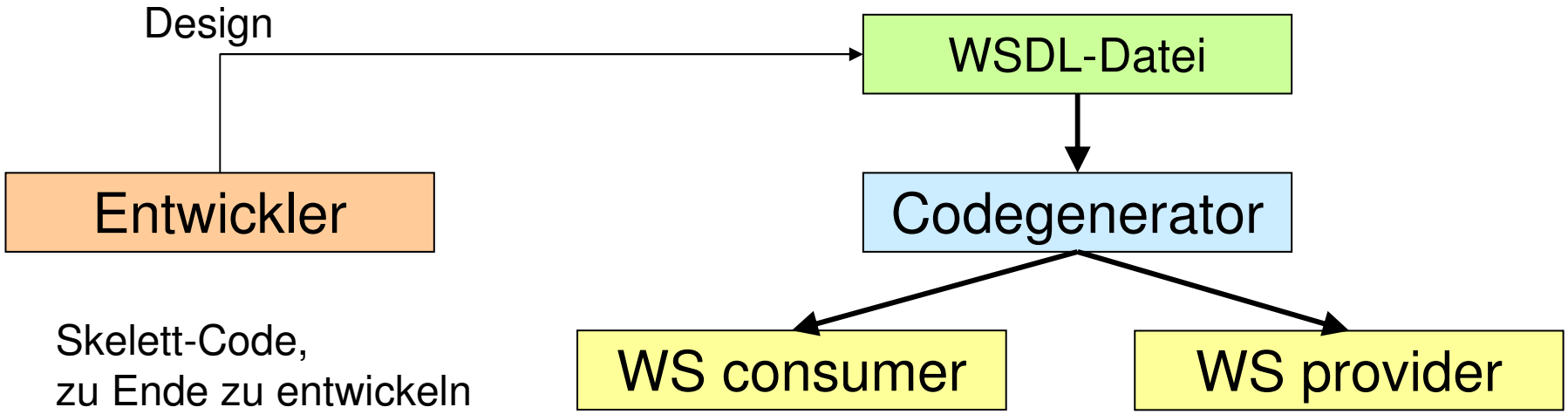
- WSDL 1.1
  - Dokumentation: http://www.w3.org/TR/wsdl
  - Namensraum: http://schemas.xmlsoap.org/wsdl/

- Ansatz:
  - Beschreibung von WS als Sammlung von Netzwerk-Endpunkten (*ports* / *interfaces*)
  - Trennung der abstrakten Interface- und Nachrichtentyp-Beschreibung von der konkreten Implementierung (im Sinne von: *bindings*, *encoding*, einzelne Nachrichten)
  - XML-Dokumententyp **`definitions`** als Grundlage
  - Ausgiebiger Gebrauch von XML Schema!
    - Teile des WSDL-Schemas sind von "Schema" übernommen, etwa die Elemente "include", "import" & das Attribut "targetNameSpace"

# WSDL

- WSDL: Anwendungsszenarien



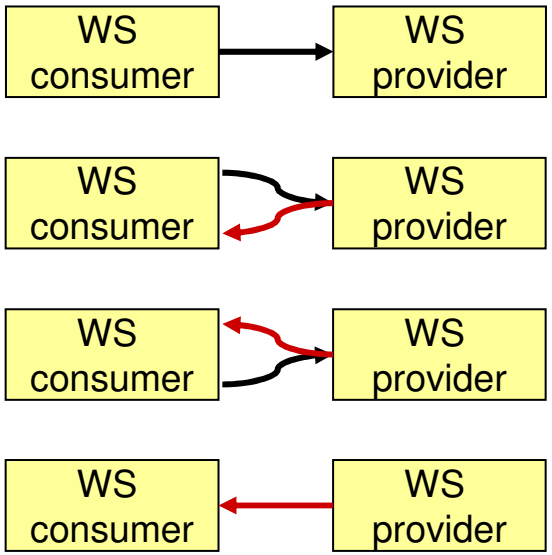
# WSDL

- WSDL 2.0
  - Dokumentation: http://www.w3.org/TR/wsdl20 (*core language*)
  - (vorl.)Namensraum: http://www.w3.org/2006/01/wsdl
  - Status: CR, zuletzt aktualisiert: März 2006
  - Geändert:
    - *Port → Interface*
  - Neu:
    - Vererbungskonzept! Attribut "extends" in Element "interface"

- Empfehlung
  - Noch zu früh für den praktischen Einsatz, aber:
  - Entwicklung beobachten!

# WSDL

- Nachtrag zu SOAP: **Operationstypen**
  - Je nach Anzahl und Reihenfolge von **input**-, **output**-, und **fault**-Elementen, die zwischen zwei SOAP-Knoten ausgetauscht werden, unterscheidet man folgende 4 abstrakte Operationstypen:
  - *One-way*
    - WS *consumer* sendet
  - *Request-response*
    - WS *consumer* sendet, WS *provider* antwortet
  - *Solicit-response*
    - WS *provider* agiert, WS *consumer* reagiert
  - *Notification*
    - WS *provider* agiert




  - In der Praxis dominieren die Typen "*One-way*" und "*Request-response*". (Publikumsfrage: Warum wohl?)

# *WSDL: Eine beispiel-orientierte Einführung*

Die WSDL-Datei vom Babelfish-Service
mit Auszügen aus dem Google-API

# WSDL 1.1: Grobstruktur

| | |
|---|---|
| • ***definitions*** | WSDL-Dokumentenelement |
| – ***types*** | Optionaler Container |
| • ***schema*** | Schemata für Dokumenten- (EDI-) Modus |
| – ***message*** | Abstrakter Teil: Nachrichtenaufbau |
| • ***part*** | Teilnachricht, etwa …*Request*, …*Response* |
| – ***portType*** | Abstrakter Teil: Zuordnung der Nachrichtenarten |
| • ***operation*** | zu logischen *Ports/Interfaces* |
| – ***binding*** | Konkreter Teil: Anbindung eines *Port-* |
| • ***operation*** | Typen an Transportprotokoll & Codierung |
| – ***service*** | Konkreter Teil: |
| • ***port*** | Assoziation eines konkreten *Ports* mit e. *Binding*. Achtung: Typen müssen zueinander passen. |

# WSDL

- Lernen von WSDL am Beispiel
  - "Babelfish-Service"
    - Grundlagen: SOAP 1.1, WSDL 1.1, RPC-Stil, HTTP *binding*
    - Input: Zwei Strings
      - String mit zu übersetzendem Text, max. 5k Zeichen
      - Quell- und Zielsprache, ISO-codiert, etwa: "en_de"
    - Output (Rückgabewert): Ein String
      - Der übersetzte Text
  - "Google-Service"
    - Analog zu Babelfish, aber mit komplexeren Datentypen
  - Vorgehen
    - Analyse der WSDL-Dateien dieser Dienste (*bottom-up*)
    - Erstellung einer Babelfish *Client*-Anwendung
    - Test

# WSDL

- Die WSDL-Datei zum WS "Babelfish"
  - Das Dokumentenelement
    - Wenig Bemerkenswertes, nur viele Namensraum-Deklarationen
    - In rot: Default-Namensraum (WSDL-Namensraum)

```
<?xml version="1.0"?>
<definitions name="BabelFishService"
   xmlns:tns="http://www.xmethods.net/sd/BabelFishService.wsdl"
   targetNamespace="http://www.xmethods.net/sd/BabelFishService.wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns="http://schemas.xmlsoap.org/wsdl/">

<!-- Hier der eigentliche Inhalt, s.u. -->

</definitions>
```

# WSDL

- *Web Service*-Beschreibung
  - Das "service"-Element

```
<service name="BabelFishService">
   <documentation>Translates text of up to 5k in length, between a
   variety of languages.</documentation>
   <port name="BabelFishPort" binding="tns:BabelFishBinding">
     <soap:address location=
        "http://services.xmethods.net:80/perl/soaplite.cgi"/>
   </port>
</service>
```

# WSDL

- *Web Service*-Beschreibung
  - Das "**service**"-Element
    - Der Dienst erhält einen Namen
    - Ihm wird ein (zunächst abstraktes) Interface (Port) zugewiesen
    - Das "binding" dazu erfolgt
      - durch Angabe eines Binding-Typs (zunächst nur ein Name)
      - durch Angabe einer konkreten Adresse (hier: ein URL)
  - Das Unter-Element "**documentation**"
    - Universelles Unter-Element - nutzen Sie es!
    - Es wirkt als Container, nimmt also neben *char data* auch beliebige (eigene) Unter-Elemente sowie Attribute auf.

# WSDL

- *Web Service*-Beschreibung
  - Das "**binding**"-Element

```
<binding name="BabelFishBinding" type="tns:BabelFishPortType">
   <soap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="BabelFish">
     <soap:operation soapAction=
        "urn:xmethodsBabelFish#BabelFish"/>
     <input><soap:body
                use="encoded" namespace="urn:xmethodsBabelFish"
                encodingStyle=
                 "http://schemas.xmlsoap.org/soap/encoding/"/>
     </input>
     <output><soap:body
                use="encoded" namespace="urn:xmethodsBabelFish"
                encodingStyle=
                  "http://schemas.xmlsoap.org/soap/encoding/"/>
     </output>
   </operation>
</binding>
```

# WSDL

- *Web Service*-Beschreibung
  - Das "**binding**"-Element
    - Das Binding erhält einen Namen
    - Ihm wird ein (noch zu spezifizierender) Interfacetyp (Porttyp) zugewiesen
    - Einzelheiten beschreiben Unter-Elemente
  - Das Unter-Element "**soap:binding**"
    - Hier werden SOAP-Stil (hier „rpc“) und Transportprotokoll (HTTP, erkennbar an einem reservierten Namensraum-URL) festgelegt.
    - Attribut „style“: Werte sind
      - „**rpc**“ SOAP-RPC Stil, eine Konvention zum Inhalt von „body“: Je ein Unterelement, abgeleitet von Methodenname (*methname*, *methname*Response, Fault)
      - „**document**“ Dokumenten- oder EDI-Stil: Inhalt von „body“ kann frei gewählt werden, ggf. beschrieben durch Schema in „**types**“

# WSDL

- *Web Service*-Beschreibung
  - Das (hier einzige) Unter-Element "**operation**"
    - Beschreibung, nach welchen Regeln Übergabeparameter „verpackt“ sind. Hier: SOAP *encoding* gemäß SOAP 1.1.
    - Attribut „use“: Werte sind
      - **encoded** Gemäß SOAP 1.1, 1.2 oder anderen Verfahren (Ruby scheint eine ASP.NET-Variante zu kennen). Parameter werden gemäß Inhalt von „encodingStyle“ codiert!
      - **literal** Keine Codierung – Dokumententeile werden einfach durchgereicht und nicht als Parameter interpretiert.
  - Die Kombinationen der Attribute „style“ und „use“
    - rpc / encoded — Häufigster Fall: SOAP-RPC, meist via http
    - document / literal — Typisch für Dokumentenmodus
    - rpc / literal — Grundsätzlich möglich
    - document / encoded — Grundsätzlich möglich

# WSDL

- *Web Service*-Beschreibung
  - Das "**portType**"-Element

```
<portType name="BabelFishPortType">
   <operation name="BabelFish"
       parameterOrder="translationmode sourcedata" >
       <input message="tns:BabelFishRequest" />
       <output message="tns:BabelFishResponse" />
   </operation>
</portType>
```

# WSDL

- *Web Service*-Beschreibung
  - Das "**portType**"-Element
    - Hier wird nun der Interfacetyp näher beschrieben
    - Ihm werden evtl. mehrere Operationen zugewiesen.
  - Das Unter-Element "**operation**"
    - Wir kennen es schon als Unterelement von "binding"
    - Im vorliegenden Kontext benennen seine Unterelemente den eigentlichen Aufbau der Body-Inhalte (hier: sowohl *input* als auch *output*, da wir ein RPC-Szenario verwenden)
    - Die Inhalte der Attribute "message" sind Elementnamen des Namensraums, der den für diesen Service verwendeten Elementen zugewiesen wurde.
    - Das Attribut „parameterOrder“ ist optional und nicht im originalen BabelFish-Beispiel enthalten. Es bringt eine determinierte Reihenfolge in die sonst hash-artige Parameterliste (parts).

# WSDL

- *Web Service*-Beschreibung
  - Das "**message**"-Element

```
<message name="BabelFishRequest">
   <part name="translationmode" type="xsd:string"/>
   <part name="sourcedata" type="xsd:string"/>
</message>

<message name="BabelFishResponse">
   <part name="return" type="xsd:string"/>
</message>
```

# WSDL

- *Web Service*-Beschreibung
  - Das "**message**"-Element
    - Bei SOAP-RPC existiert nur ein Unter-Element von "Body" und dies trägt den Namen der Methode/Prozedur.
    - Das Element "message" beschreibt genau dieses Unter-Element von "Body", sowohl im *request*- als auch im *response*-Fall.
  - Das Unter-Element "**part**"
    - Unter-Elemente von SOAP-RPC Methodenelementen sind bekanntlich die Teile eines *struct*. Jedes solche Teil wird hier als "*part*" beschrieben, mit Namen und Datentyp.
    - Interessant ist der Datentyp: In einfachen Fällen ist er einer der XML Schema-Datentypen, aber auch komplexe Datentypen aus eigenen Schemata (s.u.) werden hier oftmals verwendet.

# *WSDL: Weitere Elemente*

types: Komplexe Datentypen

(Auszüge aus der WSDL-Datei von Google)

import & include

# WSDL

- Beschreibung eines Web Service
  - Das "**types**"-Element

```
<types>
  <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
              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>  <!-- etc. -->
```

# WSDL

- Beschreibung eines Web Service
  - Das "**types**"-Element

```
<xsd:complexType name="ResultElement">
  <xsd:all>
    <xsd:element name="summary" type="xsd:string"/>
    <xsd:element name="URL"     type="xsd:string"/>
    <xsd:element name="snippet" type="xsd:string"/>
    <xsd:element name="title"   type="xsd:string"/>
    <xsd:element name="cachedSize"
                                type="xsd:string"/>
    <xsd:element name="relatedInformationPresent"
                                type="xsd:boolean"/>
    <xsd:element name="hostName" type="xsd:string"/>
    <xsd:element name="directoryCategory"
                 type="typens:DirectoryCategory"/>
    <xsd:element name="directoryTitle"
                                type="xsd:string"/>
  </xsd:all>
</xsd:complexType> <!-- Fortsetzung -->
```

# WSDL

- Beschreibung eines Web Service
  - Das "**types**"-Element

```
    <xsd:complexType name="ResultElementArray">
      <xsd:complexContent>
        <xsd:restriction base="soapenc:Array">
          <xsd:attribute ref="soapenc:arrayType"
              wsdl:arrayType="typens:ResultElement[]"/>
        </xsd:restriction>
      </xsd:complexContent>
    </xsd:complexType>
  <!-- usw., schließlich: -->
  </xsd:schema>
</types>
```

# WSDL

- Beschreibung eines Web Service
  - Das "**types**"-Element
    - Dies ist eine Hülle für Schemadokumente
    - Entweder schreibt man gleich ein (kleines) Schemadokument hier hinein, oder man importiert es.
    - Beispiele finden Sie in der WSDL-Datei von Google!
  - Die Elemente "**include**" und "**import**"
    - Sie gestatten (analog zu Schema) die Verteilung eines Schema-Dokuments auf mehrere Dateien bzw. Ressourcen.
    - Verwenden Sie "import", um WSDL-Bestandteile aus anderen Namensräumen einzubeziehen.
    - "include" (nur WSDL 2.0!) wirkt dagegen wie das direkte Einbeziehen eines Textblocks.

# *WSDL: Test & Demo*

# WSDL

- Test: Erzeugung einer *Babelfish-Client*-Anwendung
  - Werkzeug / Entwicklungsumgebung:
    - Ruby 1.8.4 oder 1.8.5, SOAP4R 1.5.5 (das *separate* Paket!)
  - Code-Generator:
    - **`wsdl2ruby.rb`** aus dem *separat* erhältlichen SOAP4R-Paket
  - Vorgehen:
    - Dateien generieren
    - Client-Datei mit einem minimalen CLI (*command line interface*) ausstatten
    - Ggf. Umgebungsvariablen setzen, hier:

```
$ export SOAP_USE_PROXY=on
$ export HTTP_PROXY=$http_proxy
```

    - On-line Demo auf einem der Linuxcluster-Rechner!