

Dienstorientierte Integration und Web Services

Andreas Schmidt
WS 2006/07

- Dienstorientierte Integration
 - ▶ Dienstorientierte Architekturen

- Das Web Services-Dreigestirn
 - ▶ SOAP
 - ▶ WSDL
 - ▶ UDDI

- BPEL4WS

- Fazit

Integrationsebenen (1)

Präsentationsebene

Präsentationsfragmente
Portlets

Prozeßebene

Anwendungslogikebene

Dienste
Dienstschnittstelle, -semantik
Dienstfindung, -orchestrierung

Informationsebene

Informationsquellen
Datenmodell, Schema,
semantische Heterogenität

Technische Ebene

Netzwerkprotokolle, RPC
Darstellungssyntax

Integrationssebenen (2)

- Bislang: **Informationsperspektive**
 - ▶ Einzubindende Anbieter sind **Informationsquellen**
 - ▶ Interaktion mit den externen Quellen läuft über das **Anfrage-Ergebnis-Paradigma**
 - ▶ Es existiert eine zentrale Stelle, an der alle Informationen zusammenlaufen
 - ▶ Hauptprobleme
 - Wie überwinde ich die technische Heterogenität?
 - Wie überwinde ich die semantische Heterogenität?
 - Anfrageübersetzung
 - Schemaintegration

Integrationsebenen (2)

- Thema heute: **Dienstintegration**
 - ▶ Einzubindende Anbieter sind **Dienste**
 - Kapselung von Daten und Funktionalität hinter einer Schnittstelle, die die Funktion eines Vertrages hat
 - ▶ **Dienst-/Methodenaufrufe** vom Dienstnehmer an den Dienstgeber als Interaktionsparadigma
 - ▶ Autonomen Dienste sollen durch eine Infrastruktur die gegenseitige Nutzung ermöglicht werden
=> **Dienstorientierte Architekturen**
 - ▶ Hauptprobleme
 - Wie mache ich Dienste interoperabel?
 - Wie finde ich benötigte Dienste?
 - Wie beschreibe ich Dienste?
 - Wie kombiniere/aggregiere ich Dienste?

Denken in Diensten



- Übertragung der Dienstleistungsorientierung in der internen Unternehmensorganisation auf die IT-Infrastruktur

- Einzelne Komponenten erbringen Dienste

- Wichtig: Dienstschnittstelle als Vertrag
 - ▶ welche Dienste werden angeboten?
 - ▶ wie kann man sie in Anspruch nehmen?

- Zusätzlich: Idee der Dienstgüte

Denken in Diensten (2)

Konsequente Trennung zwischen

Analogie

■ Schnittstelle

- ▶ nach außen veröffentlicht
- ▶ dauerhaft
- ▶ Standardisierung von Diensten möglich

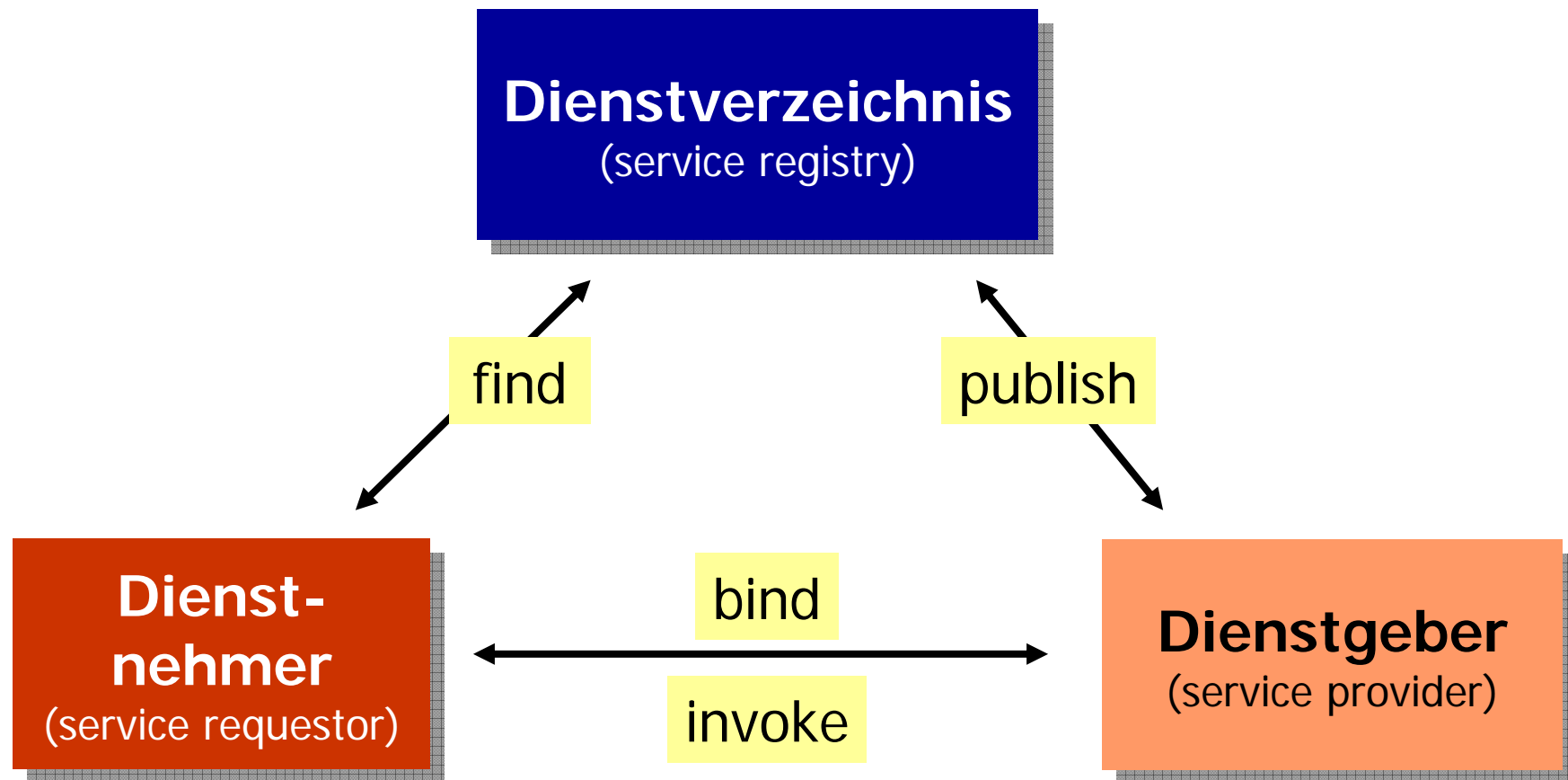
**Produktkatalog mit
Preisen und Bestell-/
Lieferbedingungen**

■ Implementierung

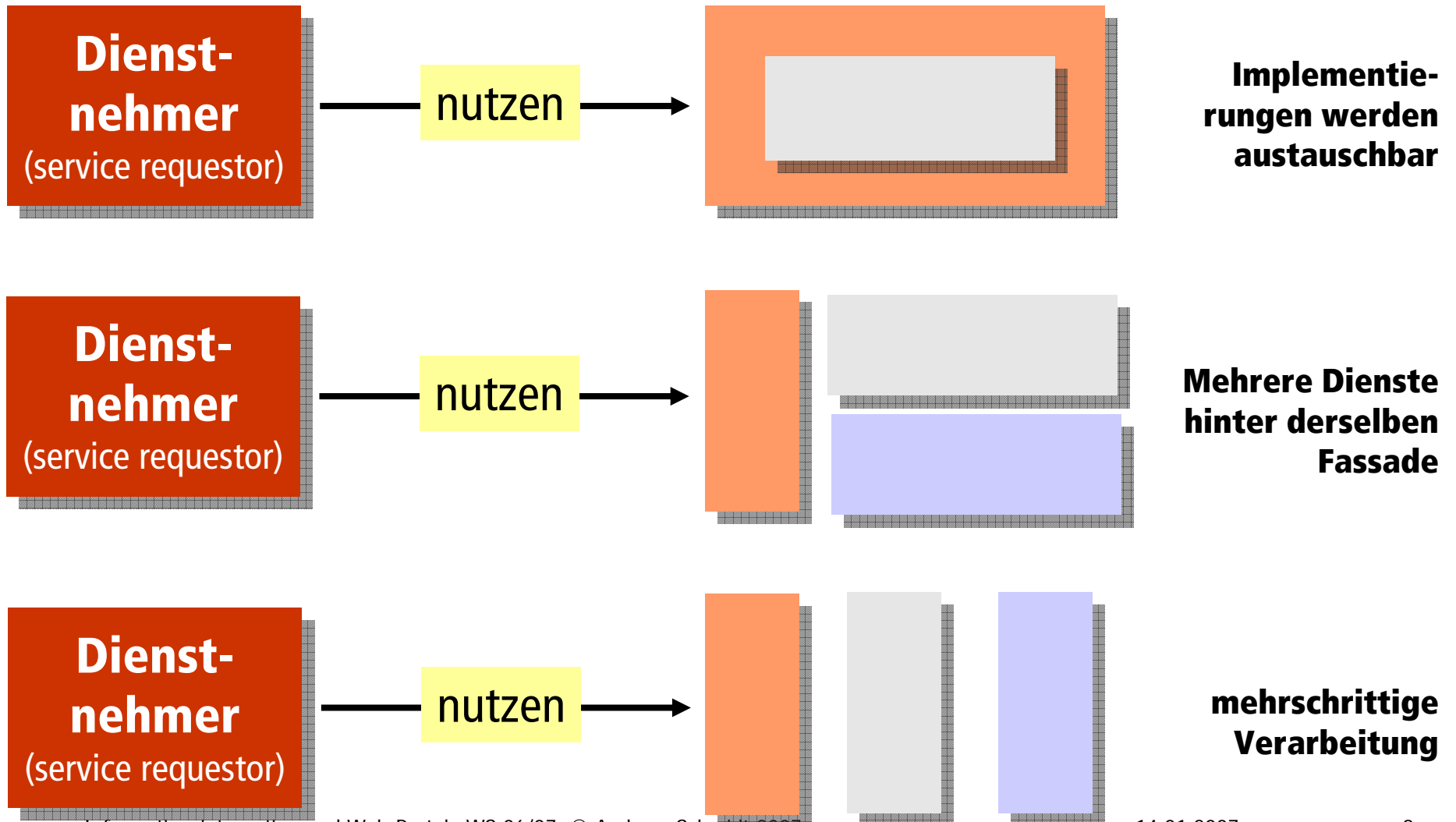
- ▶ vollkommen privat
- ▶ kann beliebig geändert werden
- ▶ insbesondere: Austausch von
Komponenten gegenüber besseren möglich

**Interne Geschäftsprozesse
(Logistik, Buchhaltung
etc.)**

Dienstorientierte Architekturen



Dienstorientierte Architekturen – Lose Kopplung



Kernprobleme für dienstorientierte Architekturen

- Kommunikationsprotokoll (technische Ebene)
 - ▶ Kodierung von Methodenaufruf und Parameterübergabe
 - ▶ Authentifizierung, Sicherheit etc.

- Dienstbeschreibung
 - ▶ Konkrete Adresse (wie erreiche ich den Dienst)
 - ▶ Dienstschnittstelle (wie rufe ich ihn auf)
 - ▶ Semantik des Dienstes (was tut der Dienst)
 - ▶ Organisatorisches (was kostet der Dienst o.ä.)

- Verzeichnisdienste

- Möglichkeiten zur Aggregation von Diensten

Interaktionsparadigmen

- Entfernter Methodenaufruf (**Remote Procedure Call**)
 - Übertragung des Methodenaufrufs in Programmiersprachen auf verteilte Umgebungen
 - Semantik liegt in der Methode und ihren Parametern
 - meist synchrone Kommunikation
 - resultiert meist in feingranularen Schnittstellen

- Nachrichtenbasierte Kommunikation (**Messaging**)
 - es werden Dokumente ausgetauscht
 - Semantik liegt in den Nachrichtentypen und ihren Instanzen
 - sehr gut für asynchrone Kommunikation geeignet
 - Messaging Queuing Systems
 - oft sehr generische Schnittstellen

Web Services



Was sind Web Services?



- Web Services sind
 - ▶ verteilte,
 - ▶ lose gekoppelte und
 - ▶ wiederverwendbare Software-Komponenten, auf die
 - ▶ über Standard-Internetprotokolle
 - ▶ programmatisch zugegriffen werden kann.

- Pragmatisch: Dienste, die mittels SOAP angesprochen werden können
 - ▶ und (meist) mittels WSDL beschrieben werden

- 1997/98
 - stark von Microsoft dominierte Entwicklung, die aber zunächst aufgrund anderer Prioritäten (XML Data u.a.) verschoben wurde
- 1999
 - SOAP 1.0
 - Hinzukommen von IBM
- 2000
 - offizielle Submission von SOAP 1.1 ans W3C
 - Gründung der XML Protocol Arbeitsgruppe am W3C
 - UDDI-Spezifikation (Ariba, IBM, Microsoft)
- 2001
 - WSDL Submission
- 2002
 - BPEL4WS (IBM, Microsoft) aus WSFL und XLANG
- 2003
 - SOAP 1.2, WSDL 2.0 Working Draft

Organisationen



- W3C: Web Services Activity
 - Web Services Architecture Group
 - XML Protocol Working Group
 - Web Services Description Working Group
- UDDI.org
 - unabhängige Industrie-Initiative
 - Ariba, IBM, Microsoft
- WSI: Web Services Interoperability
 - Interoperabilität von Web Services
 - IBM, Microsoft, Sun, SAP, BEA u.a.

Web Services – Die Technik



■ SOAP

(Simple Object Access Protocol)

- ▶ Aufrufprotokoll
- ▶ unterschiedliche Transportprotokolle: HTTP, SMTP

■ WSDL

(Web Services Description Language)

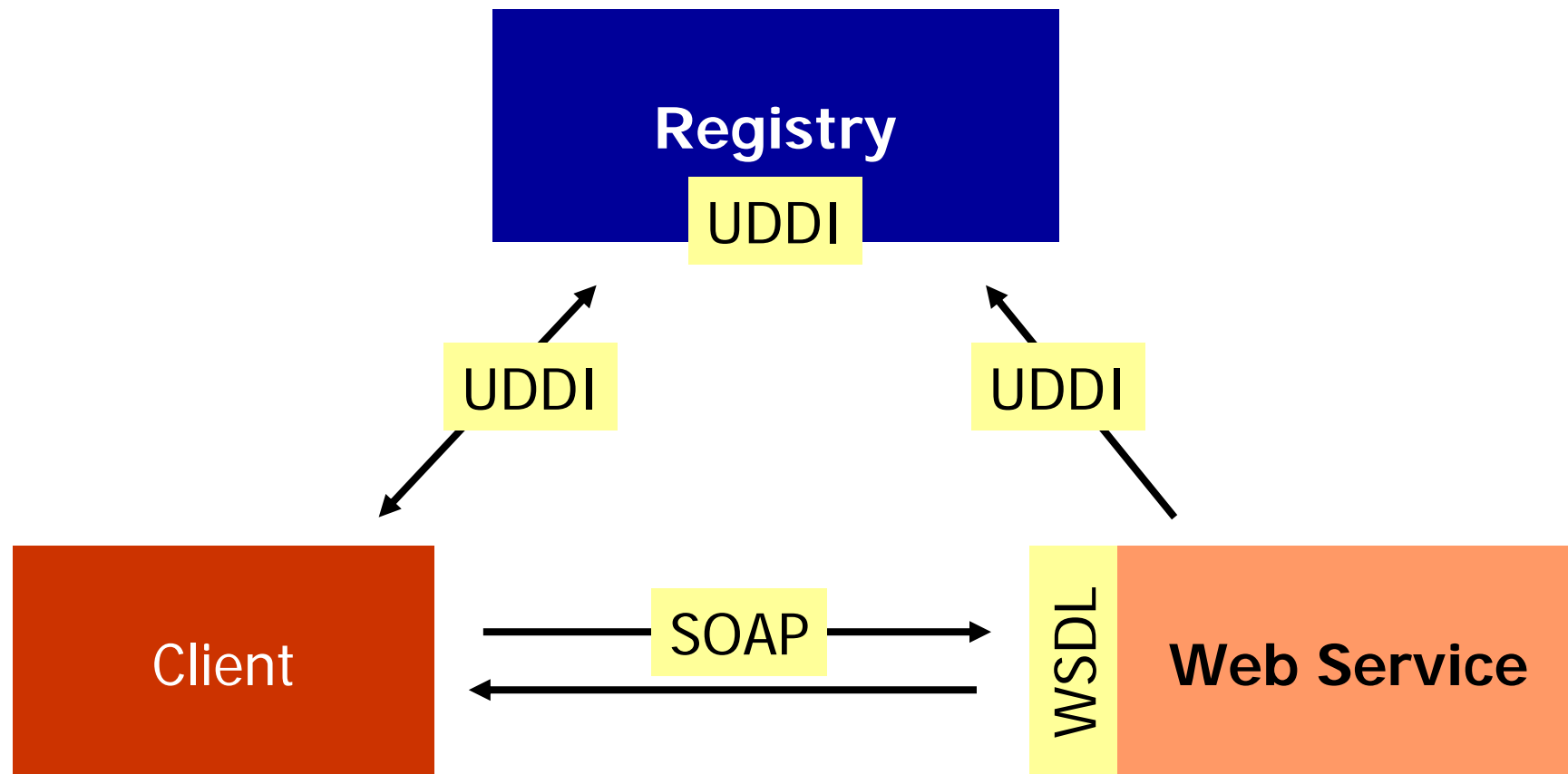
- ▶ Schnittstellenbeschreibung

■ UDDI

(Universal Description, Discovery and Integration)

- ▶ Dienstbeschreibung und Auffindung
- ▶ universeller Verzeichnisdienst für Dienstleistungen

SOAP, UDDI, WSDL



SOAP

Simple Object Access Protocol



SOAP: Überblick

- In XML kodierter entfernter Methodenaufruf (RPC) bzw. Nachrichtenaustauschprotokoll
- Beliebiges Transportprotokoll, z.B.
 - ▶ synchrone Aufrufe über HTTP:
Aufrufer wartet auf die Dienstantwort
klassischer RPC
 - ▶ asynchrone Aufrufe über SMTP (Mail):
Entkopplung von Aufruf und Antwort
Messaging
- Kodierungsregeln für Datentypen
 - ▶ Standard-Regeln entstammen XML Schema

SOAP: Aufbau eines Aufrufs

- SOAP-Envelope
 - ▶ SOAP-Header
 - z.B. Authentifizierung, Routing, Logging, Transaktionsnummern
 - Metadaten über den Aufruf
 - ▶ SOAP-Body
 - die eigentliche Methode, die aufgerufen werden soll
 - die Parameter
 - bzw. die ausgetauschte Nachricht

SOAP: Ein Beispiel

■ Aufruf

```
<soap:Envelope>  
  <soap:Body>  
    <xmlns:m="http://www.stock.org/stock" />  
    <m:GetStockPrice>  
      <m:StockName>IBM</m:StockName>  
    </m:GetStockPrice>  
  </soap:Body>  
</soap:Envelope>
```

■ Antwort

```
<soap:Envelope>  
  <soap:Body>  
    <xmlns:m="http://www.stock.org/stock" />  
    <m:GetStockPriceResponse>  
      <m:Price>34.5</m:Price>  
    </m:GetStockPriceResponse>  
  </soap:Body>  
</soap:Envelope>
```

SOAP: Amazon

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" >
  <SOAP-ENV:Body>
    <a:KeywordSearchRequest xmlns:a="urn:PI/DevCentral/SoapService" >
      <KeywordSearchRequest xsi:type="m:KeywordRequest" >
        <keyword >dog</keyword>
        <page >1</page>
        <mode>book</mode>
        <tag>webservices-20</tag>
        <type>lite</type>
        <dev-tag>your-dev-tag</dev-tag>
        <format>xml</format>
        <version>1.0</version>
      </KeywordSearchRequest>
    </a:KeywordSearchRequest>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Intermediaries



- SOAP ist auch darauf ausgelegt, daß die Verarbeitung von Dienstaufrufen über Zwischenstellen (intermediaries) abgewickelt werden kann
- Gründe
 - Unterschiedliche Sicherheitsdomänen
 - Firewalls, VPN, ...
 - Skalierbarkeit
 - z.B. Stapelverarbeitung von Nachrichten
 - Mehrwertdienste zwischenschalten
 - Verschlüsselung o.ä.
- Benötigte Metainformationen über Header-Einträge
 - mustUnderstand legt fest, ob kritisch

SOAP Encoding Rules



- Spezifizieren
 - ▶ Abbildung zwischen einem abstrakten Datenmodell und XML
 - ▶ Serialisierung eines Objektgraphen
 - ▶ Deserialisierung einer XML-Nachricht

- Sind für das RPC-Paradigma gedacht

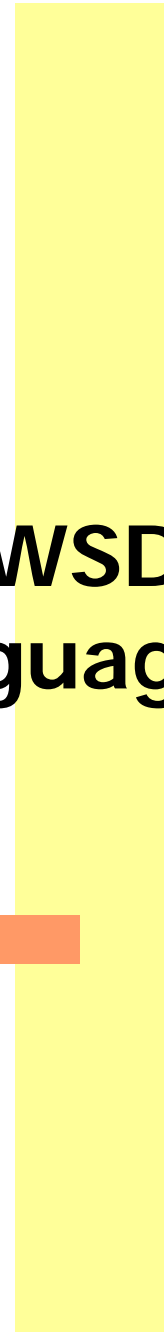
- Alternativ:
 - ▶ Jedes beliebige XML-Dokument kann als Nachricht übertragen werden
=> Messaging

SOAP und Sicherheit

- Authentisierung
 - ▶ Verteilte Authentisierungsprotokolle über SOAP (z.B. Kerberos)
- Signaturen
 - ▶ XML Signature (REC)
- Verschlüsselung
 - ▶ XML Encryption (REC)
- Transaktionen
 - ▶ BPEL4WS hat hier einige Ansätze zu bieten
 - ▶ + diverse andere Ansätze

Web Services Description Language

WSDL

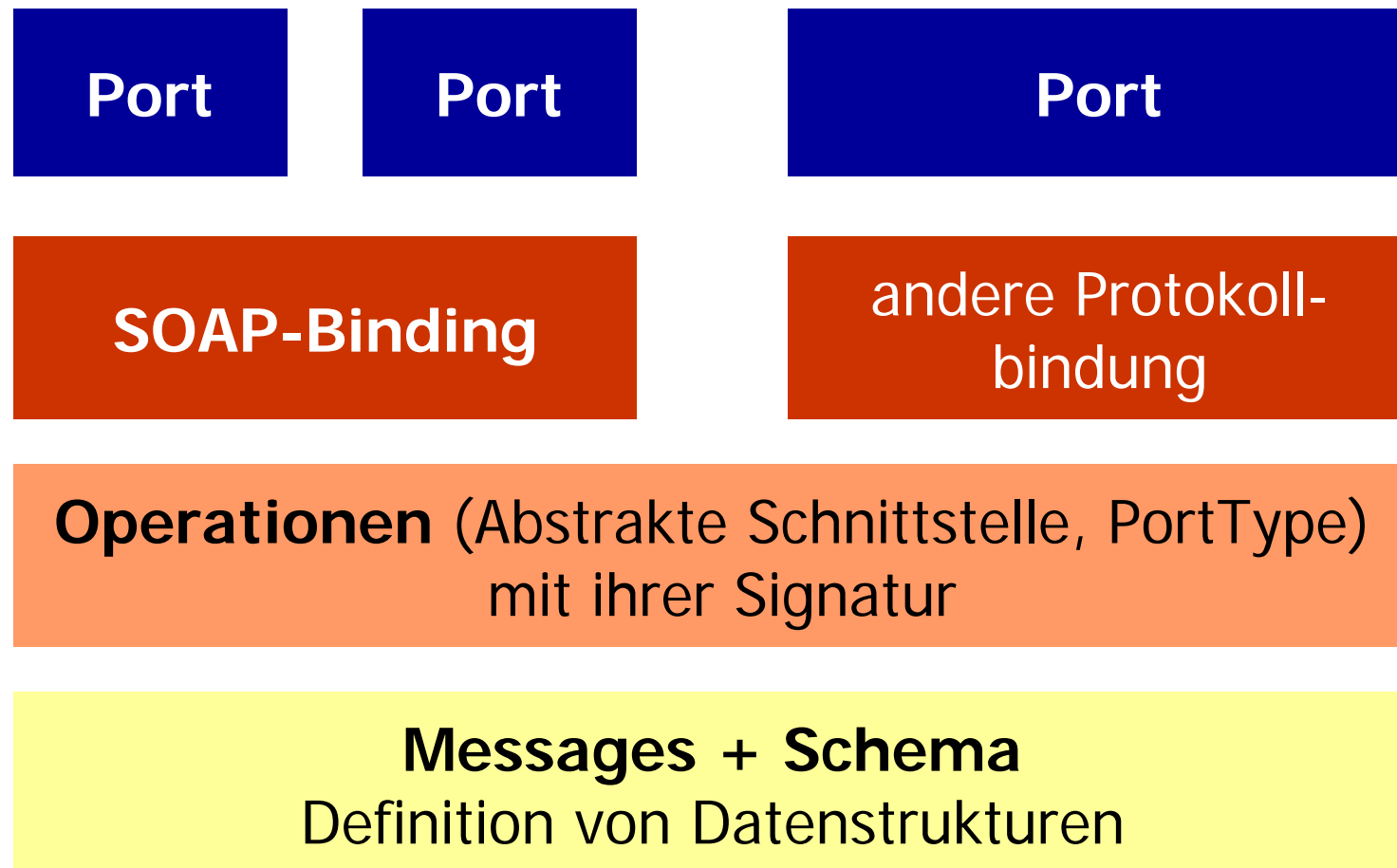


WSDL: Überblick



- Welche Operationen bietet der Dienst an?
- Welche Parameter haben die Operationen
 - ▶ Struktur der Aufrufnachricht
 - ▶ Struktur des Ergebnisses
 - ▶ Rückgriff auf XML Schema
- Wo und wie kann ich einen Dienst erreichen?
 - ▶ Adresse
 - ▶ Informationen über das Protokoll
 - z.B. SOAP, auch mehrere Protokolle!
 - verwendetes Transportprotokoll: HTTP, SMTP etc.
- aber: keine semantische Dienstbeschreibung

WSDL: Aufbau



WSDL: Aufbau (2)

```
<definitions>
  <types>
    <xsd:schema> Typdefinitionen </xsd:schema>
  </types>
  <message name="..."> Nachrichtendefinition </message>
  <portType name="...">
    <operation>
      <input message="..." /> <output message="..." />
    </operation>
  </portType>
  <binding name="...">
    <operation name="..."> <input> <soap:body ...> </input> </operation>
  </binding>
  <service name="...">
    <port name="..." binding="...">
      <soap:address location="..." />
    </port>
  </service>
</definitions>
```

Typdefinitionen

- Typdefinitionen in WSDL sind eingebettete XML Schema-Definitionen

```
<definitions name="AmazonSearch" targetNamespace="urn:PI/DevCentral/SoapService" ...>
```

```
  <types>
```

```
    <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      targetNamespace="urn:PI/DevCentral/SoapService">
```

```
      <xsd:complexType name="KeywordRequest">
```

```
        <xsd:all>
```

```
          <xsd:element name="keyword" type="xsd:string"/>
```

```
          <xsd:element name="page" type="xsd:string"/>
```

```
          <xsd:element name="mode" type="xsd:string"/>
```

```
          <xsd:element name="tag" type="xsd:string"/>
```

```
          <xsd:element name="type" type="xsd:string"/>
```

```
          <xsd:element name="devtag" type="xsd:string"/>
```

```
          <xsd:element name="version" type="xsd:string"/>
```

```
        </xsd:all>
```

```
      </xsd:complexType>
```

Typdefinitionen (2)

```
<xsd:complexType name="ProductInfo">
  <xsd:all><xsd:element name="Details" type="typens:DetailsArray"/></xsd:all>
</xsd:complexType>
<xsd:complexType name="Details">
  <xsd:all>
    <xsd:element name="Url" type="xsd:string"/>
    <xsd:element name="ProductName" type="xsd:string"/>
    <xsd:element name="Authors" type="typens:AuthorArray"/>
    <xsd:element name="ImageUrlSmall" type="xsd:string"/>
    <xsd:element name="OurPrice" type="xsd:string"/>
    <xsd:element name="Publisher" type="xsd:string"/>
    <xsd:element name="Isbn" type="xsd:string"/>
  </xsd:all>
</xsd:complexType>
<xsd:complexType name="DetailsArray">
  <xsd:complexContent>
    <xsd:restriction base="soapenc:Array">
      <xsd:attribute ref="soapenc:arrayType"
        wsdl:arrayType="typens:Details[]"/>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
```

spezieller Typ
für Arrays

- Nachrichten können auch aus mehreren Teilen bestehen
- Jeder Teil wird mit einem Schema-Typ typisiert

```
<message name="KeywordSearchRequest">
  <part name="KeywordSearchRequest"
        type="typens:KeywordRequest"/>
</message>
<message name="KeywordSearchResponse">
  <part name="return"
        type="typens:ProductInfo"/>
</message>
```

■ Schnittstellendefinition

- ▶ Name der Methode, Eingabe- und Ausgabeparameter
- ▶ Fehlerbehandlung
 - `<fault name="..." message="...">`

```
<portType name="AmazonSearchPort">  
  <operation name="KeywordSearchRequest">  
    <input message="typens:KeywordSearchRequest"/>  
    <output message="typens:KeywordSearchResponse"/>  
  </operation>  
  ...  
</portType>
```

- Protokollspezifische Angaben
 - ▶ Welches (Transport-) Protokoll?
 - SOAP over HTTP (oder SMTP, FTP), HTTP GET/POST, ...
 - ▶ Zusätzliche Angaben, die notwendig sind, um einen korrekten Aufruf zu bewerkstelligen
 - Interaktionsparadigma
 - Kodierung
 - Metainformationen (SOAP Header)

Protokollbindung

RPC als
Interaktions-
paradigma

SOAP über
HTTP

```
<binding name="AmazonSearchBinding"
  type="typeName:AmazonSearchPort">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="KeywordSearchRequest">
    <soap:operation soapAction="urn:PI/DevCentral/SoapService"/>
    <input>
      <soap:body use="encoded" namespace="urn:PI/DevCentral/SoapService"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
    </input>
    <output>
      <soap:body use="encoded" namespace="urn:PI/DevCentral/SoapService"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
    </output>
  </operation>
  ...
</binding>
```

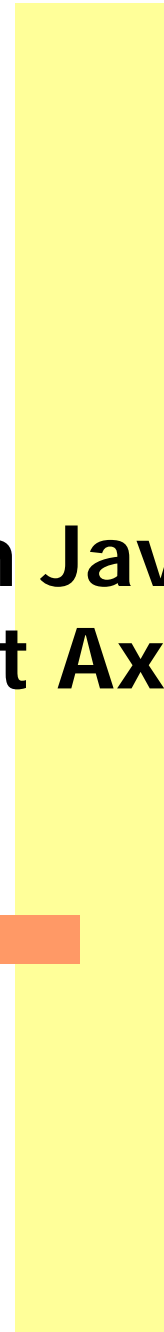
■ Festlegen der konkreten Netzwerkadresse

```
<service name="AmazonSearchService">  
  <port name="AmazonSearchPort"  
    binding="typens:AmazonSearchBinding">  
    <soap:address location="http://soap.amazon.com/onca/soap"/>  
  </port>  
</service>
```

Hier wird festgestellt, wo
der Dienst angesprochen
werden kann

- WSDL erlaubt die Beschreibung aller Informationen über die Schnittstelle und die Erreichbarkeit eines Dienstes
 - ▶ Mit Hilfe der in WSDL gemachten Angaben läßt sich problemlos ein korrekter SOAP-Aufruf schreiben
 - ▶ Beispiele:
 - .NET Einbinden von beliebigen Web Services in die IDE bzw. in Office XP (Demo folgt)
 - Java Axis erlaubt die automatische Generierung entsprechender Klassen
- Allerdings:
 - ▶ WSDL macht keinerlei Angaben, was der Dienst denn nun wirklich tut

Web Services in Java mit Axis



Überblick über Axis

- Axis ist ein sehr mächtiges Werkzeug zur Arbeit mit Web Services unter Java
 - ▶ hervorgegangen aus Apache SOAP bzw. SOAP4J von IBM
 - ▶ unterstützt die Arbeit mit SOAP und WSDL
 - ▶ auf Performanz (stromorientiert) und Erweiterbarkeit (protokollunabhängig) ausgelegt
- Baut auf der javax.xml.rpc-API auf
 - ▶ Standard-API in der J2EE 1.4
 - ▶ Schnittstellen und Spezifikationen für das Mapping von Web Services und Java-Klassen
- Für die Erstellung von Web Services:
 - ▶ Basis ist ein Application Server (z.B. Apache Tomcat)

Aufrufen eines Web Services

- Axis bringt eine Klassenbibliothek mit, die eine Erstellung von SOAP-Aufrufen "zu Fuß" erlaubt

```
String endpoint = "http://nagoya.apache.org:5049/axis/services/echo";  
Service service = new Service();  
Call call = (Call) service.createCall();
```

```
call.setTargetEndpointAddress( new java.net.URL(endpoint) );  
call.setOperationName(  
    new QName("http://soapinterop.org/", "echoString"));
```

```
String ret = (String) call.invoke( new Object[] { "Hello!" } );
```

```
System.out.println("Sent 'Hello!', got '" + ret + "'");
```

Aufrufen eines Web Services (2)

- Allerdings: man wünscht sich immer weitestgehende Transparenz, d.h. Verbergen der verteilten Aufrufe
 - ▶ Zugriff wie auf normale Java-Klassen
- Deshalb: Generierung von Java Stubs
 - ▶ analog zu CORBA
- WSDL2Java generiert automatisch entsprechende Java-Klassen, so daß eine "normale" Java-API zur Verfügung steht
 - ▶ Vgl. idl2java für CORBA

Abbildungsprozess

Ports/Service



Dienstschnittstelle als Interface
Eigentliche Proxy-Klasse (Locator)

SOAP-Binding



Für jedes Binding ein Stub

Operationen/PortType



Für jeden PortType ein Java-Interface

Messages + Schema



Java-Klassen für jeden komplexen Typ

Erstellung von Web Services



- Einfache Methode: JWS-Dateien
 - ▶ Installation von AXIS im Apache Tomcat
 - ▶ Kopieren von Java-Klasse mit Endung .jws in entsprechendes Verzeichnis
 - Aber: Einschränkungen hinsichtlich der Klassen
 - ▶ Fertig! (wie bei JSP)
 - Axis erzeugt automatisch alles weitere
 - ▶ WSDL-Spezifikation
 - Ist automatisch abfragbar mit ?WSDL an den Dienstnamen angehängt

Erstellung von Web Services (2)

■ Web Service Deployment Descriptors

- ▶ Erstellen der Java-Schnittstelle und konventionelle Java-Programmierung
- ▶ Spezifikation, was als Web Service verfügbar gemacht werden soll, mittels WSDD
- ▶ Verkettung unterschiedlicher Dienste möglich

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <service name="MyService" provider="java:RPC">
    <parameter name="className"
      value="samples.userguide.example3.MyService"/>
    <parameter name="allowedMethods" value="*" />
  </service>
</deployment>
```

- Auf den Web-Seiten zur Vorlesung gibt es ein Tutorial zum Thema Web Services mit AXIS
 - ▶ Einführung
 - ▶ geführte Beispiele für das Erstellen und Nutzen von Web Services
 - ▶ Experimentierumgebung



UDDI
Universal Description,
Discovery and Integration

UDDI: Überblick

- **U**niverselle, zentrale "E-Business"-Suchmaschine mit verteilter Konzeption (UDDI Business Registry)
 - Informationen über Unternehmen abrufen (**D**escription)
 - Geschäftspartner auffinden (**D**iscovery)
 - Geschäftsprozesse über Unternehmensgrenzen hinaus beschreiben und nutzbar machen (**I**ntegration)

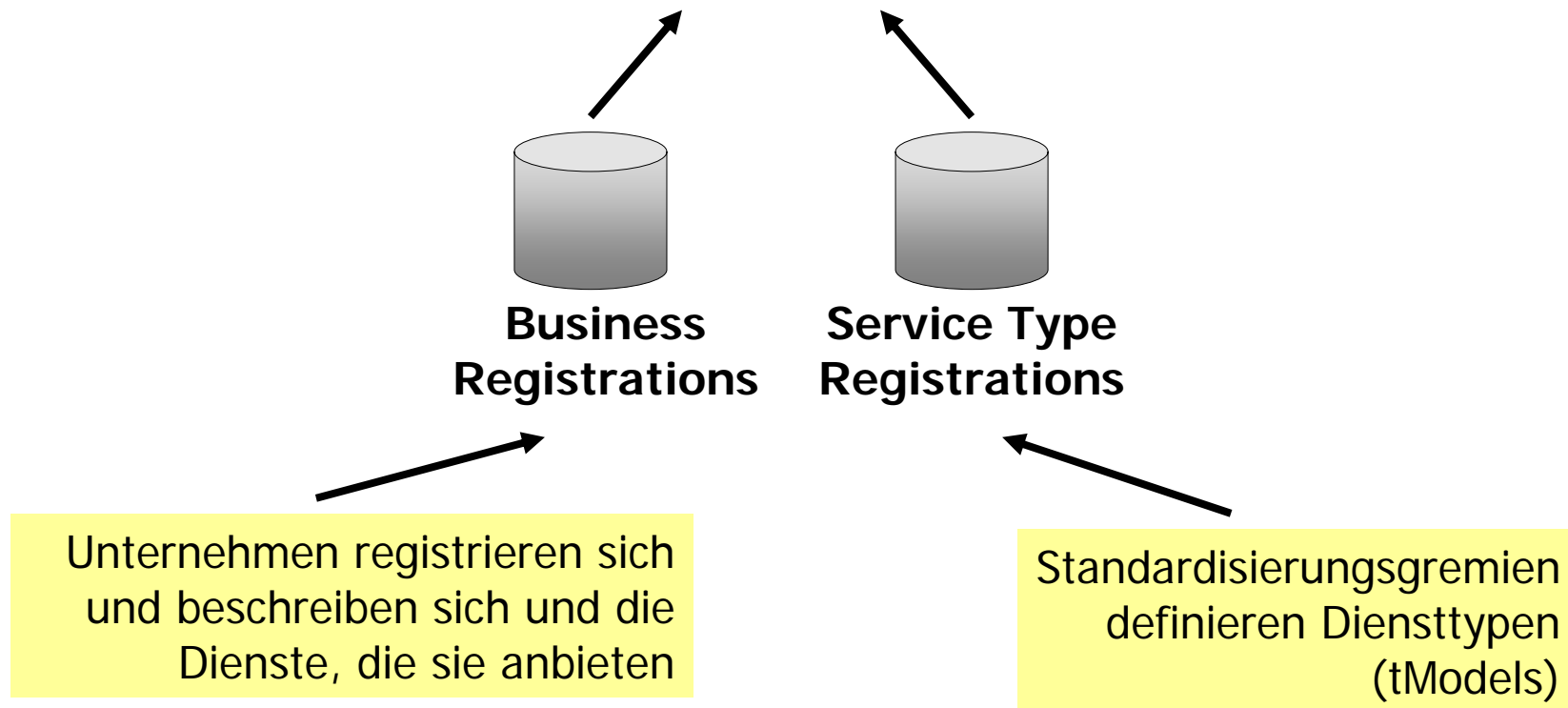
- **E**ntwicklung begann im Frühjahr 2000
 - Ariba, IBM, Microsoft
 - Erste Registry im Mai 2001

UDDI: Überblick (2)

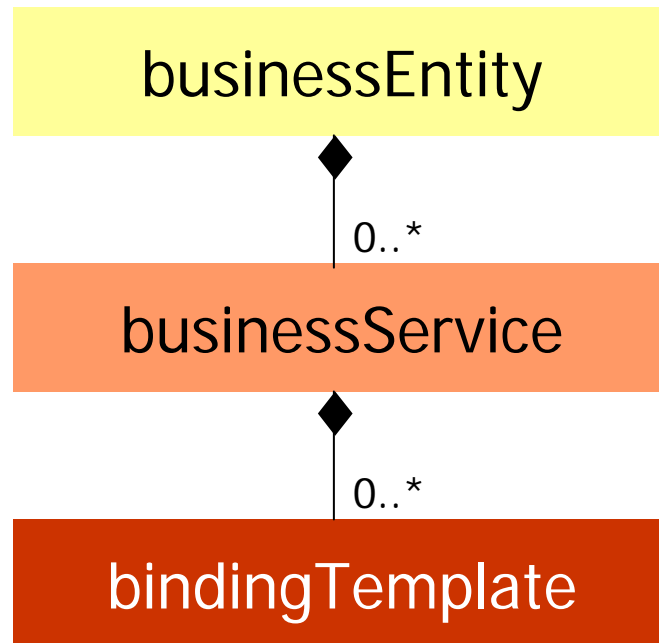
- Universeller Verzeichnisdienst
 - ▶ *white pages*: Informationen über Unternehmen (Namensregister, Details, Kontaktinformationen)
 - ▶ *yellow pages*: »Branchenbuch« - Ordnung der Unternehmen nach Technologiecodes, Branchentaxonomien oder geographischem Ort
 - ▶ *green pages*: Geschäftsmodell, technische Informationen über Nachrichtenformate, Protokolle (= > WSDL), Informationen über Geschäftsprozesse
 - ▶ *service type registrations*: Referenzen auf Beschreibungen des jeweiligen Typs
- primärer Zweck ist nicht unbedingt das *automatische* Auffinden

- Informationen über das anbietende Unternehmen
 - ▶ auch Taxonomie nach Branche o.ä. möglich
- Beschreibung des angebotenen Dienstes
 - ▶ nicht nur für Web Services
 - ▶ Kategorisierung des Dienstes (Thesauri) nach
 - Branche
 - Art der Dienstleistung
 - geographischer Ort
 - ▶ Technische Schnittstelle (z.B. WSDL, aber auch natürlich sprachliche Beschreibungen)
- Protokoll zum Veröffentlichen und Nachschlagen von Dienstinformationen (Basis: SOAP)
 - ▶ verteilte Registries

Nutzung der Daten über Suchmaschinen oder aus Anwendungen direkt heraus, um Dienste anderer Unternehmen zu finden und in eigene Anwendungen zu integrieren



Datenstrukturen



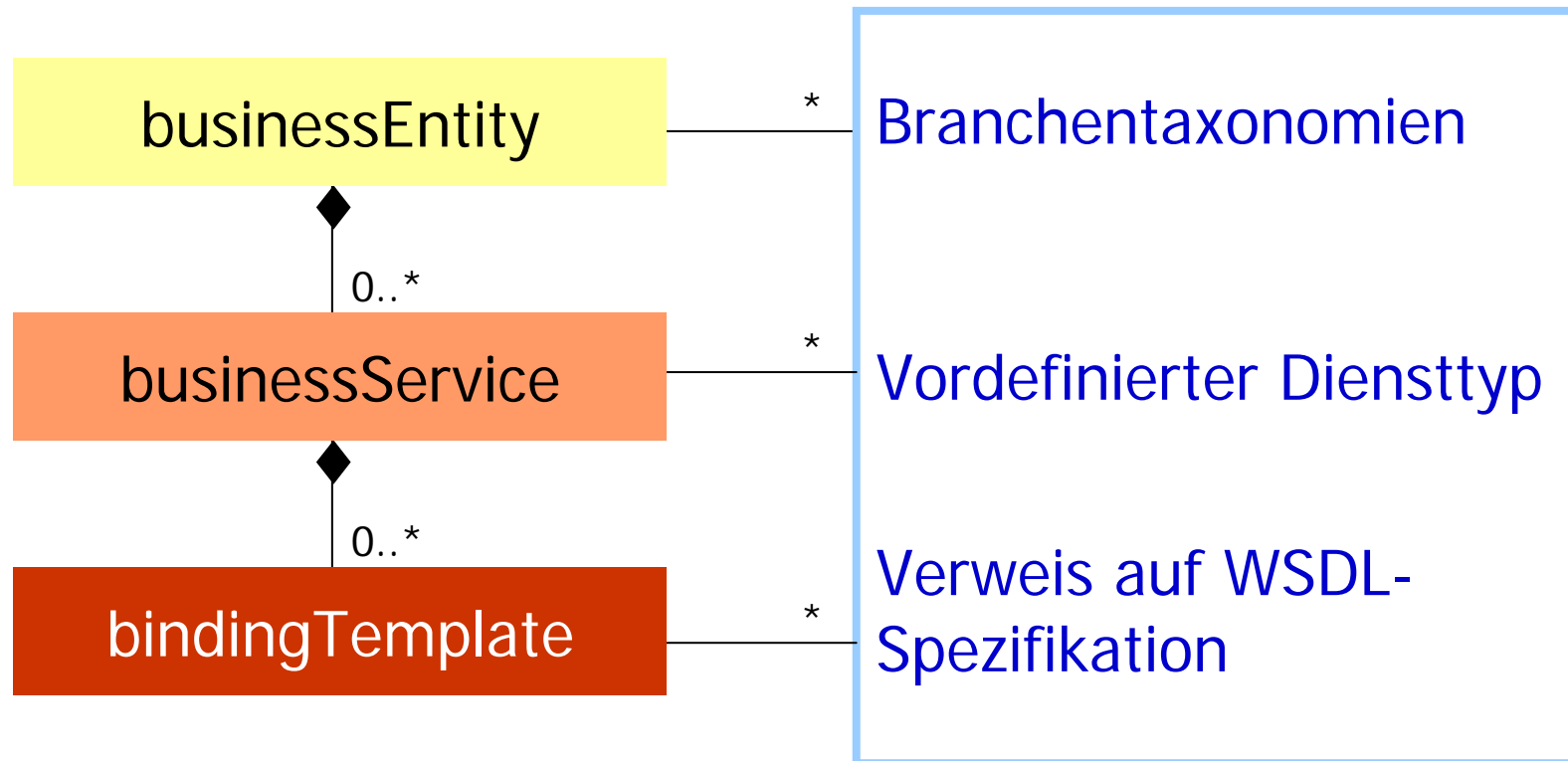
Name, Beschreibung, Kontakt
URL-Verweise auf andere businessEntities
Zuordnung zu Taxonomien

Name, Beschreibung,
Zuordnung der Dienste zu Kategorien

Name, Beschreibung,
Kommunikationsmöglichkeiten (http, phone)
Zugriffspunktdefinition (Adresse)

Technology Models (tModel)

- tModels sind zunächst nur eindeutige Identifikatoren



Musterablauf Protokoll

Browse
Suchen

find_business
find_service

Drill Down
Details abrufen

get_businessDetail
get_serviceDetail

Invocation
Dienst aufrufen

get_bindingDetail
liefert WSDL-Information
anschließend SOAP

Dienstintegration



Dienstintegration



- Was fehlt jetzt noch für eine Dienst**integration**?
 - ▶ Wie kann man das Zusammenwirken von Diensten zum Erbringen von Mehrwertfunktionalität auf abstrakter Ebene beschreiben?
- Analog zu Abbildungsregeln bei der Informationsintegration

- Idee einer Programmierung auf höhere Ebene
- Kombination von Dienstaufrufen zur Erbringung eines »höherwertigen« Dienstes
- Stichwort **Orchestrierung**:
 - neuer komplexer Dienst entsteht auf der Basis von Aufrufen von Einzeldiensten
 - Ausführung wird von einem Interpreter („dem Dirigenten“) gesteuert
- Vorteil:
 - einfache Anpassungen im Rahmen des Betriebs können mittels graphischen Werkzeugen erledigt werden
 - langfristig: bessere Verwaltung von Abhängigkeiten möglich

Choreographie



- In komplexen Szenarien, in denen autonome Dienste zusammenwirken, braucht es Regeln für die Interaktion
 - ▶ Analogie: soziale Systeme brauchen auch Konventionen

- Stichwort **Choreographie**:
 - ▶ beschreibt das Zusammenwirken von Diensten von außen
 - ▶ betrachtet nur das beobachtbare Verhalten von Diensten
 - ▶ „Choreographie ist etwas, an das sich alle Beteiligten halten“
 - im Gegensatz zu einem Dirigenten, der etwas vorschreibt

- Beschreibt also Protokolle!

BPEL4WS
Business Process Execution Language
for Web Services



- BPEL4WS kümmert sich (im wesentlichen) um die Orchestrierung von Web Services
- Grundperspektive von BPEL4WS
 - ▶ Wie wirken unterschiedliche Web Services im Rahmen von Geschäftsprozessen zusammen?
 - ▶ Prozeßbeschreibungssprache (abstrakt, aber auch ausführbar)

Grundschemata

```
<process>
```

```
<partners>
```

```
...
```

```
</partners>
```

```
<containers>
```

```
...
```

```
</containers>
```

```
<faultHandlers>
```

```
...
```

```
</faultHandlers>
```

```
<flow>
```

```
...
```

```
</flow>
```

```
</process>
```

Deklarieren der beteiligten Partner und ihrer **Rollen** mittels in WSDL deklarierten ServiceLinkTypes

```
<partner name="" serviceLinkType=""  
myRole="" partnerRole="">
```

Zustandsvariablen der Prozeßinstanz

```
<container name="" messageType="" />
```

Fehlerbehandlung: Definiert, welche Nachricht im Fehlerfall zurückgegeben werden

Aktivitäten

Primitive für Aktivitäten: Dienstaufrufe

■ Parameter

- ▶ partner,
- ▶ portType
- ▶ operation
- ▶ inputContainer, outputContainer

<receive ... >

- Wartet auf eine eingehende Nachricht
- Initiiert Aktivitäten
- Alternative: <pick> <onMessage ...> </onMessage > </pick>
 - Wartet auf eine von mehreren möglichen Nachrichten

<reply ... >

- Schickt eine Antwort an den entsprechenden Dienst

<invoke ... >

- Ruft einen Dienst auf

Primitive für Aktivitäten: Programmiersprachenelemente

■ `<assign>`

- ▶ Zuweisungsoperation `<from> ... <to>`
- ▶ Flexible Möglichkeiten über XML-basierte Strukturen
 - `<from container="" part="ncname" query="queryString" />`
 - `<from expression="" />`
 - `<to container="" part="" query="" />`
- ▶ XPath als Ausdruckssprache und Anfragesprache

■ `<switch>...<case condition=""> ... </switch>`

- ▶ Auswahlkonstrukt

■ `<while condition="">`

- ▶ Schleife

Aktivitäten: Weitere Sprachelemente

■ Fehlerbehandlung

- ▶ Analog zu Javas Exception-Mechanismus

```
<throw faultName="" />
```

```
<catch faultName="">...</catch>
```

```
<catchAll>...</catchAll>
```

Synchronisation

- BPEL4WS erlaubt die Spezifikation nebenläufiger Prozesse (`<flow>`)
- Sequentielle Prozeßabschnitte werden in `<sequence>`-Elemente eingeschlossen
- Will man Barrieren, so gibt es hierfür die Möglichkeit, Abhängigkeiten zu definieren
 - Am Beginn eines Prozeßabschnittes: Verbindungsdeklarationen mittels `<link>`
 - Verknüpfungspunkte: `<source>` bzw. `<target>`
 - Entspricht in Java `notify()` bzw. `wait()`
 - Zusätzliche Bedingungen sind auch möglich

Synchronisation (2)

<flow>

<sequence>

A

<source linkName="x"/>

</sequence>

</flow>

<sequence>

B

B wartet, bis A an der Verknüpfungsstelle angekommen ist

<target linkName="x"/>

</sequence>

Transaktionen

- BPEL4WS unterstützt Transaktionen
- Allerdings: keine ACID-Transaktionen
 - ▶ Atomizität ist für langlebige Transaktionen problematisch
 - ▶ Keine Transaktionsunterstützung bei den Diensten voraussetzbar
 - ▶ z.B. Zwei-Phasen-Commit setzt ACID-fähige Dienste voraus!
- Stattdessen:
 - ▶ Offen geschachtelte Transaktionen ("Sagas")
 - Vgl. Vorlesung Transaktionsverwaltung
 - ▶ Zwischenzustände werden bereits sichtbar
 - ▶ Im Rücksetzfall werden Kompensationen eingesetzt, die einen konsistenten Zustand wiederherstellen (sollen)

Transaktionen (2)

```
<scope name="A">
```

```
  <compensationHandler>
```

Aktionen, die im Fehlerfall aufrufen sind

```
</compensationHandler>
```

Aktivitäten

```
  <scope name="B">
```

```
    <compensationHandler>...</compensationHandler>
```

Aktivitäten

```
    <compensate/>
```

```
  </scope>
```

Aktivitäten

```
</scope>
```

- "Schlüssel" für Prozeßinstanzen
 - ▶ Spezifikation, welche Attribute (in den Nachrichtenparametern) es erlauben, die empfangene Nachricht einer Prozeßinstanz zuzuordnen
 - Beispiel: Bestellnummer für Bestellung, Statusanfrage etc.
 - ▶ Bei Empfang wird das entsprechende Attribute extrahiert und nach vorhandenen Prozeßinstanzen gesucht
 - ▶ Dadurch kann eine mehrschrittige verbindungslose Kommunikation stattfinden
- Korrelationen werden den <receive>, <reply> und <invoke>-Elementen zugeordnet



Fazit

Was ist das Neue daran?



- dienstorientierte Architektur
 - ▶ Web Services werden als echte Dienstleistungen begriffen, die man nach außen auch anbieten kann
- Granularität
 - ▶ grobe Granularität
- Rahmenbedingungen
 - ▶ Web-Umgebungen
 - ▶ Heterogenität
 - ▶ Firewall-Problematik
- (Die Technik ist noch relativ einfach)

Vergleichbare Technologien



■ CORBA

- ▶ weniger Autonomie der Dienste
- ▶ für lose Kopplung wenig geeignet
- ▶ performanter, dadurch feingranularere Verteilung
- ▶ trotz IIOP keine wirkliche Internet-Technologie

■ COM/DCOM

- ▶ ebenso eher Intranet-Technologie
- ▶ keine Plattformunabhängigkeit

Firewalls?

- Web Services (insbesondere SOAP) treten auch mit dem Argument an, daß die Firewall-Problematik, die den Einsatz anderer Middleware-Technologien wie CORBA schwierig machte, umgangen wird
 - ▶ HTTP als Transportprotokoll ist meist freigeschaltet
 - ▶ aber: HTTP büßt immer mehr seine Harmlosigkeit ein
 - Dienste mit beliebigen Auswirkungen können aufgerufen werden
 - Sinn einer Firewall?
- => Damit wird wohl auch SOAP zukünftig Firewall-Restriktionen in Kauf nehmen müssen.

Vorteile von Web Services



- breite Unterstützung der Software-Hersteller
 - Microsoft, IBM, Oracle, ...
- niedrige Einstiegshürden
 - SOAP leicht zu verstehen, wenn man XML kennt
 - WSDL und UDDI sind zunächst einmal optional, um Web Services zu nutzen!
 - leicht kombinierbar mit existierenden Technologien (Servlets, JSP/ASP)
- durchgängige Verwendung von XML
 - und die meisten Anwendungen können in der einen oder anderen Form bereits XML »sprechen«
- für Internet-Umgebungen entwickelt
 - Standard-Protokolle
 - Rahmenbedingungen berücksichtigt

- Sicherheit?
 - ▶ mit HTTPS (SSL) läßt sich Client- und Serverauthentisierung und verschlüsselter Datentransfer auf Transport-Protokollebene erreichen
 - ▶ XML Encryption / Digital Signature kann auf Anwendungsebene eingesetzt werden
- Performanz?
 - ▶ Web Services erlauben nur grobgranulare Verteilung
 - ▶ Kompressionstechniken für große Datenmengen
- Erweiterte Konzepte zur Verteilung?
 - ▶ Objektreferenzen?
- Wildwuchs an neuen Standards
- Ausgereift?

Die Web-Service-Welt

Geschäftsprozesse

BPEL
WS-Choreography
WS-Choreography Description Language

Management

BPML
WSMF
WS-Management
MUWS
WS-Events
MOWS
WSMF-WSM

Sicherheit

WS-Security
WS-Trust
WS-Security Policy
WS-Federation
WS-Secure Conversat.

Zuverlässigkeit

WS-Reliability
WS-Reliable Messag.

Transaktionen

WS-Business Activity
WS-Atomic Transact.
WS-Coordination

Ressourcen

Web Services
Resources Framework

Metadaten

WSDL
WS-Discovery
UDDI
WS-Policy
WS-Metadata-Exchange

Nachrichten

SOAP
WS-Notification
WS-Adressing

- Viele weitere Web Services Initiativen, darunter
 - ▶ WS-Coordination
 - ▶ WS-Transaction für ACID-Transaktionen
- Semantik
 - ▶ Semantische Dienstbeschreibungen mit ontologiebasierten Verfahren
- Kombination von Dienst- und Informationsintegration
 - ▶ Skalierbarkeit in vielerlei Hinsicht bei datenintensiven Anwendungen
 - ▶ Offene, verteilte und dienstorientierte Integrationsinfrastruktur erforderlich

- SOAP, WSDL, UDDI
 - ▶ Graham et al., Building Web Services with Java, SAMS Publishing, 2002
- Axis
 - ▶ <http://xml.apache.org/axis/>
- BPEL4WS
 - ▶ Spezifikation
<http://www-106.ibm.com/developerworks/library/ws-bpel/>
 - ▶ Implementierung (Editor + Ausführungsmaschine)
<http://www.alphaworks.ibm.com/tech/bpws4j>
- Vorlesung Transaktionsverwaltung