Vorlesung
Component Ware und Web-Services

- Komponentenkonzepte -

14. Web-Services II

Prof. Dr. Hans-Gert Gräbe, F. Schumacher
Wintersemester 2003/2004

Web-Services – Agenda

1. Motivatoren und Ziele
2. Grundmodell und Definition
3. SOAP – Nachrichtenaustausch
4. **WSDL – Beschreibung eines Webservices**
5. **UDDI – Registry für Webservices**
6. **Webservices und Sicherheit**
7. **Zukunft und Trends**
8. **Frameworks für Webservices**

4. WSDL – Beschreibung eines Webservice

Definition: WSDL

- **W**eb-**S**ervice **D**escription **L**anguage
- eine Art IDL in XML für Web-Services
- Standard des W3C
- entwickelt in 2001 von Ariba, IBM, Microsoft
- beschreibt Web-Services als eine Menge von Endpoints, die mit **Messages** arbeiten, welche dokumenten-orientiert oder prozedur-orientiert sind
- Operationen und Messages werden **erst** abstrakt beschrieben und **dann** an das jeweilige Netzwerk-Protokoll und Message-Format gebunden (SOAP, HTTP, MIME)

Beschreibung eines Webservice

- WSDL ermöglicht die automatisierte Erstellung von Clients für Webservices
- Die Operationen von Webservice (SOAP-RPC) setzen voraus, dass korrekte Funktionsaufrufe oder fachlich korrekte Dokumente ausgetauscht werden.
- WSDL stellt die Schnittstellenbeschreibung für diese Operationen oder Dokumente zur Verfügung
- WSDL beschreibt die Implementierung und die von Webservice-Konsumenten zu nutzenden Protokolle (Verpackung, Transport) für den Informationsaustausch
- WSDL dient dem Webservice dazu, standardisiert bekannt zu geben, welche Funktionen er besitzt, wie die Schnittstellen (Parameter) aussehen und über welche Protokolle Konsumenten ihn ansprechen können.

Beschreibung eines Webservice (2)

Vorteile von WSDL:

- Entwicklung und Pflege von Diensten wird über das standardisierte Schnittstellenformat WSDL einfacher
- Die Nutzung von Webservices kann automatisierter geschehen und Fehlerquellen bei der Clienterstellung für den Webservice-Zugriff werden minimiert
- WSDL erlaubt die dynamische Entdeckung und Anpassung der Clients an implemetierte Änderungen der Webservice-Funktionen
- WSDL lässt sich über Entwicklertools (z.B. MS .NET) automatisch aus den Applikationskomponenten erstellen

Beschreibung eines Webservice (3)

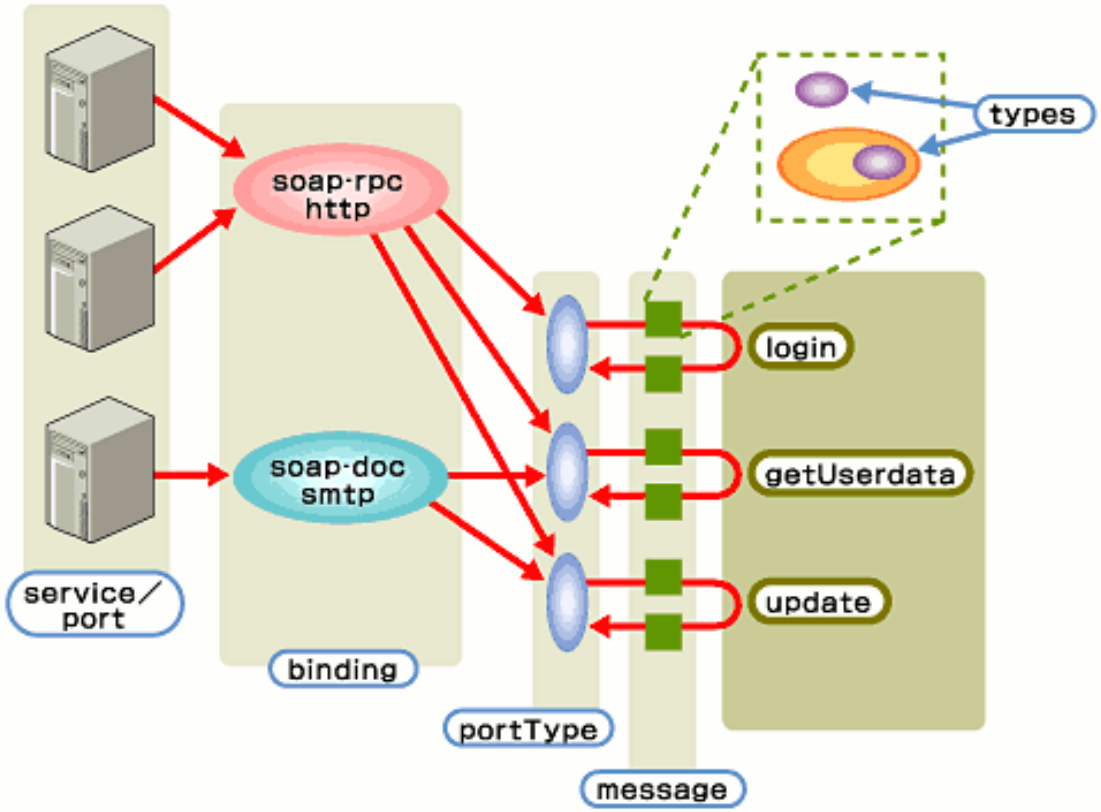
Notwendigkeit und Aufgabenbereich von WSDL:

- WSDL ist nicht zwingend für die Kommunikation mit Webservices erforderlich.
- Wenn der Konsument weiß, wie mittels SOAP mit dem Webservice kommuniziert, ist WSDL unnötig.
- Bei dynamischen Webservice-Aufrufen, wo noch nicht bekannt ist, wie mit dem unbekannten Dienst kommuniziert werden soll, stellt WSDL die Informationen für diese Kommunikation bereit

WSDL-Anatomie

- Root-Element: **<definitions>...</definitions>**
- Child-Elemente:
  - **<types>**
    Liste der verwendete Datentypen (meist XSD) der Messages
  - **<message>**
    abstrakte Darstellung der Daten bzw. Messages; in *parts*, die aus *input*, *output* oder *fault* bestehen
  - **<portType>**
    Collection von definierten Messages bzw. Aufruf und Antwortsignatur einer Webservice-Methode
  - **<binding>**
    Definition, wie die Methode des Webservice implementiert ist bzw. konkret an ein Protokoll (z.B. SOAP, HTTP) gebunden ist
  - **<port>**
    Endpoint, bestehend aus einem binding und einer konkreten Netzwerk-Adresse bzw. einer URL
  - **<service>**
    Collection von mehreren Ports, wo überall der Webservice zu finden ist

WSDL-Anatomie (2)



Quelle: http://www.atmarkit.co.jp/fxml/rensai/soap04/soap4_4.gif

- **Messages** bestehen aus **Types**
- Funktionen (PortTypes) kommunizieren über Messages
- **Funktionen** (PortTypes) sind an Protokolle (RPC, EDI) gebunden
- Die **Services** liegen auf min einem Server (services/ports), der über ein bestimmtes Protokoll anzusprechen ist.

WSDL-Anatomie (3)

```
<wsdl:definitions name=„..“ targetNamespace=„..“>
 <wsdl:types><schema>..verwendete Daten-Elemente..</schema></wsdl:types>
 ...
 <wsdl:message><part>...Die Parameter...</part>...</wsdl:message>
 ...
 <wsdl:portType> <!– definiert die Kommunikationsnachrichten mit dem Webservice -->
  <wsdl:operation>
   <wsdl:input message=„..“/>     <!-- Hier stehen Aufrufe -->
   <wsdl:output message=„..“/>   <!-- Hier stehen Rückgaben -->
  </wsdl:operation>
 </wsdl:portType>
 ...
 <wsdl:binding>... Operationen & Messages werden mit Protokoll verknüpft (z.B. SOAP)
 </wsdl:binding>
 ...
 <wsdl:service>
  <wsdl:port name=„...“ binding=„...“>... Hier ist der Pfad zur konkreten Applikation... </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

WSDL-Beispiel

- Ein VB-Programm als Web-Service:

```
Public Function GetTemperature(zipcode As String,
celsius As Boolean) As Single
    If celsius Then
        GetTemperature = 21.7
    Else
        GetTemperature = 71.06
    End If
End Function
```

- Aufgabe der **WSDL**-Datei:
  - Operation „GetTemperature“ nach Außen darstellen
  - Inputmessage (zipcode, celsius) und Outputmessage (Rückgabewert von GetTemperature) beschreiben
  - Zugangsprotokoll (hier: SOAP) mit Operation „GetTemperature“ koppeln (sog. Binding)

WSDL-Beispiel (2)

- Definition des Web-Service in WSDL und WSDL-Präambel:

```
<wsdl:definitions
 name =„WetterDienst“
 targetNamespace=„urn:Wetterdienst“
 xmlns:tns=„urn:Wetterdienst“
 xmlns:soap=„http://schemas.xmlsoap.org/wsdl/soap“
 xmlns:wsdl=„http://schemas.xmlsoap.org/wsdl“>
 <wsdl:service name=„WetterDienst“>
 </wsdl:service>
</wsdl:definitions>
```

- Von wo ist der Web-Service verfügbar, d.h welcher **Port** führt ihn aus und auf welchem Server ist er ansprechbar:

```
<wsdl:service name=„WetterDienst“>
 <wsdl:port name=“WetterSoapPort”
  binding=“tns:WetterSoapBinding” >
  <soap:address location=“http://www.wetter.de/wetter.asp”/>
 </wsdl:port>
</wsdl:service>
```

WSDL-Beispiel (3)

- Jetzt müssen die **Messages** des Web-Service definiert werden

```
<wsdl:message name=„WetterDienst.GetTemperature“>
 <part name=„zipcode“ type=„xsd:string“/>
 <part name=„celsius“ type=„xsd:boolean“/>
</wsdl:message>
<wsdl:message name=„WetterDienst.GetTemperatureResponse“>
 <part name=„Result“ type=„xsd:float“/>
</wsdl:message>
```

- Diese **Messages** müssen natürlich noch den verfügbaren **Operationen** des Webservice „Wetterdienst“ zugeordnet werden:

```
<wsdl:operation name=„GetTemperature“ parameterOrder=„zipcode
celsius“>
 <wsdl:input message=„wsdlns:WetterDienst.GetTemperature“ />
 <wsdl:output message=„wsdlns:WetterDienst.GetTemperatureResponse“ />
</wsdl:operation>
```

WSDL-Beispiel (4)

- Die Operationen für einen Web-Service befinden sich in einem sog. **PortType**, also:

```
<wsdl:portType name='WeatherSoapPortType'>
 <operation name='GetTemperature' parameterOrder='zipcode celsius'>
  <input message='wsdlns:Weather.GetTemperature' />
  <output message='wsdlns:Weather.GetTemperatureResponse' />
 </operation>
 <!- hier könnten andere Operationen stehen -->
</wsdl:portType>
```

- Zum Schluss müssen die abstrakt definierten Operationen und Messages an ein konkretes Protokoll, mit dem die Operationen aufgerufen werden, gebunden werden **>>>**

WSDL-Beispiel (5)

Das Binding bindet die abstrakte Methodenbeschreibung an das protokollabhängige Vorgehen (SOAP, HTTP, MIME):

```
<wsdl:binding name='WetterSoapBinding' type='wsdlns:WetterSoapPortType' >
 <soap:binding style='rpc' transport='http://schemas.xmlsoap.org/soap/http' />
 <operation name='GetTemperature'>
  <soap:operation soapAction='http://tempuri.org/action/Weather.GetTemperature' />
  <input>
   <soap:body use='encoded' namespace='http://tempuri.org/message/'
    encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
  </input>
  <output>
   <soap:body use='encoded' namespace='http://tempuri.org/message/'
    encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
  </output>
 </operation>
</wsdl:binding>
```

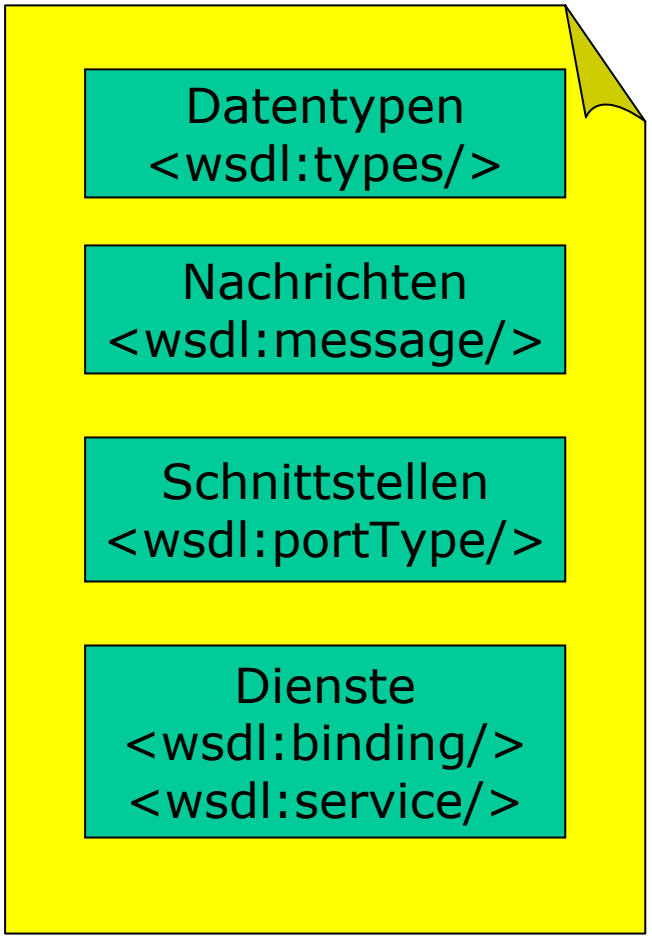
WSDL-Beispiel (6)

- Erstellung von URIs für Namespaces (xmlns:...):
  - für nicht global gültige URI kann man den URI: „*http://tempuri.org*“ benutzen
- Namespaces, die normalerweise im <definitions>-Tag eingebunden werden
  - *targetNamespace*: gibt Pfad des WSDL-Dokumentes an
  - *xmlns:tns*: wie ein this-Zeiger; auch Pfad des WSDL-Dokumentes
  - Andere Namespaces:
    - xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    - xmlns="http://schemas.xmlsoap.org/wsdl/">
- Namespaces dienen hier der Übersicht, um bestimmte XML-Elemente eindeutigen Zusammenhängen zuzuordnen und Elemente gleichen Namens (z.B. Order), aber unterschiedlichen Aufbaus zu trennen.

WSDL-Anatomie

Eine Service-Beschreibung beschreibt vier grundlegende Dinge an einem Webservice:

- die verwendeten Datentypen
- die Nachrichten
  - logische Sammlung benannter **Parts** eines bestimmten **Typs**
- die Schnittstellen
- die Dienste
  - Sammlung von **Ports**, die aus einer abstrakten Definition (dem Port-Typ) und einer konkreten Definition (der Bindung) bestehen, in der die verwendeten Protokolle spezifiziert sind



Datentypstrukturen in WSDL

- Datentypen werden in WSDL unterhalb des Tags **<wsdl:types>** definiert oder referenziert
- Datentypen dienen der **Interoperabilität** von Anwendungen und Diensten, um eine gemeinsame Austauschbasis für Daten zu gewährleisten
- In WSDL werden für Dienstkonsument und Dienstanbieter über den Mechanismus der XML-Schemas übergreifende Datentypenstrukturen geschaffen
- Das Konzept der XML-Schemas ist **plattformneutral** und sprachunabhängig
- Die **Flexibilität** der XML-Schemas erlaubt es in WSDL die Besonderheiten bestehender Programmiersprachen oder Datenaustauschstandards abzubilden

Datentypstrukturen in WSDL (2)

- WSDL erlaubt zwei Einbettungsarten für das Datentypsystems der XML-Schemen:
  - Import über Referenzierung zu einer XSD-Datei (leider noch nicht von vielen Tools unterstützt)
  - Explizite Einbindung des Schemas in die WSDL-Datei
- Beispiel für Import eines XML-Schemas:

```
<wsdl:definitions name=„MyServiceDefinition“
 targetNamespace=„urn:MyService“
 xmlns:tns=„urn:MyService“
 xmlns:types=„urn:MyServiceDataTypes“
 xmlns:soap=„http://schemas.xmlsoap.org/wsdl/soap“
 xmlns:wsdl=„http://schemas.xmlsoap.org/wsdl“
>

 <wsdl:import namespace=„urn:MyServiceDataTypes“
              location=„data_types.xsd“ />

</wsdl:defintions>
```

Hier wird aus der Datei **`data_types.xsd`**

die Definition via **`import`** referenziert

und an den Namesspace **`types`** gebunden

Datentypstrukturen in WSDL (3)

Beispiel für die Einbettung eines XML-Schemas:

```
<wsdl:definitions name=„MyServiceDefinition“
 xmlns:types=„urn:MyTypes“>

<wsdl:types>
 <xsd:schema
   xmlns:xsd=„http://www.w3.org/2000/10/XMLSchema“
   targetNamespace=„urn:MyTypes“
   elementFormDefault=„qualified“>
  <xsd:simpleType name=„productCode“>
   <xsd:restriction base=„xsd:string“>
    <xsd:pattern value=„\d{2}-\d{5}“/>
   </xsd:restriction>
  </xsd:simpleType>
 </xsd:schema>
</wsdl:types>

</wsdl:defintions>
```

Hier wird das Element `productCode` als `String` definiert, der nach diesem Muster aufgebaut ist:

„12-13452“

Webservice-Schnittstelle

- Webservice-Schnittstellen sind wie Schnittstellen (Interfaces) in objektorientierten Sprachen
- Schnittstellen enthalten Informationen über:
  - Eingabenachrichten
  - Ausgabenachrichten
  - Fehlernachrichten
- In WSDL heißt diese Schnittstelle zu einem Dienst **portType:**

```
<wsdl:portType name='WeatherSoapPortType'>
 <operation name='GetTemperature' parameterOrder='zipcode celsius'>
  <input message='wsdlns:Weather.GetTemperature' />
  <output message='wsdlns:Weather.GetTemperatureResponse' />
 </operation>
 <!- hier könnten andere Operationen stehen -->
</wsdl:portType>
```

Webservice-Implementierung

- WSDL beschreibt auch die konkrete Implementierung der Webservice-Schnittstelle
- WSDL nutzt dazu **Bindung** und **Ortsbeschreibung**

```
<wsdl:binding name='WetterSoapBinding'  type=tns:WetterSoapPortType' >
 <soap:binding style='rpc' transport='http://schemas.xmlsoap.org/soap/http' />

 <wsdl:operation name='GetTemperature'>
  <soap:operation soapAction='http://tempuri.org/action/Weather.GetTemperature' />

  <wsdl:input>
   <soap:body use='encoded' namespace=„...“ encodingStyle=„...“ />
  </wsdl:input>

  <wsdl:output>
   <soap:body use='encoded' namespace=„...“ encodingStyle=„...“ />
  </output>
 </wsdl:operation>
</wsdl:binding>
```

Webservice-Implementierung (2)

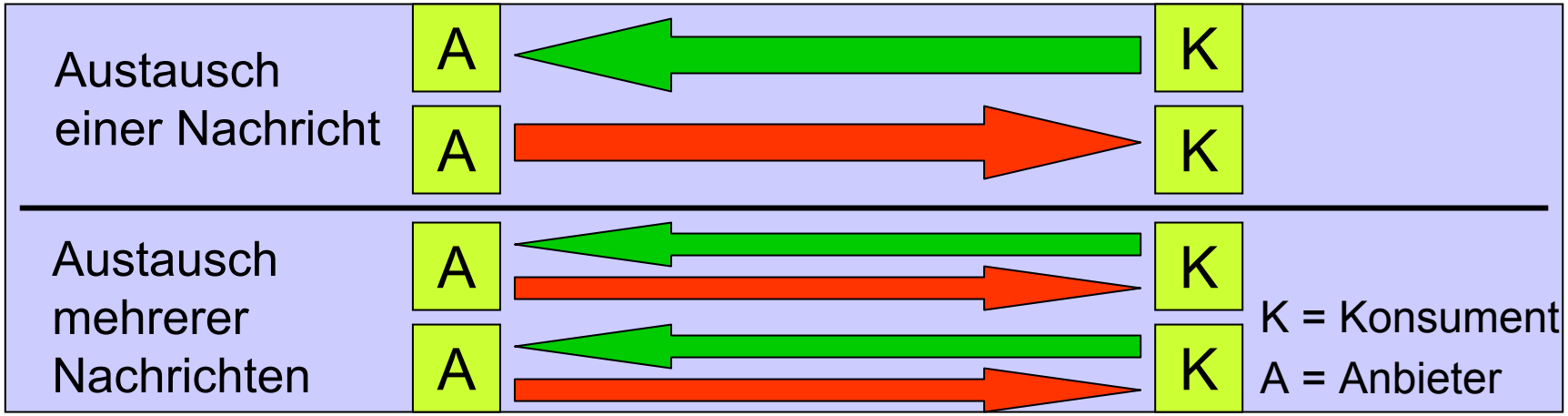
- Zusätzlich zum Binding beschreibt WSDL, an welchem Ort sich der Webservice befindet.
- Das Element **<wsdl:service>** enthält diese Information zur Lokalisierung des Webservices

```
<wsdl:service name=„WetterDienst“>

 <wsdl:port name=“WetterSoapPort” binding=“tns:WetterSoapBinding” >

  <soap:address location=“http://www.wetter.de/wetter.asp”/>

 </wsdl:port>

</wsdl:service>
```

Hier wird angeben,
auf welchem Server
bzw. auf welcher URL
der Service „Wetterdienst“ liegt.

Nachrichtenmuster

- Nachrichten können auf zwei Arten ausgetauscht werden:
  - Austausch einzelner Nachrichten = Dokumentenaustausch
  - Austausch mehrerer Nachrichten = RPC
- Je nach Austauschvariante wird der **PortType** modifiziert:
  - Bei einzelnen Nachrichten wird jeder Operation nur ein Nachrichtentyp (Input **oder** Output) zugeordnet
  - Bei mehreren Nachrichten pro Dienst werden jeder Operation eine Input- **und** eine Outputnachricht zugeordnet



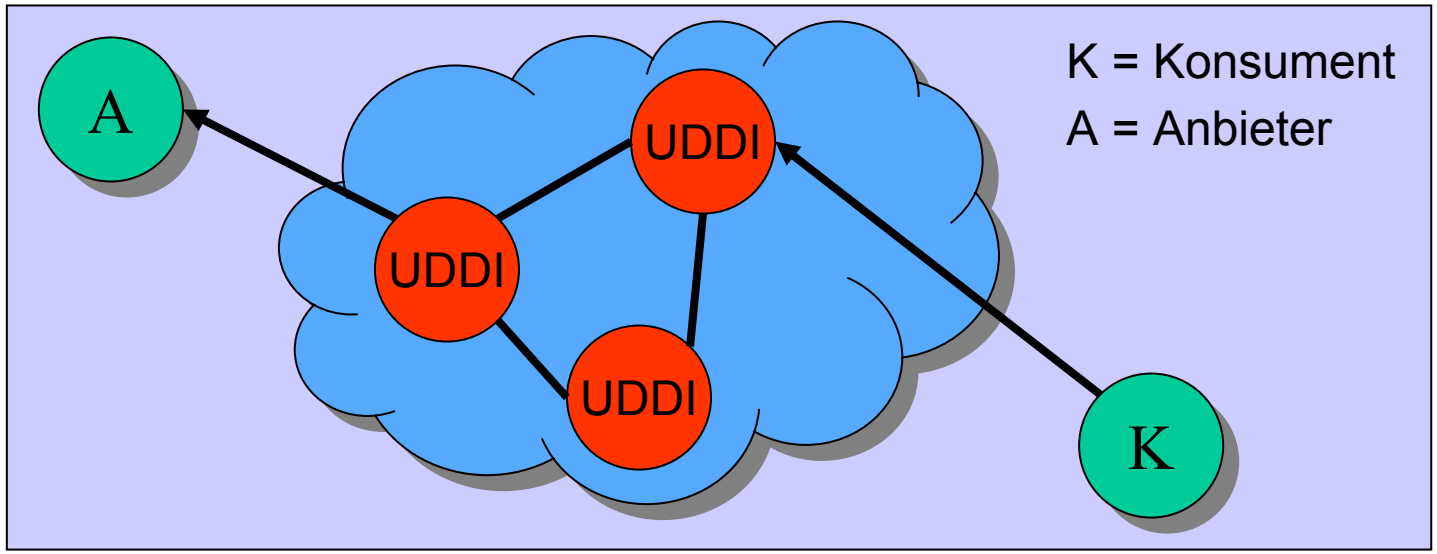
5. UDDI – Registry für Webservices

Definition: UDDI

- **U**niversal **D**escription, **D**iscovery and **I**ntegration
- „Gelbe Seiten“ für Web-Services, damit ein Dienstkonsument die Webservices auch findet, um diese zu benutzen.
- dient zur Lokalisierung und Veröffentlichung von Web-Services im Internet
- UDDI = Register für Dienste und ihre Beschreibungen + Suchmethoden + Publishingmethoden
- UDDI-Daten enthalten Kontakt-Informationen, Listen von Business Services und Infos, wie ein Service via Protokoll angesprochen werden kann
- Seit Sept. 2000 von Microsoft, Ariba, IBM und 33 anderen entwickelt, mehr als 300 Community-Members
- ***„Making it possible for organizations to quickly discover the right business from millions currently online“ (uddi.org)***

Definition: UDDI (2)

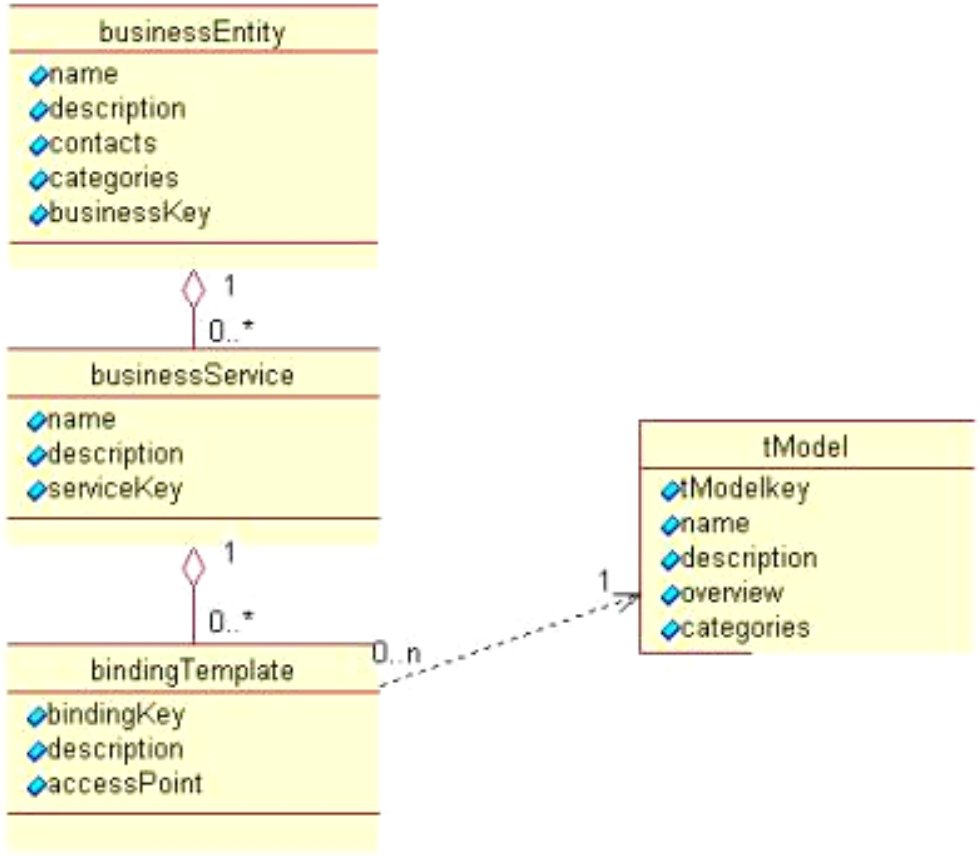
- UDDI besteht aus einem **Netzwerk** miteinander verbundener Registries
- Alle UDDI-Registries haben mit SOAP alle dieselbe Webservice-Schnittstelle für **Veröffentlichen** und **Finden** für Webservices implementiert.
- Es existieren auch **private** UDDIS (Firmen-UDDIs), um private Services und Methoden über das Internet verfügbar zu machen und in die verteilte Unternehmenslandschaft zu integrieren.



Elemente der UDDI-Registry

- Die UDDI-Registry besteht aus einer XML-Struktur
- In dieser Struktur wird eine Hierarchie von Beschreibungen dargestellt über:
  - *das Unternehmen (Business-Entity)*
  - *die Services (Business-Service)*
  - *die Bindungen (Binding-Template)*
- Zur Modellierung von abstrakten Begriffen oder Strukturen für Business, Service und Template wird das Element **TModel** verwendet
- TModel ist das Struktur-Bausteinsystem für UDDI

Elemente der UDDI-Registry (2)



Quelle: www.vbws.com/tutors/ uddi/uddi.aspx

Jeder Service gehört zu einer Firma (*BusinessEntity*) und hat einen Typ (*BindingTemplate*)

Alle Elemente haben einen eindeutigen Identifier (*UUID*)

Die *TModels* sind bekannte, gebräuchliche Typen in UDDI, wie z.B. SOAP

Elemente der UDDI-Registry (3)

- Das Element **Business-Entity** beschreibt den Anbieter eines Webservices
- Es enthält Daten über Firma, Branche, Kontaktinformationen, Kategorien und die Liste der angebotenen Dienste.

```
<businessEntity businessKey=„uuid:F1I2R3M4A-5K6E7Y ...>
 <name>Fiktive Firma</name>
 <description>Wir bauen WebServices</description>
 <contacts>
  <contact useType=„general info“>
   <description>Standardauskunfts</description>
   <personName>Hans Mueller</personName>
   <phone>0341 123456</phone>
  </contact>
 </contacts>

 <businessServices>
 ...Hier stehen die angebotenen Webservices...
 </businessServices>
 <categoryBag/>
</businessEntity>
```

Hier können auch Kategorien (**CategoryBags)** enthalten sein.

Kategorie-Möglichkeiten:
- NAICS
- D-U-N-S

NAICS= North American Industry Classification System
DUNS = Dun-and-Bradstreet-No

Elemente der UDDI-Registry (4)

- Das Element **Business-Service** beschreibt einen einzelnen Webservice, der von einer **Business-Entity** angeboten wird
- Es enthält Informationen über die Bindungs des Webservices (SOAP, HTTP, SMTP,...), Art des Webservice und Kategorie.

```
<businessService serviceKey="uuid:S1E2R3V-I4C5E6-K5E8Y"
                 businessKey="uuid:F1I2R3M4A-5K6E7Y">
 <name>Ein Webservice</name>
 <description>Fliesstext</description>

 <bindingTemplates>
 ... Dazu kommen wir gleich ...
 </bindingTemplates>
 <categoryBag/>

</businessService>
```

Bemerkung:
Jedes Element ist über eine eindeutige **UUID** bestimmt.

UDDI vergibt die UUIDs automatisch, bei erstmaligem Eintrag in die Registry.

UUID = Universally Unique Identifiers

Elemente der UDDI-Registry (5)

- Das Element **Binding-Template** enthält die technische Beschreibung des Webservices, der in einer Business-Service-Struktur enthalten ist
- Wie in WSDL kann jeder Webservice mehrere Bindungen besitzen, abhängig davon, ob er über HTTP oder SMTP angesprochen wird.
- Entspricht in WSDL dem Element **<wsdl:service>**

```
<bindingTemplate serviceKey=„uuid:S1E2R3V-I4C5E6-K5E8Y“
                 bindingKey=„uuid:B1I2N3D4I-N6G7-8K9E0Y“>
 <description>SOAP-Bindung für Webservice</description>

 <accessPoint URLType=„http“>
  http://www.woistderservice.de  <!- die URL zum Webservice -->
 </accessPoint>

 <TModelInstanceDetails>
  ... Hier steht ein Verweis zu einer TModel-Struktur ...
 </TModelInstanceDetails>

</businessService>
```

UDDI – Schnittstellen

- Der Zugriff auf UDDI, um Dienste zu veröffentlichen oder zu suchen erfolgt über **zwei** SOAP-Schnittstellen
- Dienstkonsumenten nutzen **InquireSOAP**, um einen Webservice zu finden
- Dienstanbieter nutzen **PublishSOAP**, um einen Service bekannt zu machen
- Die Dienstschnittstellen liegen als WSDL-Dokumente vor
- Die Datentypen von UDDI sind als XML-Schema unter **http://www.uddi.org/schema/2001/uddi_v1.xsd** verfügbar

InquireSOAP (9 Methoden):
- find_business
- find_*XYZ*
- get_businessDetail
- get_businessDetailExt
- get_*XYZ*Detail

XYZ = {binding, service, TModel}

PublishSOAP (11 Methoden):
- get_authToken
- discard_authToken
- save_business
- save_*XYZ*
- delete_business
- delete_*XYZ*
- get_registeredInfo

UDDI – Schnittstellen (2)

Übersicht über die Kern-Methoden von SOAP-Publish von UDDI:

- **Get_AuthToken**
  - Besorgt einen Authentifizierungscode
  - Nur über den Authentifizierungscode können Änderungen an der UDDI-Registry vorgenommen werden
- **Discard_AuthToken**
  - Entspricht dem Logout aus UDDI
- **Save_Business**
  - Erzeugt / Aktualisiert die Information über eine Business-Entity
- **Save_Service**
  - Erzeugt / Aktualisiert die Information über einen Webservice
- **Save_Binding**
  - Erzeugt / Aktualisiert die techn. Informationen über den Webservice
- **Delete_*{XYZ}***
  - *{XYZ}* = business, service, binding, TModel,
  - Entfernt ein bestimmtes Objekt aus der UDDI-Registry

UDDI – Schnittstellen (3)

Übersicht über die Kern-Methoden von SOAP-Inquire von UDDI:

- **find_binding**
  Gibt eine Auflistung von Services mit bestimmten techn. Kriterien zurück
- **find_business**
  Gibt eine Auflistung von Business-Entities mit bestimmten Kriterien zurück
- **find_service**
  Gibt eine Liste von Services, die gewisse Kriterien erfüllen, zurück
- **get_businessDetail**
  Gibt Detailinformationen zu einer Business-Entity zurück
- **get_serviceDetail**
  Gibt Informationen über einen Service zurück
- **get_bindingDetail**
  Gibt Informationen über ein Bindungsschema zurück

Ablauf einer Registrierung mit UDDI4J

- Für den UDDI-Zugriff steht neben kommerzieller Software auch das auf Java basierende OpenSource-Tool **UDDI4J** von IBM zur Verfügung (http://oss.software.ibm.com/developerworks/projects/uddi4j)
- Vorgehensweise mit **UDDI4J**:
  1. Registrierung des Serviceanbieters als UDDI-Business-Entity
  2. Angabe beschreibender Kategorien zur Business-Entity
  3. Registrierung des Webservice als UDDI-Business-Service
  4. Angabe beschreibender Kategorien zum Business-Service
  5. Registrierung der Implementierungsdetails (SOAP, HTTP, ...) und des URLs des Webservices
- Wichtiges vor der Registrierung des Webservices:
  - Einrichtung eines eigenen Accounts bei UDDI (*http://www.uddi.org*)
  - Für UDDI4J muss das Apache SOAP-Toolkit installiert sein

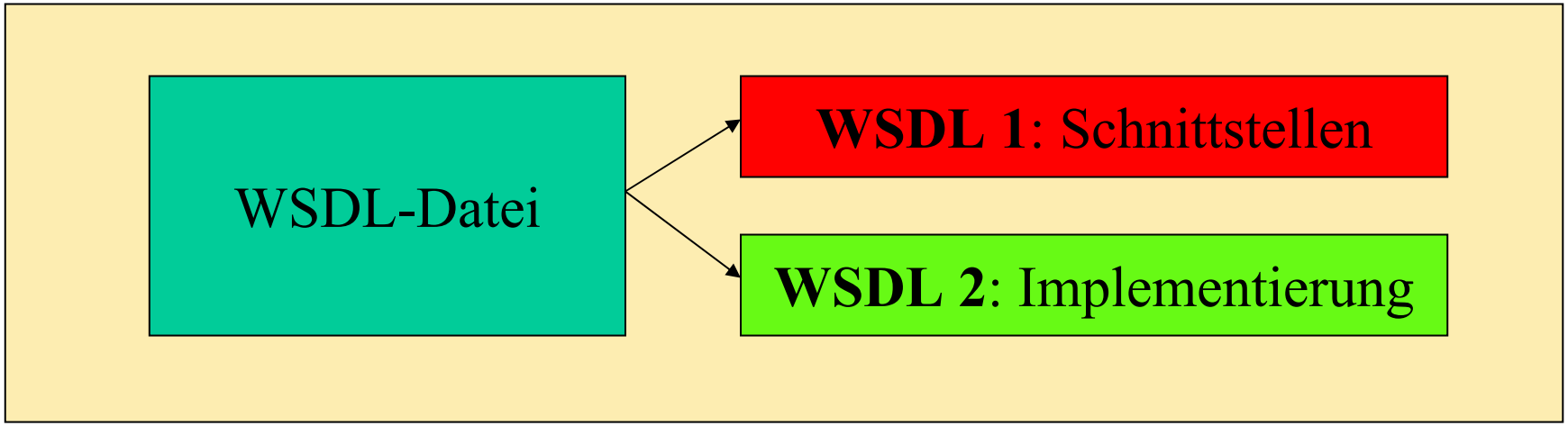
UDDI aus WSDL erzeugen

- WSDL hat mit seinem **<wsdl:service/>**-Tag, in dem Ort und Implementierung des Webservices definiert werden, einen Teil von UDDI dargestellt
- Das Konsortium hat für Nutzung von UDDI und WSDL und die Erzeugung von UDDI aus WSDL eine Anleitung entwickelt, um eine dynamische Entdeckung von Webservice-Beschreibungen zu gewähren
- UDDI4J unterstützt dieses Vorgehensmodell

UDDI aus WSDL erzeugen (2)

1) WSDL-Datei in 2 getrennte Dateien aufteilen

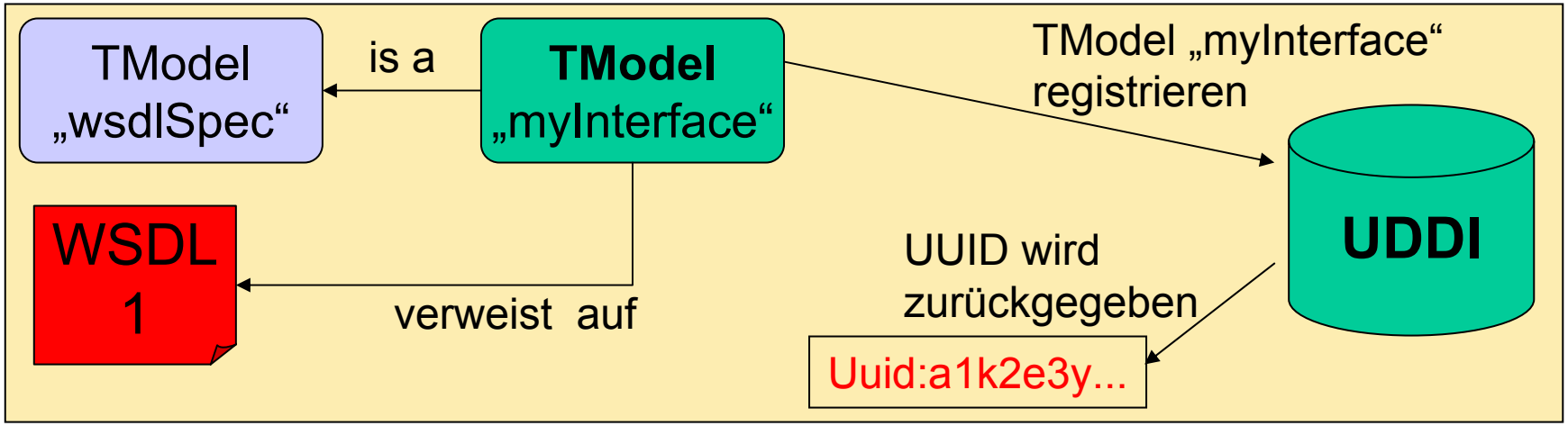
- WSDL-Datei 1:
  Schnittstellenbeschreibung mit Datentypen, Nachrichten, Port-Typen und Bindungen
- WSDL-Datei 2:
  Implementierungsbeschreibung mit der Dienstdefinition und einer Importanweisung für Datei 1



UDDI aus WSDL erzeugen (3)

2) WSDL-Datei 1 als TModel speichern

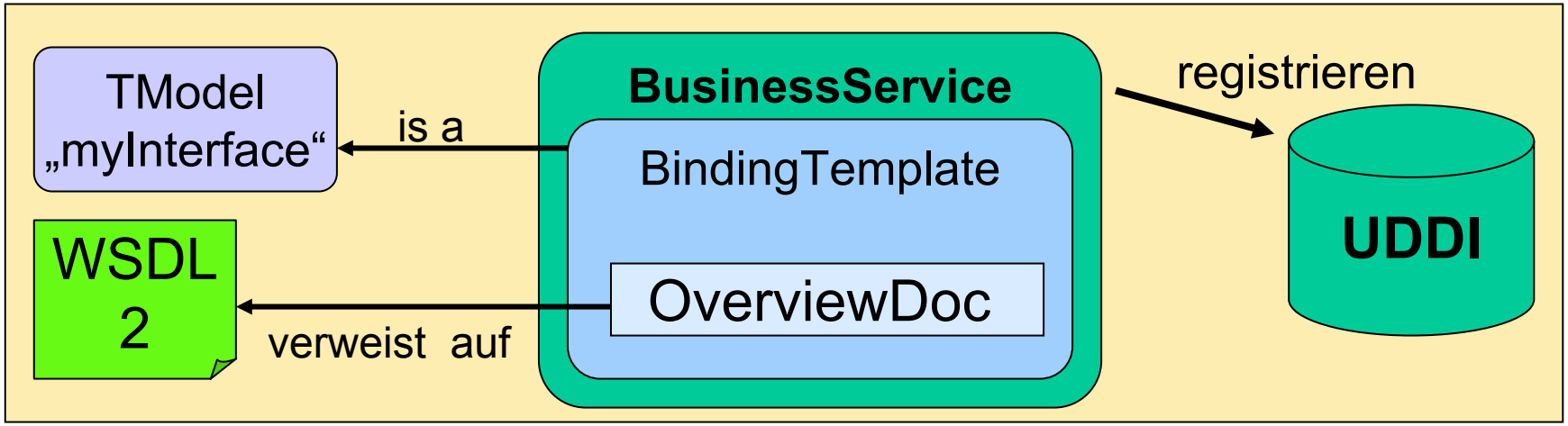
- Die Schnittstellenbeschreibung wird als neues TModel-Object (hier: myInterface) von „wsdlSpec“ abgeleitet und in UDDI registriert.
- Das neue TModel verweist dabei auf die Schnittstellenbeschreibung (WSDL-Datei 1).
- Beim Anlegen dieses TModel wird ein generierte UUID-Key von UDDI zurückgegeben.



UDDI aus WSDL erzeugen (4)

WSDL-2 als Bindung für den Service speichern

- Ein Business-Service-Objekt wird in UDDI4J angelegt.
- Im BusinessService wird das Unterobjekt „BindingTemplate“ erzeugt
- Das Attribut „OverviewDoc“ des BindingTemplate zeigt auf die URL der Implementierungsbeschreibung (WSDL-Datei 2).
- Das Attribut „TModelInstanceInfo“ des BindingTemplates ist ein TModel vom Typ „myInterface“.



Einsatz von WSDL und UDDI

- WSDL-definierte Webservices können in UDDI gesucht werden
- Moderne Toolkits (z.B. WebServices Toolkit von IBM) unterstützen die Suche in UDDI und den Aufruf der gefundenen Services über ein durch die WSDL-Datei dynamisch konfigurierten Proxy
- Typischer Ablauf:
  - Finden des Webservices (über SOAPInquire-Methoden) in UDDI
  - Zugriff auf die referenzierte WSDL-Beschreibung des Webservice
  - Aufruf des Webservice über den dynamischen Proxy
- Vorteile:
  - Implementierung des entfernten Webservice wird gekapselt
  - Aufruf und Zugriff auf UDDI, WSDL ist plattformunabhängig
  - Dynamischer Proxy kapselt SOAP-Aufrufe und technischen Details

Alternative: WS-Inspection

- Als einfache und nützliche Alternative wurde die Web Service Inspection Language (WS-Inspection) von IBM und Microsoft entwickelt
- WS-Inspection dient zur Erzeugung eines einfachen Registers von Servicebeschreibungen im Netz.
- WS-Inspection-Dokumente sollten an eine leicht auffindbare Stelle auf dem Firmenwebserver gestellt werden.
- Die Spezifikation legt fest, dass ein Inspektionsdokument (Inspection.wsil) im Rootverzeichnis des Servers stehen sollte.
- WS-Inspection ist nützlich, wenn der Dienstanbieter dem suchenden Dienstkonsumenten bekannt ist und das WSIL-Dokument dient dann nur zur genauen Lokalisation des Webservice

Alternative: WS-Inspection (2)

Elemente von WS-Inspection:

```
<inspection xmlns=„...“>
     ... mehrere <abstract/>,<service/> und <link/> möglich ...
  <service>
      <abstract>
        ... Hier kann der Titel des Service stehen ...
        ... Einfache Dokumentationsmöglichkeit
      </abstract>

      <description>
        ... Hier kann der ServiceKey von UDDI stehen ...
        ... Hier können Bezüge zu WSDL- und UDDI-Dokumenten stehen ...
      </description>

      <link referencedNamespace=„...“ location=„...“>
  </service>
</inspection>
```

6. Sicherheit von Webservices

Aspekte der Sicherheit

- Sicherheit ist ein zentraler Aspekt für **Akzeptanz** von Webservices.
- Standards und verlässliche Sicherheitskonzepte sind notwendig.
- Kern der Sicherheit sind **Authentifizierung** und **Vertraulichkeit**.
- **Ziele** der Sicherheit von Webservices:
  - **Authentifizierung:** Nur berechtigte Dienstkonsumenten haben Zugriff auf die für sie bestimmten Webservices.
  - **Vertraulichkeit:** Die Datenströme zwischen Dienstkonsument und Dienstanbieter sind vor Zugriff unberechtigter Dritter geschützt.
- Die drei bekanntesten (konkurrierenden) **Architekturen** sind:
  - **Microsoft Passport (aktuell Version 3.X)**
  - **AOL Magic Carpet**
  - **Network Identity (Opensource, von Sun initiiert)**

Aspekte der Sicherheit (2)

- Was ist ein **sicherer** Webservice:
  - Absender kann vertrauen, dass der tatsächlich Empfangende auch wirklich der bestimmte Empfänger ist.
  - Nur der bestimmte Empfänger kann die ausgetauschten Informationen empfangen und verstehen.
- Resultierende Aufgaben für die Sicherheit:
  - Mechanismen für die **Authentifizierung**
  - Verschlüsselung für **Vertraulichkeit** und Integrität

Authentifizierung

- **Aspekte** der Authentifizierung:
  - Wer bin ich und weise ich nach, wer ich bin?
  - Wie kann ich nachweisen, dass ich wirklich *ich* bin?
  - Wie kann ich meinem Gegenüber trauen?
- **Lösungsansätze**:
  - SAML (Security Assertions Markup Language)
    - Fähigkeiten zum Single-Sign-On
    - Standardisiert und maschinenlesbar
    - Austausch digital signierte Bestätigungen einer vertrauenswürdigen Stelle, welche die Identität garantiert.
  - WS-Security / WS-Licence (Microsoft, früher „Hailstorm“)

Vertraulichkeit

- **Aspekte** der Vertraulichkeit:
  - Schutz von Gütern und Dienstleistungen
  - Schutz persönlicher Daten (z.B. Kreditkartennummer)
  - Schutz vor unerlaubtem Zugriff (Autorisierung, Verschlüsselung)
- **Lösungsansätze**:
  - W3C Platform for Privacy Preferences (P3P):
    - XML basierte Sprache für Vertraulichkeitsprofile
    - **Aufgabe:** Dienstanbieter bezeugen in diesen Profilen wie sie mit den Daten der Dienstkonsumenten umgehen.
    - **Problem**: Vertraulichkeitsprofile sind nicht rechtlich bindend.
  - SSL:
    - **Aufgabe**: Verschlüsselung der Daten während des Transports
    - **Problem**: Hindert Dienstanbieter nicht an Datenmissbrauch.

Microsoft Passport Version 2.5

- **Allgemeines:**
  - Bietet Dienste für Single Sign-On und digitale Wallets an
  - Teil der .NET Strategie
- **Methode:**
  - Benutzer besucht eine Passport-fähige Internetseite.
  - Dort ist ein „Passport Sign-On“- Link, der den Browser auf einen Loginbereich lenkt, wo er sich mit Email und Kennwort einloggen kann.
  - Nach dem Login wird auf seinem PC ein Userprofil-Cookie abgelegt.
  - Passport lenkt Benutzer zurück zur Originalseite, welche die Daten aus dem Cookie liest und den Benutzer als eingeloggt erkennt.
- **Probleme:**
  - Passport-Fakes könnten Benutzer täuschen und ihre Daten erspähen
  - Ein Passportdienst könnte ungehindert Daten verteilen.

AOL Magic Carpet

- **Allgemeines:**
  - Single Sign-On für diverse Webseiten im Internet
  - Kompatibilität zu *AOL Instant Messenger*, *AOL* und *CompuServe* Accounts
  - Erweiterung von *AOL Screen Name*
- **Methode:**
  - Der User hat die Möglichkeit für Websites, die er besucht, Profile anzulegen.
  - Die User kann die Profile so modifizieren, dass die besuchten Websites nur auf bestimmte Inhalte des Userprofils zugreifen können.

Network Identity

- **Allgemeines:**
  - Entwickelt von *Sun* und Konsortium *Liberty Alliance*
  - Teil von *Sun ONE*
- **Ziele:**
  - Dienstkonsumenten oder Geschäften wird ermöglicht, persönliche Daten sicher dezentral zu verwalten
  - Es wird ein offener Standard für „Single Sign-On“ angeboten.
  - Standard soll für alle Geräte und Infrastrukturen dienen, die mit dem Internet verbunden sein können, um eine standardisierte Identifizierung zu gewährleisten.

7. Zukunft und Trends

Zukunftsthesen für Webservices

- Aspekte der Programmiersprachen, Betriebssysteme, Datenbanksysteme oder Objektmodelle werden durch Webservices in den Hintergrund treten.
- Probleme der **Interoperabilität** werden verschwinden, da bei Webservices die Servicesuche (UDDI), die Serviceschnittstellen (WSDL) und die Art des Datentransfers zwischen Anbieter und Konsument (SOAP) unabhängig von konkreten Programmiermodellen sind.
- Sprachen und Systeme, die nie wirklich miteinander kommuniziert haben (Java und COM, COBOL und Perl) werden kommunizieren können, dank offener Standards.
- Webservices werden bestehende technische Infrastrukturen nicht ersetzen, d.h Großrechner bleiben, können aber besser durch Webservices integriert werden.

Zukunft von SOAP

- SOAP Version 1.1 und Version 1.2 sind noch keine offiziellen W3C-Standards werden aber wahrscheinlich bald in die Version 1.0 von ***XML Protocol*** einfließen, einem dann standardisierten Protokoll.
- *XML Protocol* soll abwärtskompatibel zu SOAP 1.2 sein.
- Experten erwarten keine großen Unterschiede zu SOAP.
- Unter *http://www.w3.org/2000/xp* gibt es die Mailingliste xml-dist-app, in der die aktuelle Diskussion zu *XML Protocol* verfolgt werden kann.
- W3C Arbeitsgruppe zu SOAP Version 1.2 soll Arbeit bis 31.01.2004 beendet haben

Zukunft von WSDL

- Auch WSDL ist noch kein definiter W3C-Standard
- Erweiterungen von WSDL werden besonders für eCommerce-Szenarien notwendig sein, die weitergehende Informationen der Webservice-Schnittstelle anbieten, z.B:
  - Sicherheitserfordernisse (Information über Authentifizierungsmethode)
  - Servicequalitätsattribute
  - Operationenfolge
- Arbeiten vom W3C an WSDL Version 2.0:
  - Core Language
  - Message Patterns
  - Bindings

Zukunft von UDDI

- UDDI soll mit Version 3.0 zur Standardisierung eingereicht werden.
- UDDI soll in Zukunft eine Sicherheitsinfrastruktur bereitstellen, die es Dienstkonsumenten erlaubt, die Identität eines Webservice-Providers zu verifizieren, um so mehr Vertrauen zu schaffen.
- UDDI muss das Problem „toter“ Links lösen, wo Verzeichniseinträge auf nicht mehr existierende Firmen oder Dienste verweisen.
- Die Aufgabe und der Nutzen von UDDI ist vielen Firmen noch unbekannt, so dass die Vorteile dieser öffentlichen Registry für Webservices besser kommuniziert werden müssen.

Trends und Szenarien

- Webservices können neue Dienstleistungen wie „elektronische Geldbörsen“ darstellen, um einfache und sichere Online-Transaktionen automatisiert durchzuführen.
- Webservices können als preiswerte Alternative zu EDI (Electronic Document Interchange), dem automatisiertem Austausch von Bestellungen, Rechnungen etc., eingesetzt werden.
- Webservices können in Unternehmen als Infrastrukturdienste für Trust Management, Buchung, Rechnungsstellung oder dynamische Warenbeschaffung eingesetzt werden.
- Softwareagenten können durch XML und Webservices entwickelt werden, die Flug-, Mietwagen- und Hotelbuchungen nach festgelegten Kriterien vornehmen. (aber: hohe Sicherheit + hohe Zuverlässigkeit)
- Zuverlässige und standardisierte Nachrichtenübermittlung im Unternehmen zwischen „unsicheren“ Endpunkten über SOAP (z.B. BizTalk) und HTTP-R.

8. Frameworks für Webservices

8.1 Microsoft .NET

8.3 Apache SOAP

Microsoft .NET

- Methoden zum Finden u. Aufrufen sind in das .NET Framework integriert.
- Diese Funktionen sind über den Namensraum `System.Web.Services` verfügbar
- Für einen Webservice muss kein spezieller Code geschrieben werden außer dem reinen Implementierungscode für die Funktion in C#, VB.NET oder ASP.NET.
- Der Funktion, die als Webservice agieren soll, wird das Attribut `[WebMethod]` hinzugefügt.
- Webservices werden in Visual Studio .NET von der Klasse `System.Web.Services.WebService` abgeleitet.
- Visual Studio .NET erzeugt automatisch Webseiten, welche den Webservice beschreiben (ASMX-Dateien) und mit denen auch direkt der Webservice getestet werden kann.
- Auch WSDL-Seiten werden automatisch generiert und können z.B. über `http://localhost/myServices/Hello.asmx?wsdl` abgerufen werden
- Bevor ein Client den Webservice nutzen kann, muß mit Hilfe des .NET Tools aus der WSDL-Datei eine Proxy-DLL erzeugt werden.

Microsoft .NET

- Beispiel für einen .NET Webservice:

```
Using System.Web.Services;

Namespace myServices
{
   public class Hello : System.Web.Services.WebService
   {
        [ WebMethod ]
        public string sayHello ()
        {
             return „Hello World!“;
        }
   }
}
```

Apache SOAP

- Apache SOAP ist eine Implementierung des SOAP-Protokolls durch die Apache Software Foundation
- Apache SOAP läuft hierbei als Servlet innerhalb eines belieben Java-HTTP-Servers (z.B. Tomcat).
- Diese SOAP-Implementierung bildet nur den Proxy-Teil der Nachrichtenverarbeitung bei Webservices ab.
- Nach dem Erstellen der Java-Klasse, die als Webservice agieren soll, muß diese im Klassenpfad des Webservers abgelegt werden
- Weiterhin muss diese Klasse im Deployment-Descriptor eingetragen werden und dem ServiceManager von Apache SOAP mitgeteilt werden.
- Um den Java-Webservice aufzurufen, ist ein Client zu erzeugen
- Der Code des Clients lässt sich u.a. durch das Web Service Toolkit von IBM reduzieren, welches mit auf WSDL-basierenden dynamischen Proxys arbeitet.

Apache SOAP (2)

- Beispiel für einen Java-Webservice:

```
public class Hello
{
   public String sayHello()
   {
     return „Hello!“;
   }
}
```

Apache SOAP (3)

- Beispiel für den Java-Client, der den Webservice aufruft:

```
import java.io.*; import java.net.*; import java.util.*;
import org.apache.soap.*; import org.apache.soap.rpc.*;
public class HelloClient{

      public static void main(String[] args) throws Exception{
         URL url = new URL(„http://localhost/soap/servlet/rpcrouter“);

         Call call = new Call();
         call.setTargetObjectURI(„urn:HelloWorld“);
         call.setMethodName(„sayHello“);
         call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);

         System.out.println(„Hallo PC“);
         Response resp = call.invoke(url, „“); // Aufruf des Webservice
         System.out.println(„PC: “+resp.getReturnValue().getValue());
      }
}
```

Quellen

- **Hauptquellen**
  - Snell, J. et al: Webservice-Programmierung mit SOAP, O'reilly, 2002
  - Liberty, J.: Programmieren mit C#, O'reilly, 2002

- **Ziele, Motivatoren**
  [1a] Why We Need Web Services Networks
  http://e-serv.ebizq.net/wbs/spicer_1.html

  [1c] The true nature of Web Services
  http://www.perfectxml.com/articles/xml/wsnature.asp

- **Allgemein, Überblick, Modell**
  [2a] Web Services and Distributed Objects: Competing or Complementary?
  http://e-serv.ebizq.net/wbs/conway_1.html

  [2b] Web Services: The Next Generation of Distributed Computing
  http://e-serv.ebizq.net/obj/hildreth_1.html

  [2c] Web Services Spotlight Report (Triple Tree – Investment Bank)
  http://e-serv.ebizq.net/shared/goldclub.jsp?tripletree_5b.html

  [2d] Web Services: Standardizing EAI
  http://www.eaijournal.com/PDF/WebSeryvicesKuzyk.pdf

  [2e] Web Services, Business Objects and Component Models
  http://www.orchestranetworks.com/us/pdf/tmxwp09.pdf

Quellen (2)

Bausteine, Architektur

IIIa SOAP

[3aa] Introduction to Web Service, SOAP & WSDL
http://www.aspalliance.com/yusuf/Article11.asp

[3ab] Introducing SOAP
http://www.vbxml.com/soapworkshop/articles/intro/default.asp

[3ac] SOAP: The Internet Is Not Just for Browsing Anymore
http://e-serv.ebizq.net/shared/goldclub.jsp?newcomer_1.pdf

[3ad] SOAP: RPC or Messaging?
http://www.learnxmlws.com/tutors/rpcmsg/rpcmsg.aspx

[3ae] A Gentle Introduction to SOAP
http://radio.weblogs.com/0101679/stories/2002/03/16/aGentleIntroductionToSoap.html

[3af] SOAP-W3C-Specification
http://www.w3.org/TR/SOAP/

[3ag] Busy Developers Guide to SOAP 1.1
http://www.w3.org/TR/SOAP/

Quellen (3)

Bausteine, Architektur

IIIb WSDL

[3ba] WSDL: W3C Note 15 March 2001
http://www.w3.org/TR/wsdl

[3bb] WSDL: an Introduction
http://imatch.lcs.mit.edu/docs/seminar_wsdl_files/frame.htm

[3bc] Introduction to WSDL
http://www.learnxmlws.com/tutors/wsdl/wsdl.aspx

IIIc UDDI

[3ca] UDDI Registries and Reuse
http://e-serv.ebizq.net/wbs/meyer_1.html

[3cb] Introduction to UDDI
http://www.learnxmlws.com/tutors/wsdl/wsdl.aspx

[3cc] UDDI.org Whitepapers
http://uddi.org/pubs/ProgrammersAPI-V2.00-Open-20010608.pdf

Quellen (4)

Trends, Technologien, Zukunft

[4b] ***Web Services and Application Frameworks***
http://www.webservicesarchitect.com/content/articles/samtani04.asp

[5a] ***Why Web Services Will Gradually Dominate e-Business Development***
http://e-serv.ebizq.net/wbs/thomas_1.html

[5b] ***Asynchronous Web Services and the Enterprise Service Bus***
http://www.webservices.org/index.php/article/articleview/352/1/1/

[5c] Webservice Architect
http://www.webservice-architect.com