

Vorlesung Netzbasierte Informationssysteme

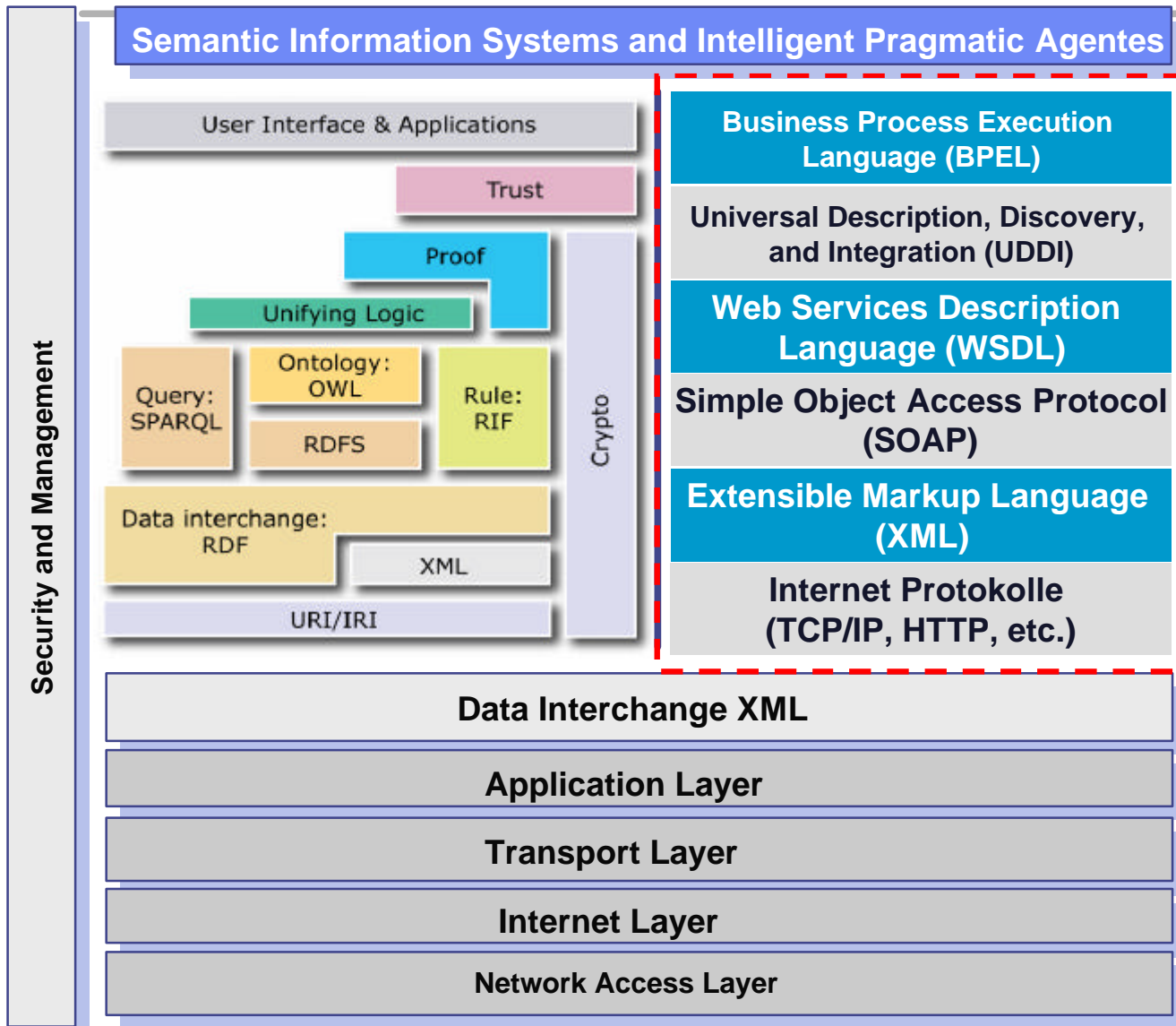
Web Services

Prof. Dr. Adrian Paschke

Arbeitsgruppe Corporate Semantic Web (AG-CSW)
Institut für Informatik, Freie Universität Berlin
paschke@inf.fu-berlin.de
<http://www.inf.fu-berlin.de/groups/ag-csw/>



Überblick



Corporate Semantic Web / Pragmatic Web

Semantic Web
Technologien

Web Services
Technologien

XML, DTD, XSD, DOM, SAX, ...

FTP, SMTP, HTTP, TELNET,
NEWS

TCP, UDP

IP (mit ICMP, ARP)

SLIP, PPP,
IEEE 802.3, 802.11 ...



Software as a (Web) Service

- SOA - Service-oriented Architecture
 - 3 Basic Concepts: Service Description, (Publish, Discovery), Invocation
- WS(A) - Web Services Architecture
 - SOAP (Invocation)
 - WSDL (Web Service Description)
 - UDDI (Publish, Discovery (Search and Find))

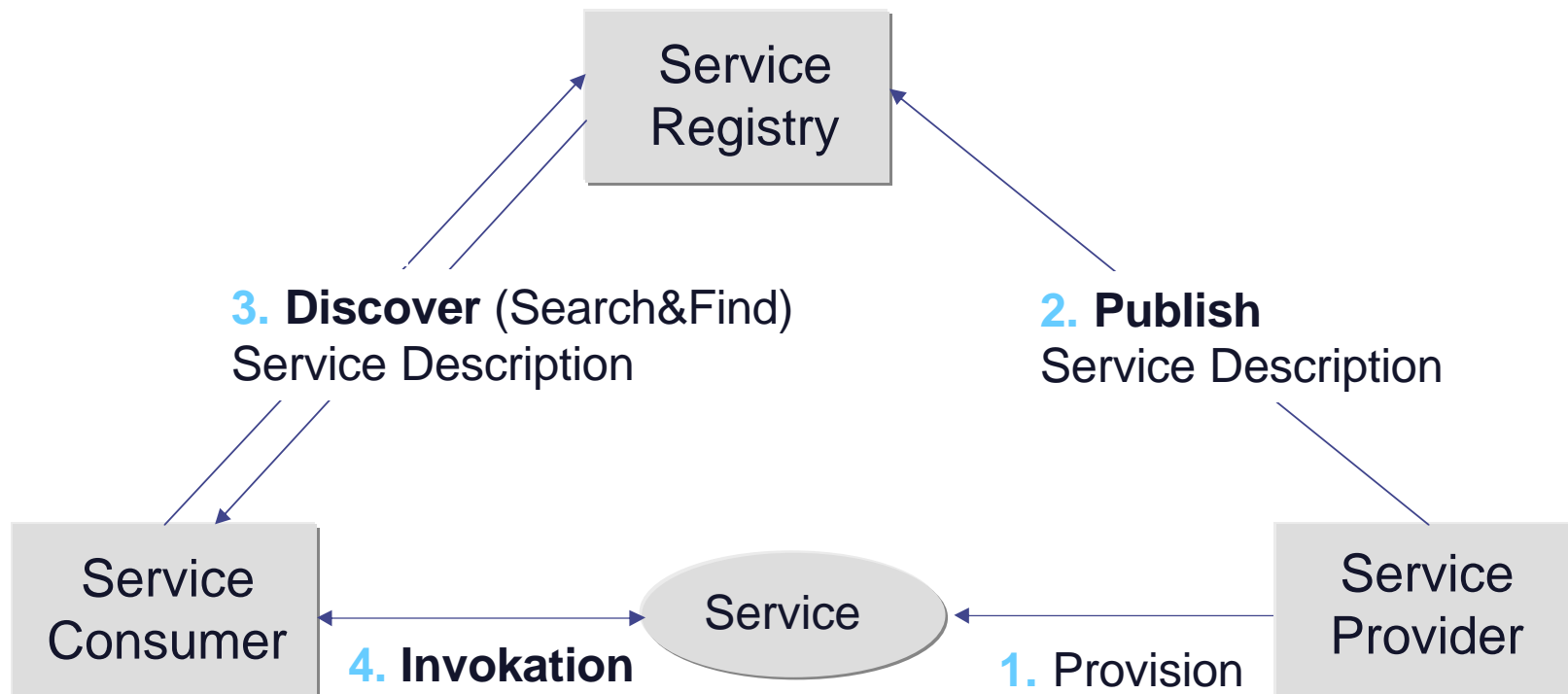
Definition: Service Oriented Architecture

- W3C: „ A Set of components which can be **invoked**, and whose **interface description** can be **published** and **discovered**“

- CDBi: „Services can be **invoked**, **published** and **discovered**, and are abstracted away from the implementation using a **standards based interface**“

- Service Interface Description
- Publish and Discover (Interface Description)
- Service Invocation
- Standardization

SOA Dreieck: Rollen und Aktivitäten

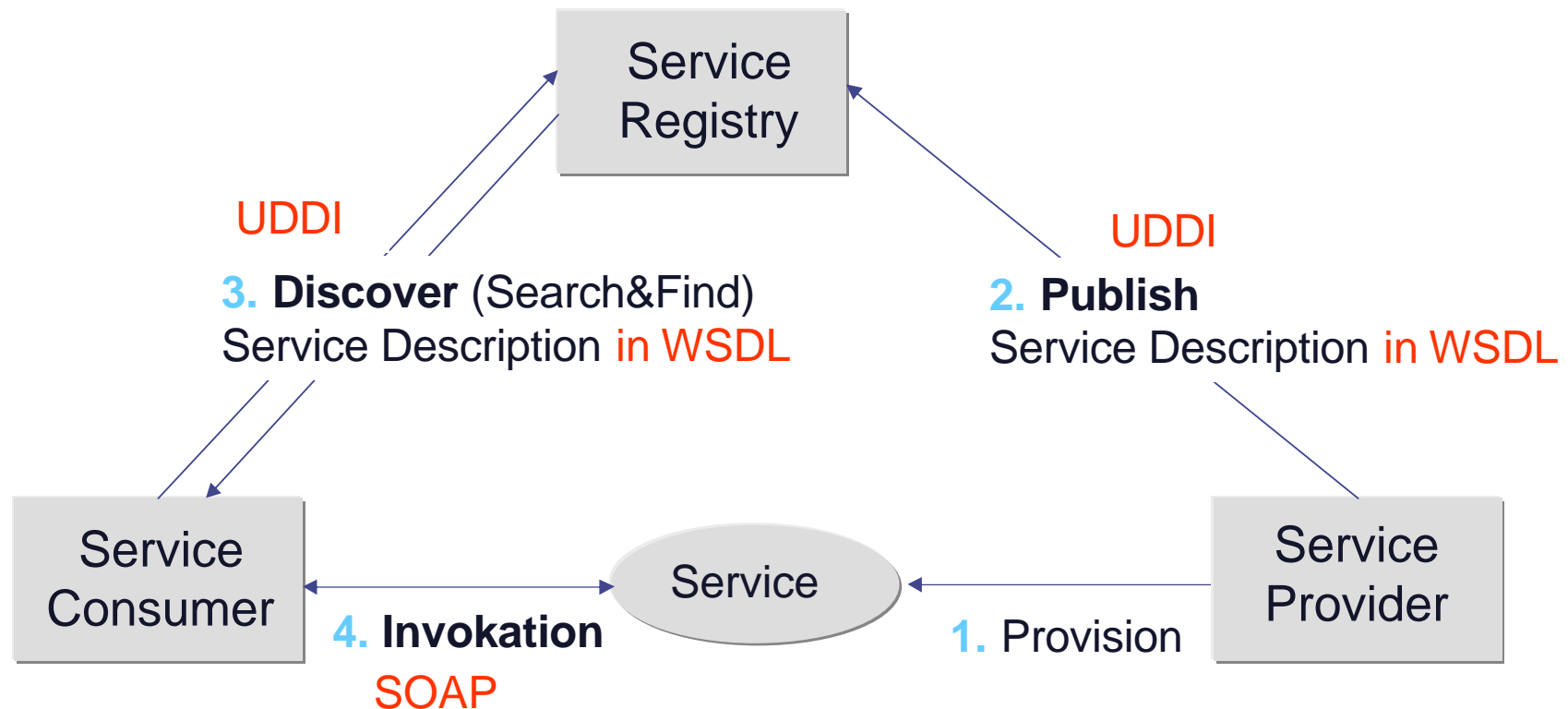


Definition: Web Services

- W3C „A Web service is a software system designed to support interoperable **machine-to-machine interaction** over a network. It has an **interface described** in a machine-processable format (specifically **WSDL**). Other **systems interact** with the Web service in a manner prescribed by its description using **SOAP** messages, **typically conveyed using HTTP** with an XML serialization in conjunction with other Web-related standards”

- Verzeichnisdienst (Service Registry) nicht erwähnt
 - Kein notwendiges Kriterium für den Einsatz von WS
...solange alle verwendeten Dienstbeschreibung vorab bekannt
 - Jedoch: Mit Verzeichnisdienst (**UDDI**) wird WS Architektur zu SOA-Architektur
- WS-Architekturbeschreibung gehen i.d.R. von UDDI Existenz aus

SOA Dreieck: Rollen und Aktivitäten

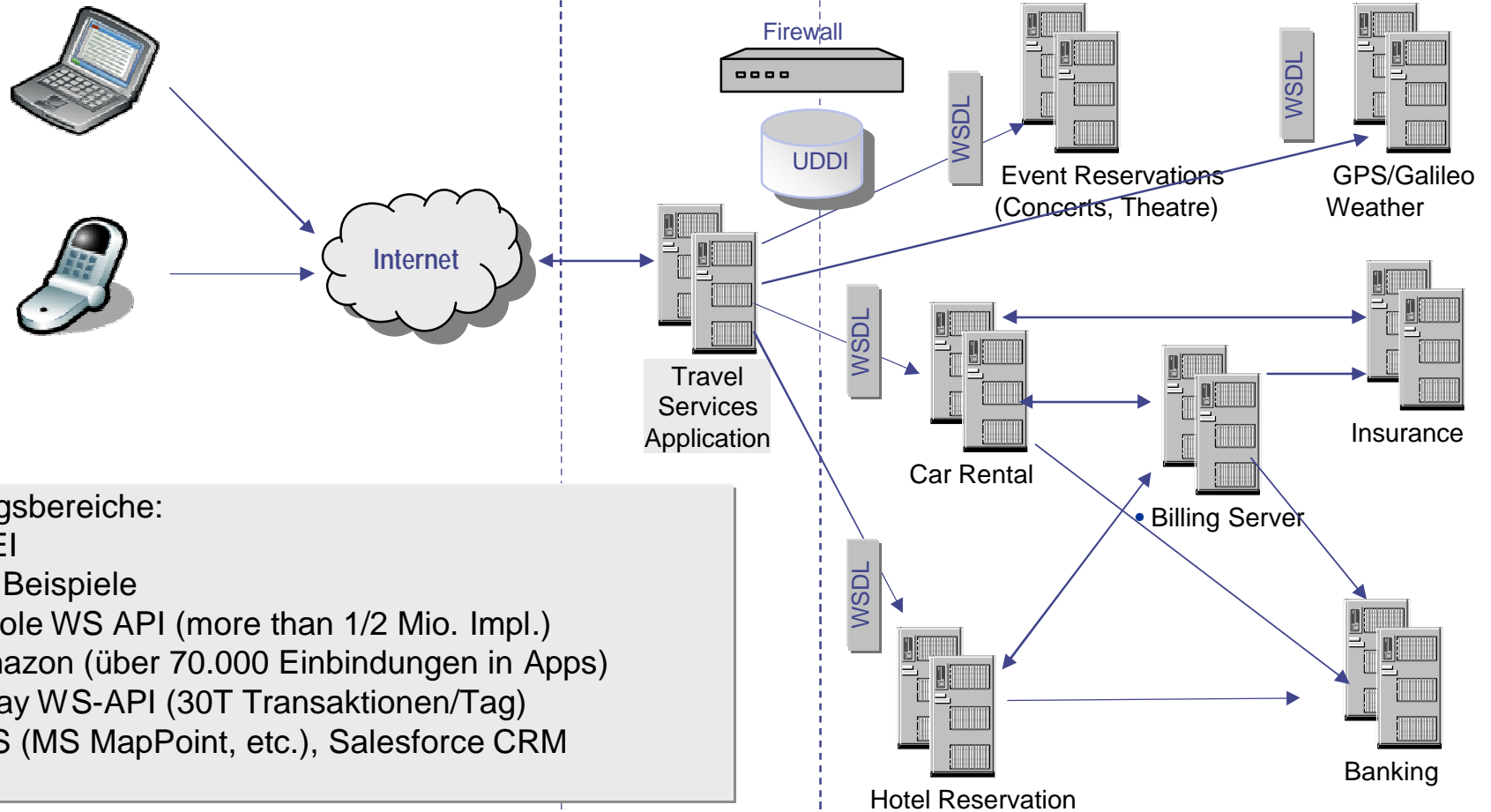


Realisierung mittels WS-Architektur

Ziele von WS (SOA)

- Lose Koppelung von Komponenten
- Dynamische Einbindung von Diensten
- Standardisierte Dienste, etablierte Internetprotokolle
- Wiederverwendung von Code
- Aggregation (Orchestration) verschiedener Dienste über Unternehmensgrenzen (Interworkflow) hinweg
- Globale Web Service Registerdatenbanken
- Dienste vergleichbar machen
- Kluft zwischen großen und kleinen Betrieben überwinden

Business-Services / IT-Services auf Anwendungsebene



Anwendungsbereiche:

- EAI, IEI
- einige Beispiele
 - Google WS API (more than 1/2 Mio. Impl.)
 - Amazon (über 70.000 Einbindungen in Apps)
 - eBay WS-API (30T Transaktionen/Tag)
 - GIS (MS MapPoint, etc.), Salesforce CRM

WS Protokollschichten

- Basis Schichten sind bereits etabliert (HTTP – WSDL)
- Höhere Schichten befinden sich noch im Aufbau
- UDDI und BPEL sind dennoch Bestandteile der Architektur
- Weitere Konzepte:
 - Sicherheit
 - Authentifizierung
 - Transaktionsmanagement
 - Quality and Service Management
 - ...

Business Process Execution Language (WS-BPEL)

Universal Description, Discovery, and Integration (UDDI)

Web Services Description Language (WSDL)

Simple Object Access Protocol (SOAP)

Extensible Markup Language (XML)

Internet Protokolle (TCP/IP, HTTP, FTP, etc.)

Abgrenzung SOA-WS

- SOA muss nicht mittels WS Technologie umgesetzt werden (Corba, RMI, ...)
- WS Technologie muss nicht SOA-basiert sein
- Aber...
 - Prognose Gartner Group: „Through 2008, SOA and WS will be implemented together in more than 75% of new SOA or WS projects“ (2005)

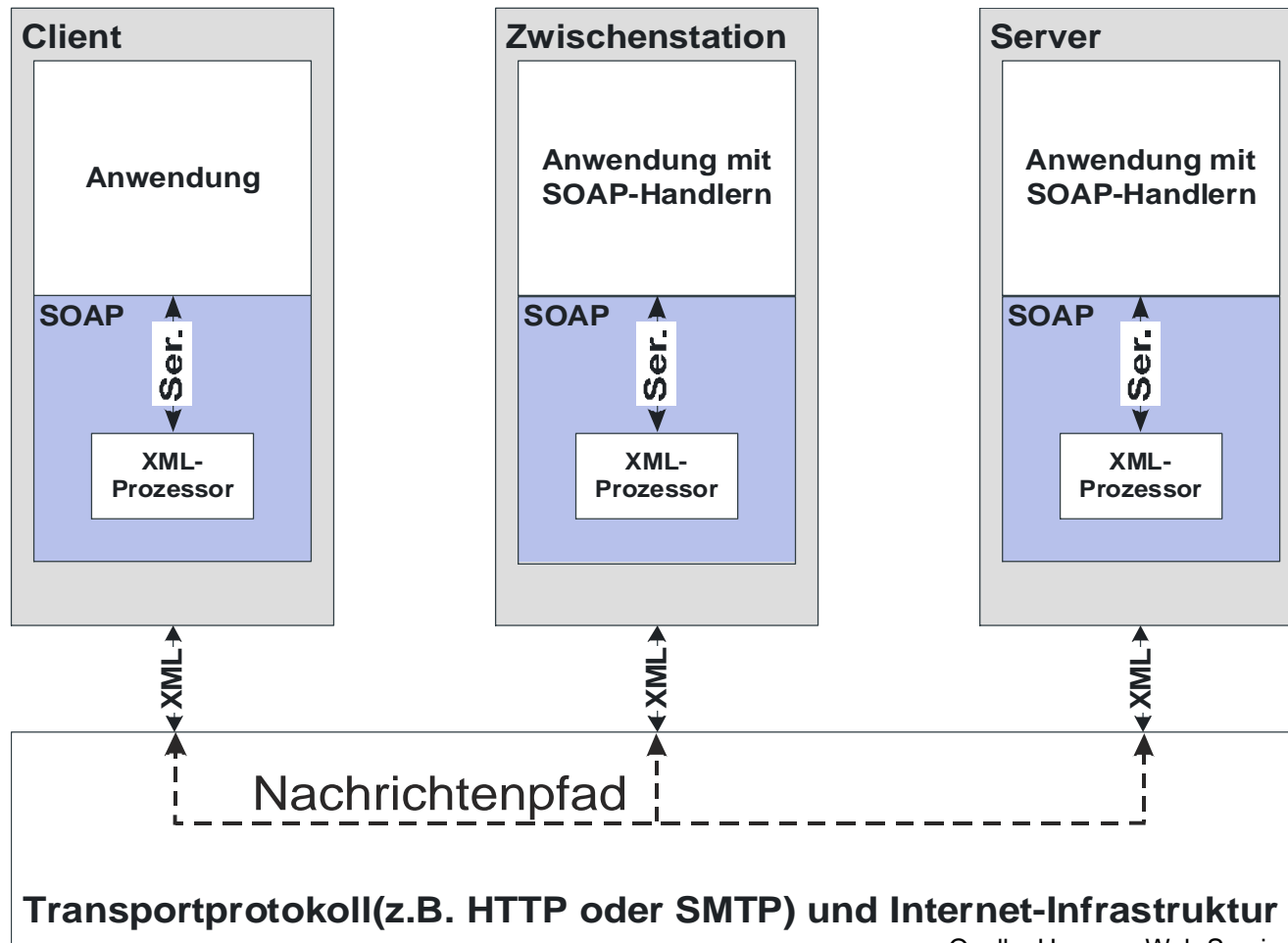
SOAP Überblick I

- Simple Object Access Protocol
- Standardisiertes durch W3C, aktuelle Version: 1.2
- Einfaches, XML-basiertes Protokoll zum Nachrichtenaustausch
- Nicht verbindungsorientiert
 - Frage-Antwortmodell mit einem Netzwerkumlauf
- I.d.R. verwendet für RPC-artigen Zugriff (hier betrachtet)
 - Synchrone Kommunikation – XML-Strukturierung vorgegeben
- Asynchrone Kommunikation möglich
 - Kein RPC sondern „Messaging“
 - Beliebiges XML Dokument als SOAP-Inhalt
 - Behandlung der Nachricht liegt in der Verantwortung der Server
 - Es kann „irgendwann“ eine oder auch keine Server-Antwort erfolgen:
Es werden nur Info-Nachrichten zum Server gesendet

SOAP Überblick II

- Bindung an Standard Internet-Protokolle wie HTTP, SMTP, FTP
- Wichtigstes Ziel: Interoperabilität verschiedenster Informationssysteme
- Derzeit noch keine direkte Unterstützung von Sicherheitskonzepten oder Transaktionen (WS Security in der nächsten Einheit)
- Größerer Aufwand für Aufbereitung und Transport der Daten
 - „aufwändiges“ XML-Format (höhere Datenraten)
 - Einsatz von XML-Parsern an den Endpunkten (hoher Processing Overhead)
- Keine Aussage, wie SOAP verarbeitet werden soll (welche XML-Parser etc.)
- Anwendungen sollen nicht mitbekommen, dass ihre Nachrichten über SOAP transportiert werden (Transparenz, Einsatz von Übersetzern an den Endpunkten)

SOAP Nachrichtenübertragung



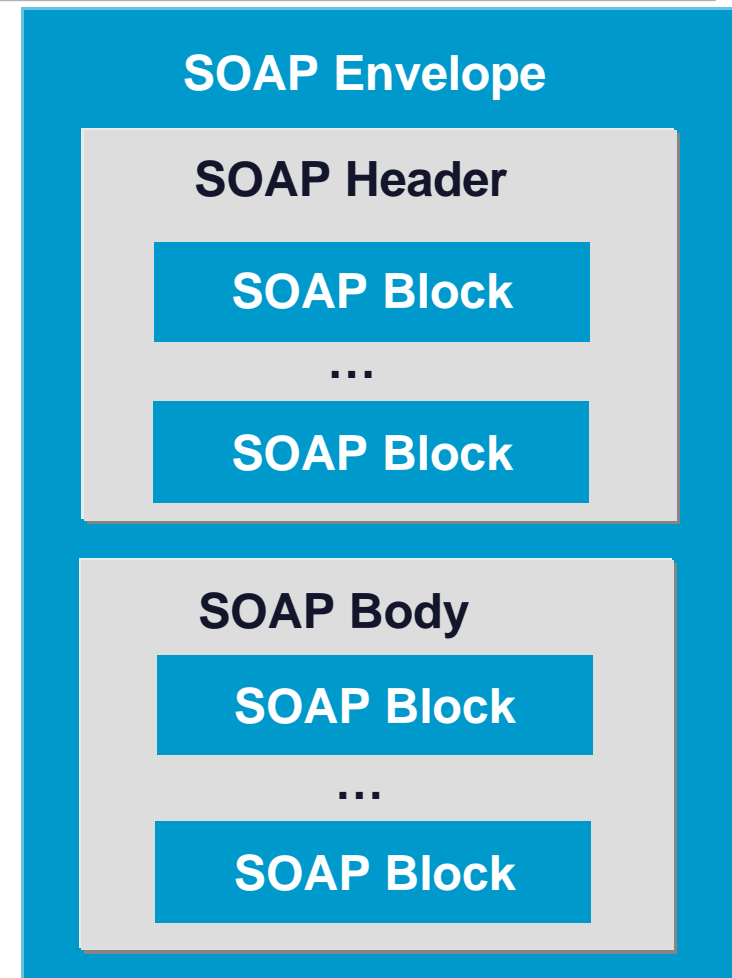
Quelle: Huemer: Web Services

SOAP Nachrichtenübertragung

- SOAP Knoten sind für die Bearbeitungen der Nachrichten zuständig
 - Initial SOAP sender (Client)
 - SOAP intermediaries
 - Ultimate SOAP receiver (Web Service)
- Nachrichtenpfad
 - Reihung der Knoten über die eine SOAP Nachricht übertragen wird
 - Jeder Knoten verarbeitet die Header Blöcke die seiner Rolle entsprechen und reicht die Nachricht weiter
- (Finaler) Empfängerknoten (Server) bearbeitet Body und unadressierte Header Blöcke

SOAP Spezifikation

- SOAP v. 1.2, W3C Recommendation (06/03):
- SOAP Messaging →
 - Struktur einer SOAP Nachricht
- SOAP Encoding
 - Deserialisierungsregeln (Encodierung von Datentypen)
- SOAP Protocol Bindings
 - Wie soll Binding an ein bestimmtes Transport Protokoll (z. B. HTTP) erfolgen



SOAP Request

```
POST /cswservice/cswWS HTTP/1.1
Host: mysoapserver
Content-Type: text/xml; charset=utf-8
...
```

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV=„http://www.w3.org/2003/05/soap-envelope/“>
```

```
<SOAP-ENV:Header>
  <t:TransactionCode xmlns:t=„my-URI“ xsi:type=„xsd:int“
    SOAP-ENV:mustUnderstand=„1“
    SOAP-ENV:role=„http://einserver/actor/TPMonitor “>
    156533245
  </t:TransactionCode>
</SOAP-ENV:Header>
```

```
<SOAP-ENV:Body>
  <method:getNextCoord xmlns:method=„my-URI2“>
    <method:xCoord>15</method:xCoord>
    <method:yCoord>124</method:yCoord>
  </method:getNextCoord>
</SOAP-ENV:Body>
```

```
</SOAP-ENV:Envelope>
```

HTTP Header

Namespace mit SOAP-Vokabular, definiert auch die SOAP Version

Header mit Zusatzinfo; z. B. Transaktionscode

Body enthält die Nutzdaten, in diesem Fall Parameter für RPC

Keine Angaben zur Deserialisierung (später)

SOAP Header

- Headerblöcke enthalten Daten für die Bearbeitung auf SOAP Knoten, z. B.
 - Authentifizierung des Nutzers
 - Transaktionsmanagement
 - Zahlungsinformation
- Das *must understand* Attribut legt fest ob der SOAP Knoten die Nachricht bearbeiten muss
- Header sind durch „*role*“-Attribut genau identifiziert und einer Rolle zugeordnet.
 - Z.B. URI für Transaktionsmanagement -> ein SOAP Knoten für Transaktionskontrolle verarbeitet Header Block
- Fehlt das „*role*“-Attribut wird der Headerblock dem Empfänger der Nachricht zugeordnet.

SOAP Response

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV=„http://www.w3.org/2003/05/soap-envelope“
  SOAP-ENV:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <SOAP-ENV:Body xmlns:m="http://myur">
    <m:getNextCoordResponse>
      <m:coord>123</m:coord>
    </m:getNextCoordResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Standard
Encoding
basierend
auf
XML
Schema

Kodierregeln
für RPC

Fault

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=„...“ >
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <SOAP-ENV:Code>
        <SOAP-ENV:Value>SOAP-ENV:Sender</SOAP-ENV:Value>
        <SOAP-ENV:Subcode>
          <SOAP-ENV:Value>rpc:BadArguments</SOAP-ENV:Value>
        </SOAP-ENV:Subcode>
      </SOAP-ENV:Code>
      <SOAP-ENV:Reason>
        <SOAP-ENV:Text xml:lang="en-US">Processing error</SOAP-ENV:Text>
        <SOAP-ENV:Text xml:lang="cs">Chyba zpracování</SOAP-ENV:Text>
      </SOAP-ENV:Reason>
      <SOAP-ENV:Detail>
        <e:myFaultDetails xmlns:e="http://myurl">
          <e:message>Argument of wrong type</e:message>
          <e:errorCode>999</e:errorCode>
        </e:myFaultDetails>
      </SOAP-ENV:Detail>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Im <body>:
<Fault> mit mindestens
den Unterelementen
<Code>, <Reason>

Subcode ist optional

Detail ist optional

Fehler bei RPC-
Ausführung;
Ebenfalls HTTP-
Errorcodes

SOAP Error-Codes

- **Sender**
 - Fehler liegt auf Client (Initial: Sender) Seite
- **Receiver**
 - Fehler liegt auf Empfänger (Initial: Server) Seite
(z.B. kann ein benötigter Server-Dienst nicht gestartet werden, etc.)
- **VersionMismatch**
 - Ein Knoten auf dem SOAP-Nachrichtenpfad erwartet andere SOAP-Version
- **MustUnderstand**
 - Knoten kann Pflicht-Header-Eintrag nicht auswerten
- **DataEncodingUnknown**
 - Beim Auftreten von Datentypen, die nicht in eine SOAP-Nachricht übersetzt werden können

- **Verpflichtend:**

```
<SOAP-ENV:Fault> ...  
  <SOAP-ENV:Code>...  
    <SOAP-ENV:Value>...  
  <SOAP-ENV:Reason>...
```

- **Optional:** Subcode, Detail

SOAP Datentypen

- encodingStyle attribute identifies encoding rules (Deserialization)
 - commonly located on the Envelope
 - encodingStyle="": Implementation will have to figure out how to deserialize data
- SOAP Encoding (Standard, recommended)
 - encodingStyle="http://www.w3.org/2001/12/soap-encoding"
 - Nutzt weitgehend XML Schema (it borrows), mit einigen Einschränkungen
 - Alle (44) atomare Typen der XML Schema Specification: like: string, integer, float, ...
 - Zusammengesetzte Datentypen struct und array (auch mehrdimensional oder verschachtelt)
 - Folgende Ausdrücke definieren einen identischen Typ (xsi:type from http://www.w3.org/1999/XMLSchema-instance to define a type):

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"  
  soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"  
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"  
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"  
  xmlns:xsd="http://www.w3.org/1999/XMLSchema"> ...
```

```
<vorname xsi:type="xsd:string">Hulk</vorname>  
=  
<vorname xsi:type="SOAP-ENC:string">Hulk</vorname>
```

SOAP Datentypen (2)

- If the application knows what type is being sent, the xsi:type attribute is not required
 - E.g. using WSDL (later)

```
<vorname>Hulk</vorname>
=
<vorname xsi:type="SOAP-ENC:string">Hulk</vorname>
```

- Bei Arrays: Unterschied zwischen SOAP Encoding und XML Schema
 - Elemente müssen nicht vom gleichen Typ sein, Array kann beliebig lang sein

```
<mix xsi:type="SOAP-ENC:Array" >
  <e xsi:type="xsd:string">John Elway</e>
  <e xsi:type="xsd:integer">7</e> ,,,
```

- Beliebige Typen, beschränkte Länge:
 - Attribut **SOAP-ENC:arrayType="SOAP-ENC:ur-type[4]"**
- Bestimmte Typen, beschränkte Länge:
 - Attribut **SOAP-ENC:arrayType="xsd:integer[5]"**>

SOAP Protocol Binding

- Umsetzung der SOAP Konzepte auf darunter liegendes Transport Protokoll (HTTP, SMTP, FTP, ...)
- Identifikation des Nachrichtenempfängers wird dem Transport Protokoll überlassen
- HTTP
 - Eventuell unter Host-Headerfield-Rückgriff Identifikation des Zielservers (Pfadauflösung & Dispatching evtl. durch Web-Server, vlg. Java-Servlets)
 - Weit verbreitet, überall zugänglich (Firewall Toleranz)
 - Einfach umsetzbar
 - Sicherheitskonzepte lassen sich übernehmen (HTTPS)

Protocol Binding zu HTTP

- SOAP request und response werden auf HTTP request/response umgelegt
- SOAP „intermediaries“ sind nicht direkt umzulegen
- Adressierung des Endknotens durch URL
- Über **HTTP Header Feld SOAPAction** kann ein URI übergeben werden, der den Zweck des Requests spezifiziert
 - Kann sukzessive auf „intermediary“ URIs gesetzt werden
- Erfolg einer SOAP Anfrage wird über HTTP Status Code angezeigt
- Fehlercode 500: „internal server error“ deutet Fehler an; Fehlerinformation ist im Body, in einem Fehler Block zu finden

HTTP Binding

```
POST /Reservations HTTP/1.1
Host: travelcompany.example.org
SOAPAction=„Intermediary-URI“
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
>
  <env:Header>
    <t:transaction
...

```

HTTPS: Sicheres Umfeld für SOAP Nachrichten

SMTP Protocol Binding

- SMTP für asynchrone Kommunikation (z. B. in B2B-Anwendungen)

```
From: reservations@travelcompany.example.org
To: a.oyvind@mycompany.example.com
Subject: Which NY airport?
Date: Thu, 29 Nov 2001 13:35:11 EST
Message-Id: <200109251753.NAA10655@travelcompany.example.org>
In-reply-to:<EE492E16A090090276D208424960C0C@mycompany.example.com>
Content-Type: application/soap+xml
```

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
      env:mustUnderstand="true">
```

SOAP für RPCs

- RPC als häufigstes Einsatzgebiet von SOAP
- RPC Daten werden im SOAP Body übertragen
 - URI des Empfängerknotens (bzw. im HTTP-Header)
 - Funktions-/Methodenname
 - Funktions-/Methodensignatur
 - Funktionsparameter
- RPC Request
 - Struktur mit dem Namen der Funktion
 - Funktionsparameter müssen typisiert und benannt werden
 - Reihenfolge der Parameter entspricht der Schnittstelle
 - Ein- sowie Ein/Ausgabe Parameter sind möglich

SOAP-RPC Beispiel

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Body >
    <tmp:getTemp xmlns:tmp="http://csw.inf.fu-berlin.de/temp" >
      <tmp:plz>4040</tmp:plz>
      <tmp:strasse>Altenbergerstrasse</tmp:strasse>
      <tmp:hsnr>74</tmp:hsnr>
    </tmp:getTemp>
  </env:Body></env:Envelope>
```

Request

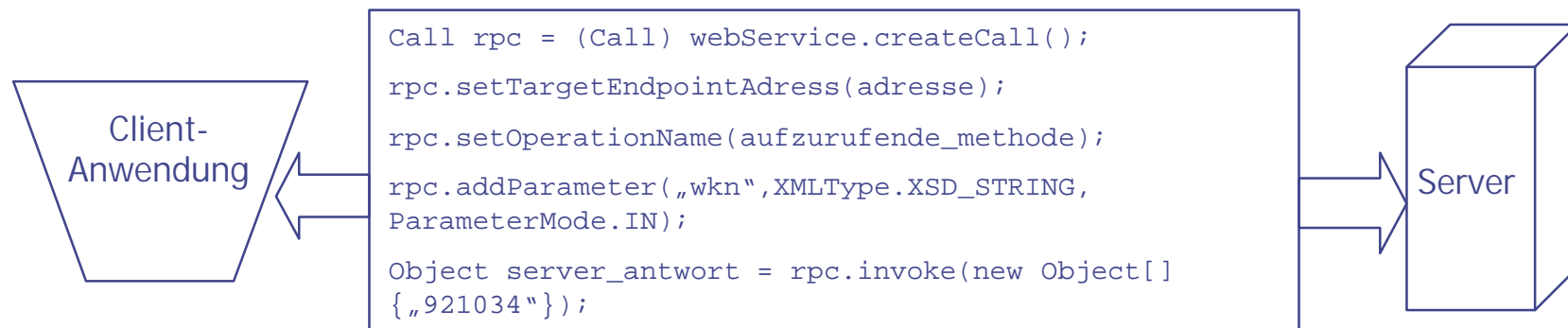
Die Reihenfolge der Parameter entspricht der Schnittstelle

Response

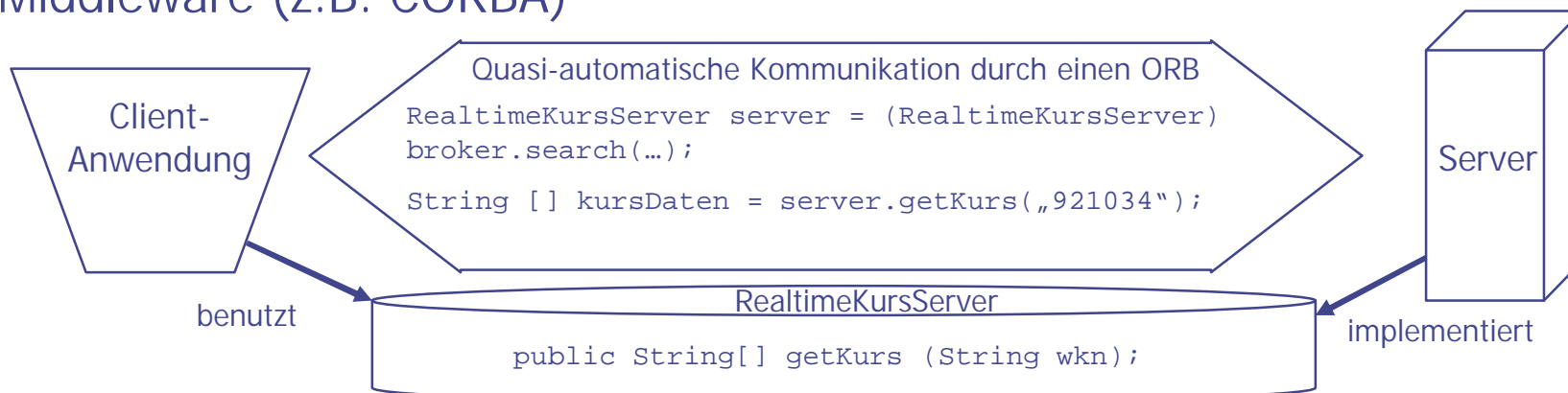
```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Body >
    <tmp:getTempResponse>xmlns:tmp="http://csw.inf.fu-berlin.de/temp">
      <tmp:datumZeit>10.12.2001</tmp:datumZeit>
      <tmp:tempCelsius>-3</tmp:tempCelsius>
    </tmp:getTempResponse>
  </env:Body>
</env:Envelope>
```

SOAP/RPC vs. Classic Middleware

SOAP/RPC



Middleware (z.B. CORBA)



SOAP Ausblick

- SOAP Einsatz nicht nur als RPC-Protokoll, sondern für Nachrichtenaustausch allgemein
- Bemühungen des W3C für SOAP-Nachfolger XML Protocol (XMLP)
 - Komplette Architektur für verteilte Anwendungen
 - Bessere Unterstützung von Zwischenstationen
 - Integration von Mechanismen für Sicherheit und Privatsphäre (W3C P3P)

WSDL

- Für den Einsatz von SOAP muss man Parameter, Datentypen, Methodennamen und die Adresse eines Web Services kennen
- Beschreibung eines WS durch die Web Service Description Language
 - Welche Parameter erwartet ein Web Service?
 - Wo liegt der Web Service genau ?
 - Welche Schnittstellen bietet der Service?
 - Mit welchen Protokollen kann man den Web-Service einbinden?
- Kann als Verweis eines UDDI-Verzeichniseintrags abgerufen werden, direkt in ein UDDI-Eintrag eingebettet sein, oder vor Dienstnutzung z.B. per Email ausgetauscht werden.
- Betrachtet WS auf zwei Ebenen
 - Funktional = abstract: Welche Funktionalität bietet der WS, welche Schnittstellen (Parameterliste)
 - Technisch = konkret: Ort sowie Art und Weise, wie ein WS aufgerufen wird (Adresse? HTTP?)
- WSDL Version 1.1 (W3C RFC) derzeit „de facto“ Standard zur Beschreibung von WS
 - Version 2.0 gerade in der Verabschiedung als Standard (Hier betrachtet: Version 1.1.)
- Einsatz von WSDL zur Generierung von Proxies für Web-Services
 - Vergleichbar mit den IDL-Dateien von CORBA
 - „Pseudeolokaler“ Zugriff auf Web-Service Methoden

Inhalte einer WSDL Datei

- Welche Datentypen werden übermittelt?
 - <types/> Element
- Welche Nachrichten werden übermittelt?
 - <message/> Element
- Welche Operationen werden unterstützt?
 - <portType/> Element

abstract

- Wie werden die Nachrichten im Netz transportiert?
 - <binding/> Element
- Wo befindet sich der Dienst?
 - <service/> Element

concrete

Struktur einer WSDL Datei (Komponenten)

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<definitions ...>
```

```
<types/>
```

Types

```
<message/> <portType/>
```

Messages/portTypes

```
<binding/>
```

Bindings

Service Interface

```
<service/>
```

Service Definition

Service Implementation

```
</definitions>
```

WSDL Dokument



Definition „eigener“ Datentypen eines Dienstes

```
<?xml version="1.0"?>
```

```
<definitions name="StockQuote"
```

```
  targetNamespace="http://example.com/stockquote.wsdl"
```

```
  xmlns:tns="http://example.com/stockquote.wsdl"
```

```
  xmlns:xsd1="http://example.com/stockquote.xsd"
```

```
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
```

```
  xmlns="http://schemas.xmlsoap.org/wsdl/">
```

Dropped in WSDL 2.0

```
<definitions>
```

- Wurzel-Element jedes WSDL-Dokuments
- deklariert Namespaces
- Container für die WSDL-Komponenten

```
<types>
```

```
  <schema targetNamespace=http://example.com/stockquote.xsd"
```

```
    xmlns="http://www.w3.org/2000/10/XMLSchema">
```

```
    <element name="xsd1:TradePriceRequest">
```

```
      <complexType> <all> <element name="tickerSymbol" type="string"/>... </all> </complexType>
```

```
    </element>
```

```
    <element name="xsd2:TradePrice">
```

```
      <complexType> <all> <element name="price" type="float"/> ...</all> </complexType>
```

```
    </element>
```

```
  </schema>
```

```
</types>
```

```
...
```

```
<types>
```

- umfasst Element-Datentyp Definitionen
- verwendet XML Schema als sein vorgegebenes Typ-System
- erlaubt aber auch andere (vom W3C nicht empfohlen)

Nachrichten und Operationen eines Dienstes: message, portType

```
<message name="GetLastTradePriceInput">
  <part name="body" element="xsd1:TradePriceRequest"/>
</message>
```

```
<message name="GetLastTradePriceOutput">
  <part name="body" element="xsd1:TradePrice"/>
</message>
```

Element attribute:
Global Element (Single)

```
<message name=„,NotUsedHere">
  <part name=„,when" xsi:type="xsd:date"/>
  <part name=„,nr" xsi:type="xsd:int"/>
</message>
```

XML Schema standard types

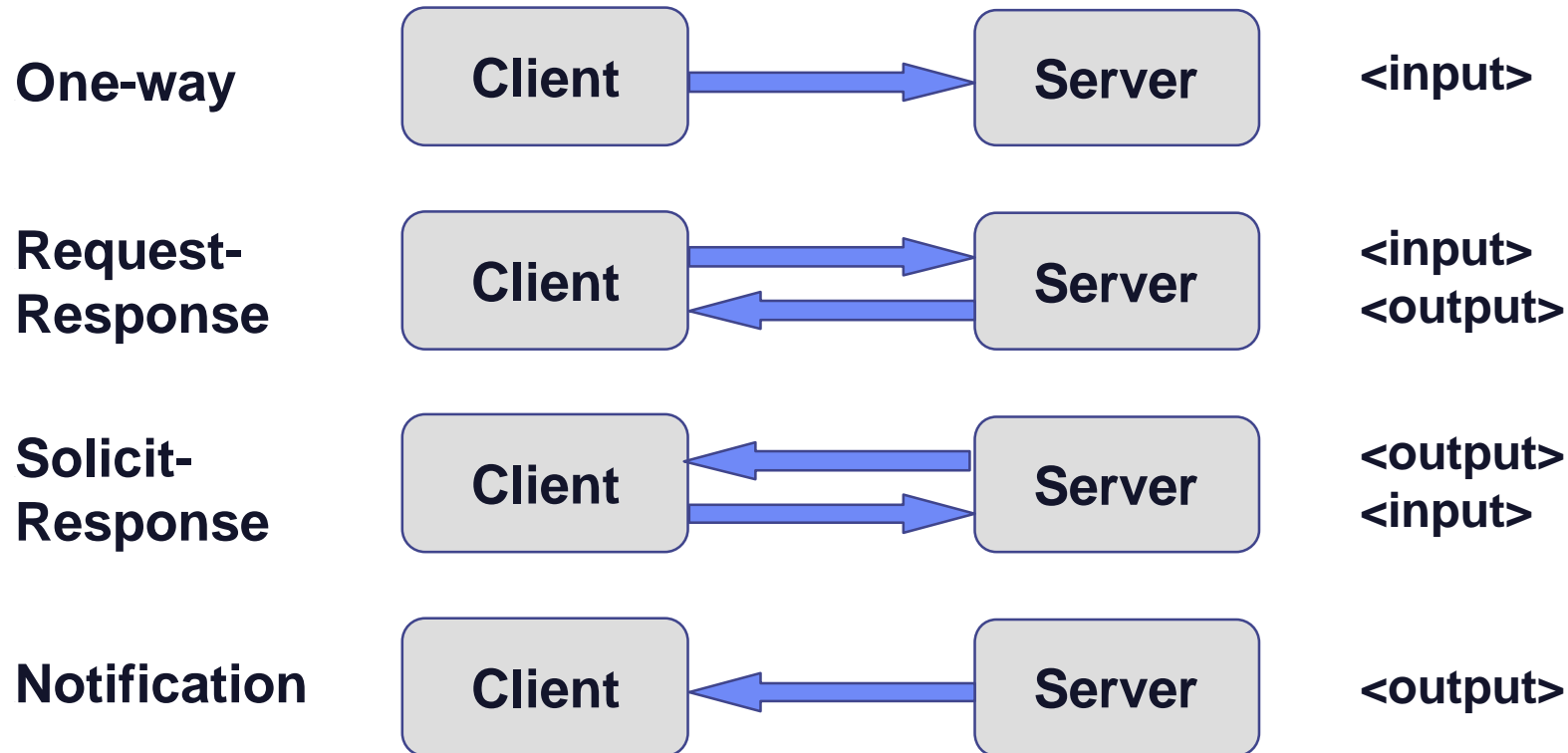
```
<portType name="StockQuotePortType">
  <operation name="GetLastTradePrice">
    <input message="tns:GetLastTradePriceInput"/>
    <output message="tns:GetLastTradePriceOutput"/>
  </operation>
  <operation name=„,XYZ...
</portType>
```

- Abstrakte Operation fasst Nachrichten zusammen.
- Abstrakter PortType fasst Operationen zusammen.
- Input Message: Eingehend, Output Message: Antwort



PortType und Operation

- Grundmodelle von Operationen:



Binding

- Definiert konkrete Nachrichtenformate und Protokoll-Einheiten
- Die meist benutzten Protokolle sind SOAP over HTTP(s) POST.
- Ein Binding muss exakt ein Protokoll spezifizieren.
- Jedes Binding baut auf genau einem Port Type auf.
- Zwei Binding-Styles möglich (Sollten bei RPC den gleichen XML-Code generieren):
 - Document style: All message parts are directly inserted into the SOAP envelope as children of <soap:body>; Content of <soap:Body> is specified by XML Schema defined in the <wsdl:type> section.
 - RPC style: The structure of an RPC style <soap:Body> element needs to comply with the rules specified in the SOAP 1.1 specification. According to these rules, <soap:Body> may contain only one element that is named after the operation, and all parameters must be represented as sub-elements of this wrapper element.
- Wird meist von SOAP-Engines ignoriert; in WSDL 2.0 fällt Unterscheidung weg.

Wie werden Nachrichten transportiert?

```
<binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetLastTradePrice">
    <input> <soap:body use="literal"/> </input>
    <output> <soap:body use="literal"/> </output>
  </operation>
</binding>
```

- Binding attributes
 - name: Referenziert in <service> Elementen (nächste Folie)
 - type: Referenziert <portType>
 - Soap:binding: Über welches Protokoll wird auf einen Dienst zugegriffen
- Unterelemente <operation> weist jeder Methode des portType (des Web Service) eine Codierung zu
- use Attribut gibt Encoding Regeln der SOAP Nachricht an
- use="literal": Type definitions literally follow an XML schema definition.
 - Recommended; the only allowed "use" value in future WSDL Versions
 - use="encoded": The rules to encode and interpret a SOAP body are in a URL.

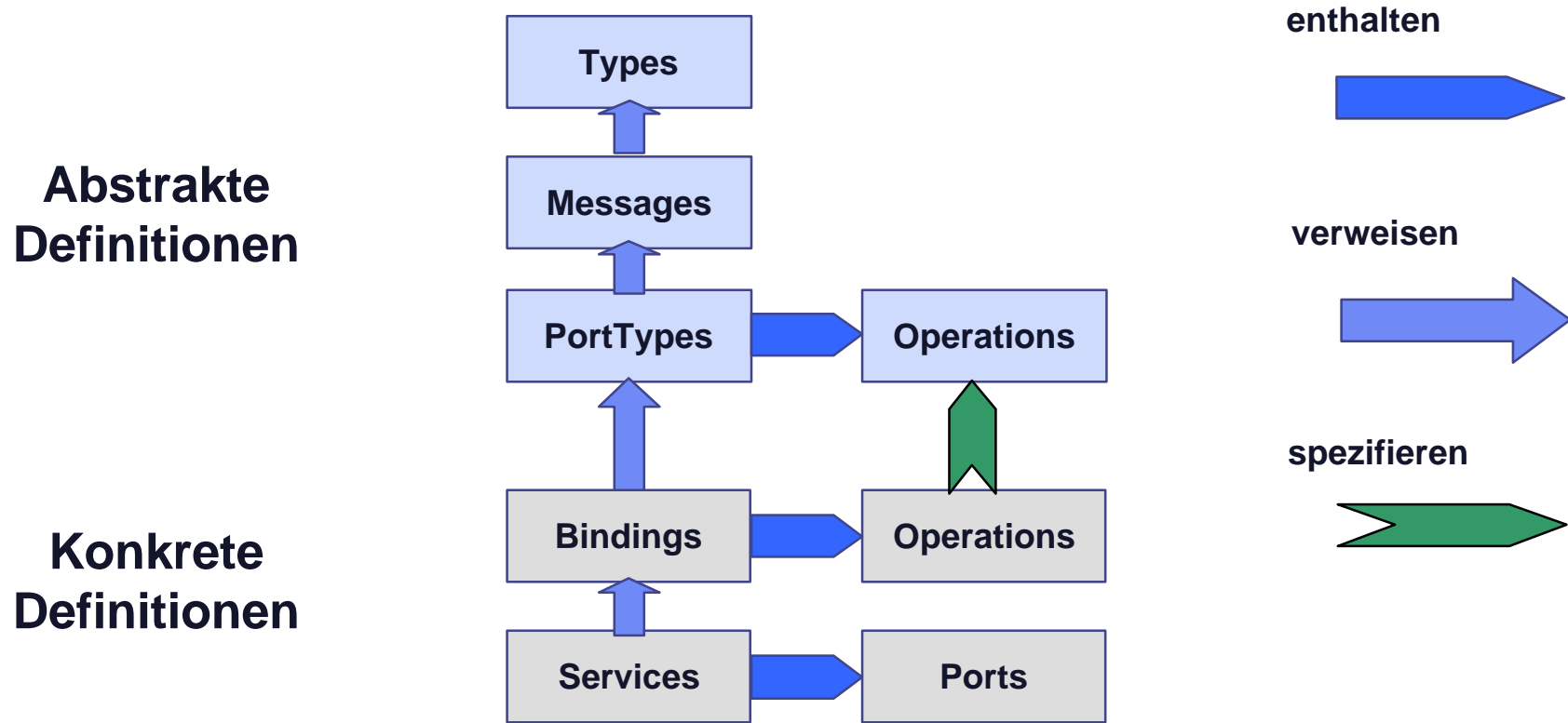
Wo befindet sich der Dienst?

```
<service name="StockQuoteService">
  <documentation>My first service</documentation>
  <port name="StockQuotePort" binding="tns:StockQuoteSoapBinding">
    <soap:address location="http://example.com/stockquote"/>
  </port>
</service>

</definitions>
```

- Jeder **Port** spezifiziert die Adresse einer einzelnen Bindung.
- **Service** ist eine Sammlung von Ports.

WSDL- Dokumentstruktur



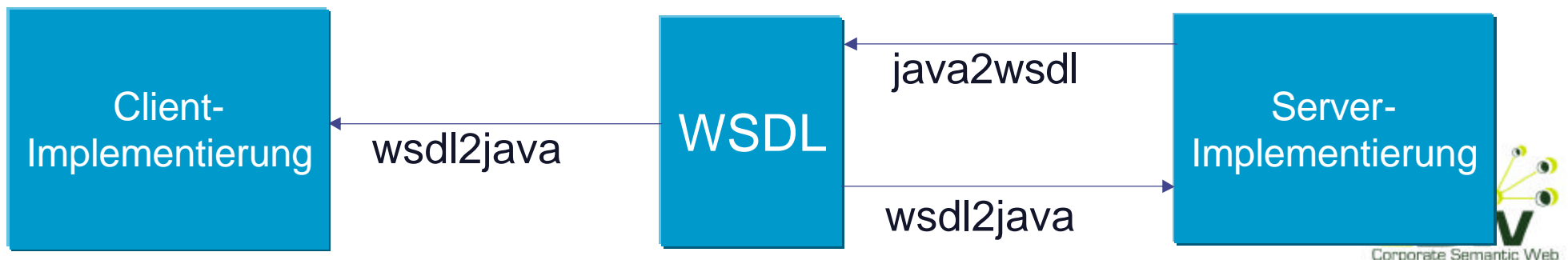
Eine oder mehrere Adressen eines WS, über die angebotene Operationen angesprochen werden können

Ausflug: Werkzeug: Axis

- Open Source SOAP-Engine in Java von Apache
 - <http://ws.apache.org/>
- Axis-Servlet agiert als Transport-Listener
 - nimmt HTTP Requests entgegen
 - Extrahiert SOAP-Nachricht
 - Übergibt Nachricht der Axis Engine
 - Axis Engine ruft „native“ Java-Methode auf (Operations)
 - Erhält return Value der Methode, codiert diese in SOAP-Nachricht
 - Schickt die Response an den Client zurück
- Zusätzliche Werkzeuge für die Nutzung und Bearbeitung von WSDL
 - WsdI2java für Clients und Server
 - Java2wsdl

Generieren von WSDL

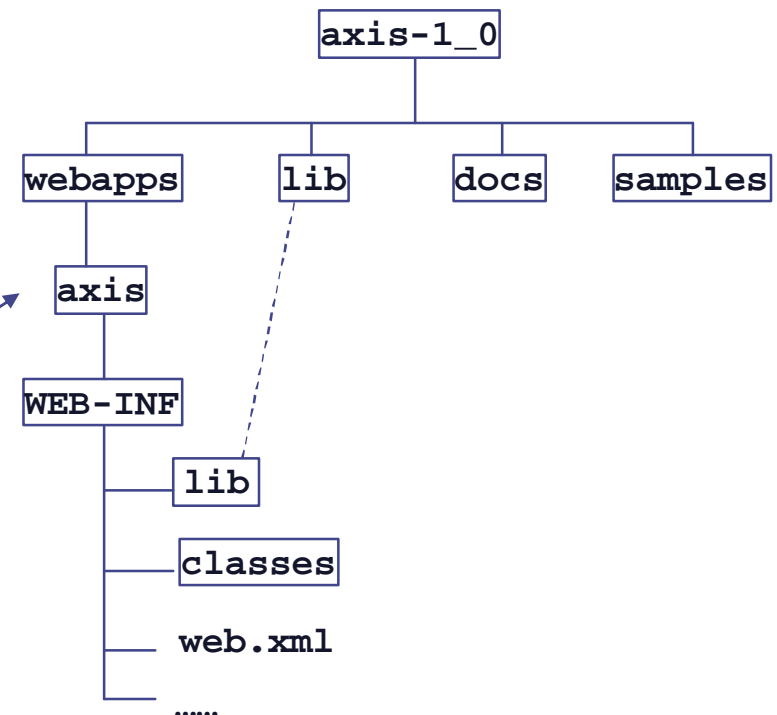
- Die Abbildung von WSDL auf eine Programmiersprache und umgekehrt ist möglich
- Apache AXIS bietet folgende Möglichkeiten:
 - Generierung von WSDL aus bestehender Webservice Implementierung
 - Gewöhnliche Java-Klasse mit public-Methoden
 - Clientseitige Implementierung aus WSDL (Proxy + Gerüst des Clients)
 - Serverseitige Implementierung aus WSDL (Proxy und Gerüst der Dienst-Klasse)



Apache Axis

- Man benötigt
 - J2SE SDK 1.3 oder 1.4
 - Servlet Container (z. B. Tomcat)
- Download (Zip-File) unter <http://xml.apache.org/axis>
- Axis läuft als Servlet.
- Deployment von Axis.
 - Kopieren in Tomcat webapps Verzeichnis
 - Tomcat starten


Verzeichnisstruktur:



UDDI

- Universal Description, Discovery and Integration
 - Globaler, branchenspezifischer oder intranetweiter Verzeichnisdienst
 - Ermöglicht Eintragen und Suchen nach Web Services
 - Spezifikation durch OASIS, aktuelle Version 3.0
- Große, öffentliche UDDI-Server bei HP, IBM, Microsoft, SAP, ...
 - Z.B. UDDI Business Registry (über Nacht gespiegelt)
- Standardisierter Zugriff auf UDDI Directory über SOAP
 - Publisher-API
 - Inquiry API
 - Daneben: Zugriff „für Menschen“ über Web-Dialog-Interface
- Alternative: WS-Inspection (Web Services Inspection Language)
 - Wesentlich simpler
 - Basiert auf vielen kleinen Verzeichnissen mit wenigen Einträgen (im Basisverzeichnis des jeweiligen WebServers)
 - ... diesen muss man also zuvor kennen! !
 - Hier nicht betrachtet

B2B Nutzungsszenario

1.  Software-Firmen, und Standardisierungsgremien füllen das Verzeichnis mit der Beschreibung verschiedener Arten von Diensten

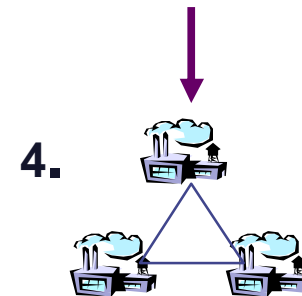
UDDI Business Registry (UBR)
öffentlich, von IBM, SAP, MS, ...



2. Registry weist jedem Dienst- und Geschäftseintrag eine eindeutigen ID zu



UDDI Business Registry
Marktplätze, Suchmaschinen und Geschäftsanwendungen fragen beim Verzeichnis an, um Dienste anderer Firmen zu finden



Unternehmen nutzen die Daten zur Integration mit anderen Organisationen

Semantische Teilung

White
Pages

Adresse, Kontaktinformationen,
Ansprechpartner

Yellow
Pages

Einordnung des Unternehmens
gemäß Geschäftsbereich (business type)

Green
Pages

Technische Informationen zu angebotenen
Dienstleistungen und Verweise zu
genauen Spezifikationen
(WSDL Service Interface)

UDDI Komponenten

- UDDI XML Schema
 - Definiert UDDI Datenmodell
 - 5 Datenstrukturen
 - businessEntity
 - businessService
 - bindingTemplate
 - tModel
 - (publisherAssertion)
- UDDI APIs
 - Dient zur Interaktion zwischen UDDI Client und UDDI Registry Server
 - Eine Reihe von Schnittstellen (SOAP, also Austausch von XML)
 - Zum veröffentlichen, verwalten und suchen von Verzeichnisdaten

UDDI Datenstruktur

businessEntity

+businessKey +name +contacts +description +businessServices
+identifierBag +categoryBag ...

businessService

+serviceKey +businessKey +name +description
+bindingTemplates +categoryBag ...

bindingTemplate

+bindingKey
+serviceKey
+accessPoint
+hostingRedirector
Description
+tModelInstanceDetails

tModel

+tModelKey
+name
operator
categoryBag
overviewDoc
Description
authorisedName

About categoryBag

- Categorization is very important
- categoryBag contains list of keyedReferences (Keys of tModels)
- Each keyedReference indicates one categorization scheme
 - KeyedReference contains two other parts of information
 - keyName (not Used, for internal purposes)
 - keyValue, particular value for this instance of the category system
 - E.g.: ISO 3166 (Geographical Category System)
 - Recommendation: Name of the tModel = uddi:uddi.org:ubr:categorization:iso3166
 - Oder uddi:uddi.org:ubr:categorization:iso3166 direkt referenzieren (in categoryBag)
 - Organization wants to indicate that it operates in California:
 - keyedReference to the key of the tModel and a keyValue „US-CA“
 - To find an Organization in California:
 - Include the same keyedReference in the search request
 - Search for keyValue „US-CA“, or „US-%“ to find all registered US Organizations

businessEntity

- Fasst die Information über den Dienstanbieter zusammen (Firma, Organisation)
- Notwendige Angaben:
 - businessKey: Eindeutiger Identifier einer businessEntity-Instanz
 - name: Name der registrierten businessEntity-Instanz
- Optional:
 - description: Textuelle Beschreibung der businessEntity
 - contacts: Kontaktdaten der Ansprechpartner für die businessEntity
 - businessServices: Liste von Referenzen auf businessServices (von Diensten), die von der businessEntity angeboten werden
 - identifierBag: Liste von Identifier/Name-Paaren, die die businessEntity eindeutig kennzeichnen (z.B. Steuernummer=XYZ“)
 - Eventuell wieder Verweise auf tModels
 - categoryBag: Liste von keyedReferences auf tmodels, die Name/Kategorie-Paare zur Kategorisierung von businessEntities enthalten; z.B. per Branche gemäß Industrie (NAICS) oder per geographischer Lage (ISO 3166) oder über angebotene Produkte und Services (UNSPSC)
 - Dsig:Signature: Um eine digitalen Signatur zuzuweisen

Beispiel businessService

```
<businessEntity businessKey="35AF7F00-1419-11D6-A0DC-000C0E00ACDD">
  <name>BooksToGo</name>
  <description xml:lang="en"> Our books are nice</description>
  <contacts>
    <contact>
      <personName>Hubert Buch</personName>
      <phone> 11 2342 </phone>
      <email> hubert.busch@bookstogo.com<email>
    </contact>
  </contacts>
  <businessServices> ... </businessServices>
  <categoryBag> ... </categoryBag>
  ...
</businessEntity>
```

businessService

- Repräsentiert einen bestimmten Web Service, der auf unterschiedliche Art und Weise aufgerufen werden kann
- Felder:
 - serviceKey: Eindeutiger Identifier eines businessKey
 - businessKey: Enthält Key der businessEntity, zu der der businessService gehört
 - description: Textuelle Beschreibung der businessService
 - bindingTemplates: Listet die Keys der techn. Beschreibungen (bindingTemplates) der enthaltenen Web Services auf.
 - categoryBag: Liste mit Referenzen auf tModels, die Name/ Kategorie-Paaren zur Kategorisierung des businessService enthalten
 - Dsig:Signature: Um eine digitalen Signatur zuzuweisen

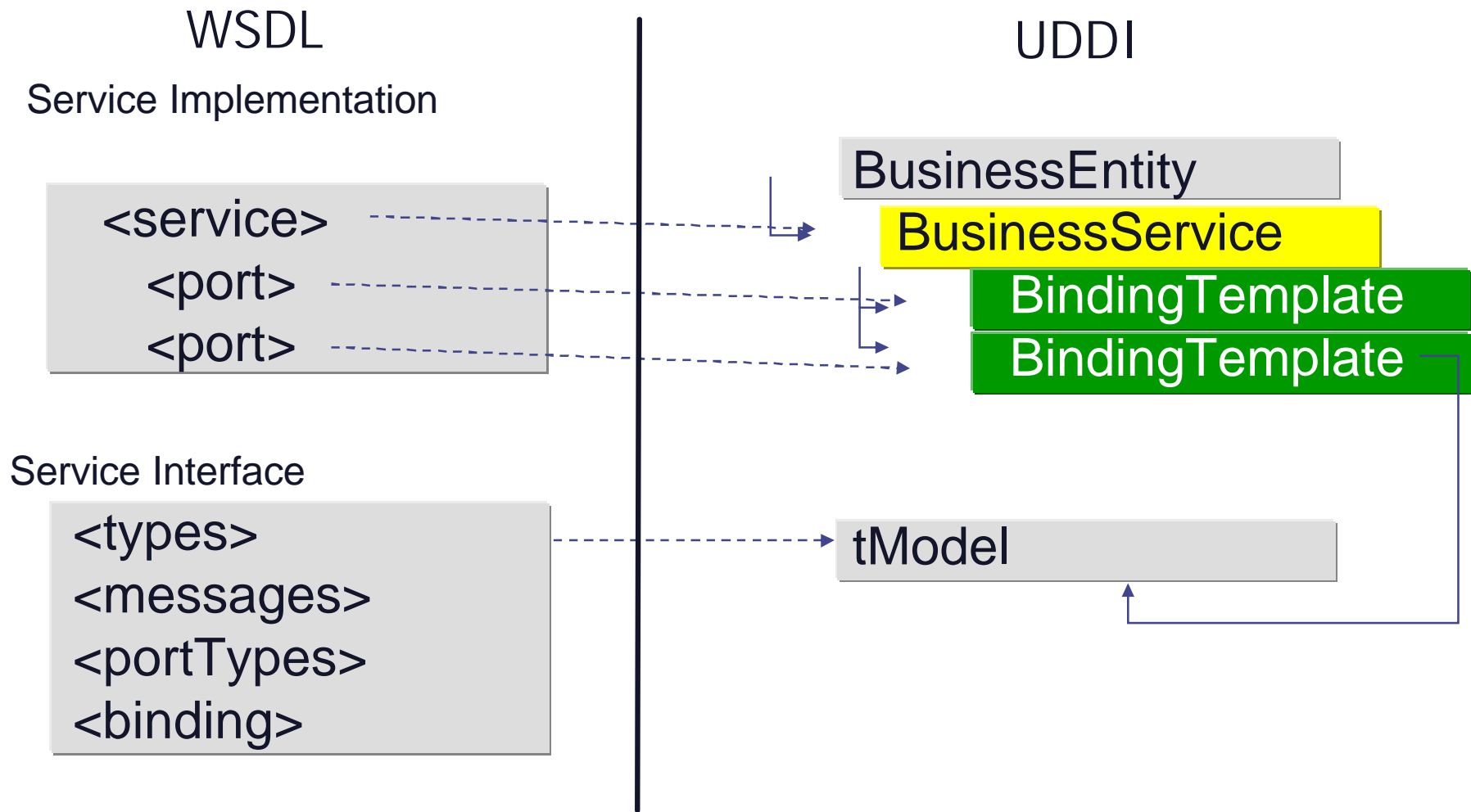
bindingTemplate

- Repräsentiert eine konkrete Implementierung eines Dienstes (enthält die zur Nutzung eines Dienstes notwendigen technischen Infos)
- Zumindest die Adresse, über die der Dienst aufgerufen werden kann (meist URL)
- Weitere Beschreibung des Dienstes erfolgt über tModels, die referenziert werden.
- Felder:
 - bindingKey: Eindeutiger Identifier einer bindingTemplate-Instanz
 - serviceKey: Key des businessService, zu dem das bindingTemplate gehört
 - accessPoint: Spezifiziert Adresse eines Dienstes, meist URL
 - description: Textuelle Beschreibung des bindingTemplate
 - tModelInstanceDetails: Referenziert ein oder mehrere tModels; die Summe der tModels (i.d.R. = Summe angebotener Operationen) bilden die technischen Schnittstellenbeschreibung des Web Service
 - Ursprüngliche Motivation der tModels
 - categoryBag: Liste von Name/Kategorie-Paaren zur Kategorisierung des bindingTemplates (eventuell wieder tModel-Referenzierend)
 - Dsig:Signature: Um eine digitalen Signatur zuzuweisen

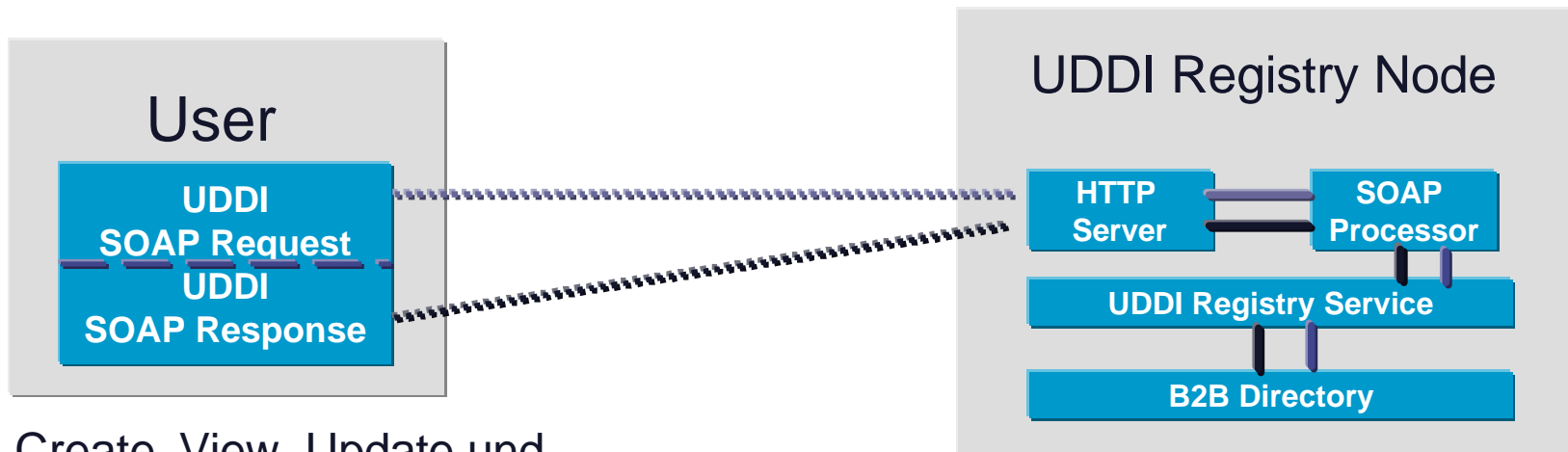
tModel

- Hat mehrere Funktionen
 - Beschreibt Web Service auf technischer Ebene
 - Repräsentiert: Identification Systems, Category Systems, Namespaces
 - Repräsentiert Transportprotokolle wie HTTP
 - Repräsentiert: Formate für Postanschriften etc.
- Enthält folgende Felder:
 - tModelKey: Eindeutiger Identifier einer tModel-Instanz
 - deleted: Gibt an, ob das tModel gelöscht wurde (wird lediglich als gelöscht markiert).
 - name: Qualifizierter Name des tModel – URI! (z.B. auf
 - description: Textuelle Beschreibung der businessService
 - overviewDoc: Verweis auf Dienstbeschreibung, z.B. URL der WSDL-Datei
 - Außerhalb der UDDI-Registry gespeichert
 - ... oder auch Verweise auf Spezifikationen, die relevant zur Dienstnutzung sind (falls tModel in categoryBag auftaucht)
 - categoryBag: Liste von Verweisen auf technische Konzepte, z.B. SOAP-Spezifikationen etc.
 - identifierBag: Angabe weiterer Daten, die das tModel eindeutig identifizieren
 - Dsig:Signature: Um eine digitalen Signatur zuzuweisen

WSDL zu UDDI Mapping, mit Description Splitting, wenn man will...



UDDI Interaktions-Ablauf



Create, View, Update und
Delete von Eintragungen

Inquiry und Publishers API

Inquiry API

Suchen

- **find_business**
- find_service
- find_binding
- find_tModel

Details abfragen

- get_businessDetail
- **get_serviceDetail**
- get_bindingDetail
- get_tModelDetail

Publishers API

Speichern

- save_business
- save_service
- save_binding
- save_tModel

Löschen

- delete_business
- delete_service
- delete_binding
- delete_tModel

Sicherheit

- get_authToken
- discard_authToken

Bsp. UDDI Inquiry API

- Auffinden von Information in UDDI
- Operationen zum durchsuchen (Inhalt des <SOAP:Body> einer Anfrage)
 - **find_business**, find_binding, find_service, find_tModel

```
<find_business generic=„2.0“ xmlns=uddi-org:api-v2„ maxRows=„10“>  
  <name>Buch%</name>  
</find_business>
```

Hier: Simple Regular
Expression Search
with a wildcard

- Liefert Null oder mehr <businessInfo> Elemente. Jedes <businessInfo> Element (strukturiert) enthält unter anderem den zugehörigen businessKey
- Operationen für das Erfragen genauerer Informationen:
 - **get_businessDetail**, get_serviceDetail, get_businessDetailExt, get_tModelDetail, ...

```
<get_businessDetail generic=„2.0“ xmlns=uddi-org:api-v2>  
  <businessKey>860eca90-c16d</businessKey>  
</get_businessDetail>
```

- Liefert ein <businessEntity> Element (strukturiert) mit weiteren Informationen, wie businessService-Keys mit denen dann weitergesucht werden kann
 - get_serviceDetail -> get_bindingDetail -> get_tModelDetail -> ... Overview.doc (WSDL)

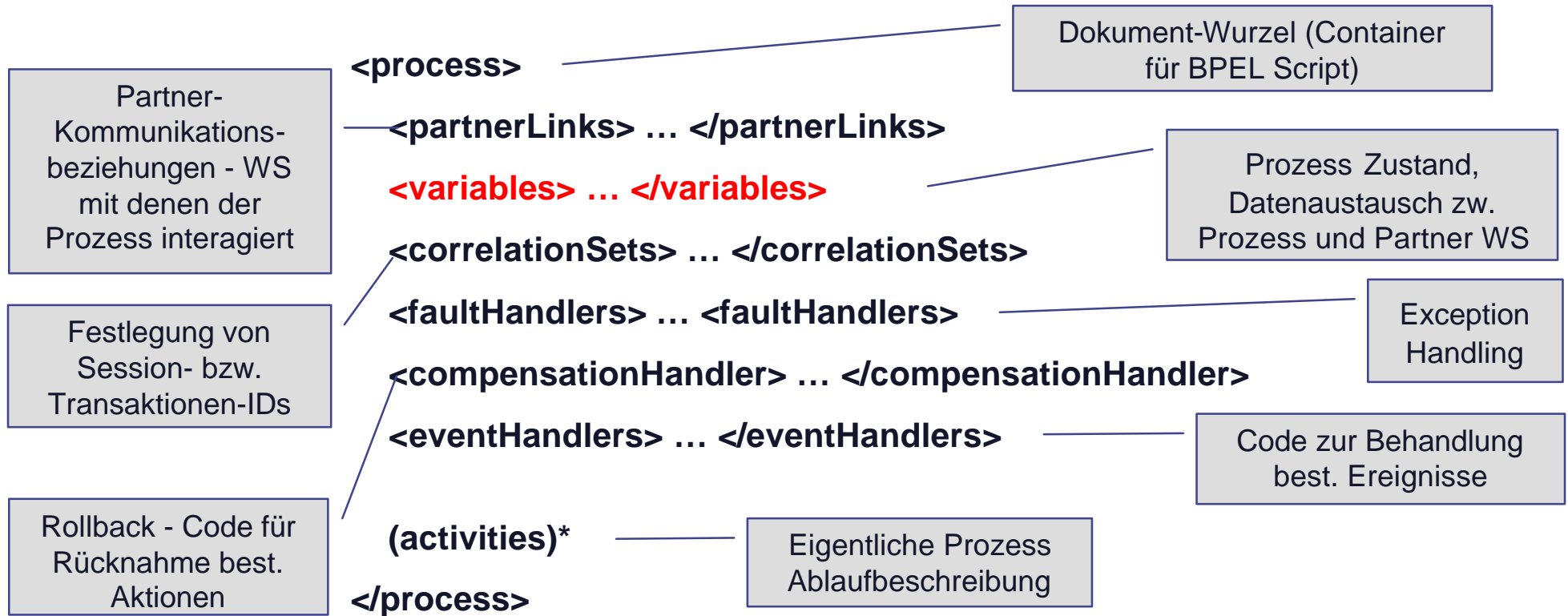
UDDI Fazit

- **Positiv**
 - Einfaches Einbinden von Services fremder Unternehmen
 - Mit WSDL noch bessere Integration
 - Mit Version 3 wurden Sicherheitsaspekte besser berücksichtigt
 - Digitale Signaturen etc.
- **Negativ**
 - Akzeptanz: Bislang wenige registrierte Dienste
 - Fehlende Semantik
 - unterschiedliche Beschreibung gleicher Web Services
 - Keine standardisierten tModels für gleichartige Services

WS-BPEL

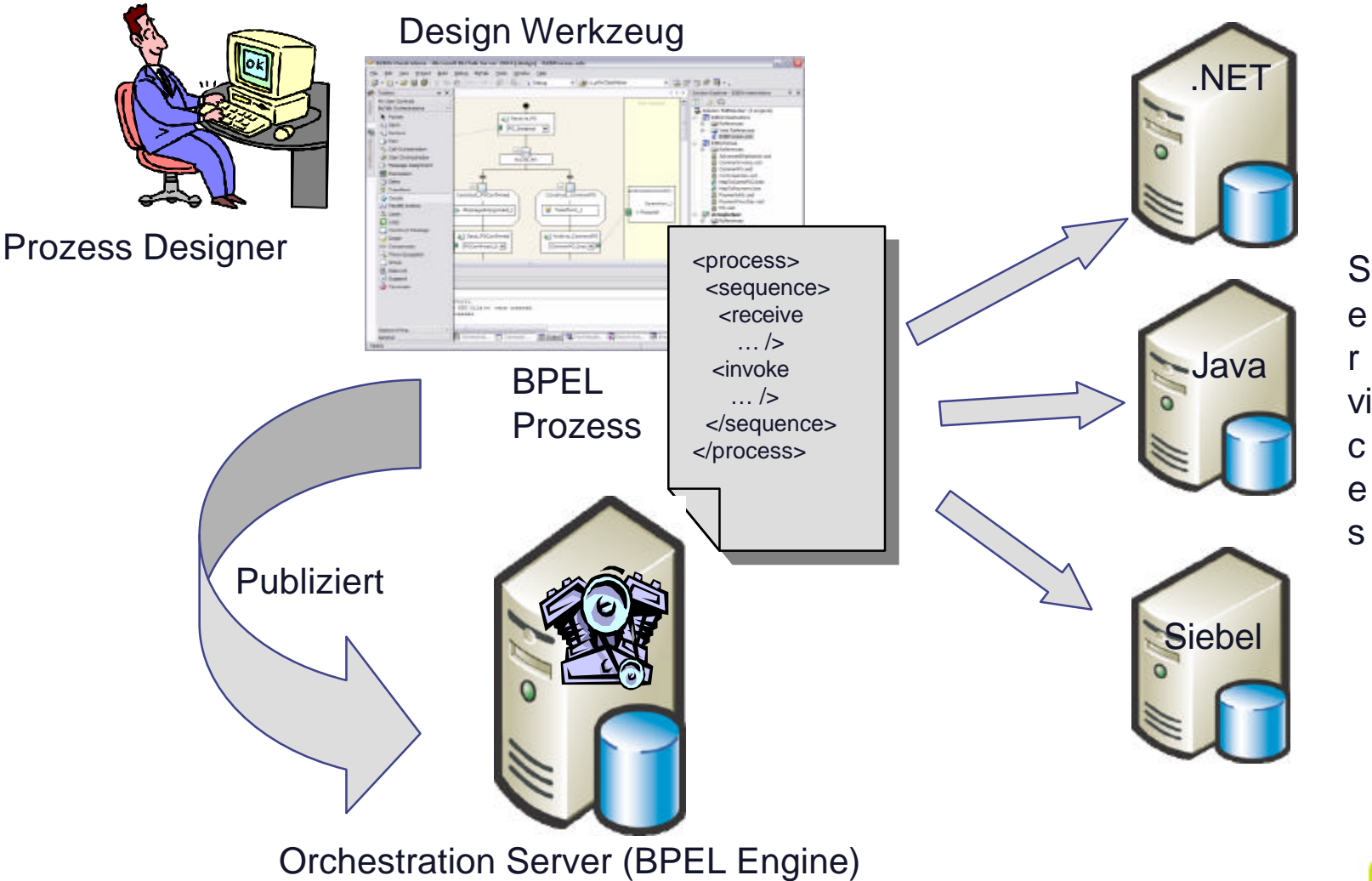
- XML-basierte Sprache zur Spezifikation von Geschäftsprozessen mit mehreren Webservices.
- Hervorgegangen aus Microsofts XLANG (graphenorientiert) und IBMs WS-FL (algebraisch).
- Aktuelle Version: WS-BPEL Version 2.0 – Standardisierungs-Entwurf kurz vor Verabschiedung bei OASIS (Organization for the Advancement of Structured Information Standards).
- Vorgänger BPEL4WS 1.1 (2003) durch BEA, IBM, Microsoft, SAP, Siebel
- Ähnlich höheren imperativen Programmiersprachen (if, while,...).
- BPEL baut auf XML Schema, XPath und WSDL auf.
- Rekursive Komposition: BPEL Prozess tritt selbst als WS auf (WSDL mit PortTypes, Operations, ...)

BPEL Syntax



activities = <receive>, <reply>, <invoke>, <assign>, <throw>, <terminate>, <wait>, <empty>, <sequence>, <switch>, <while>, <pick>, <flow>, <scope>, <compensate>

WS-Orchestration mit BPEL

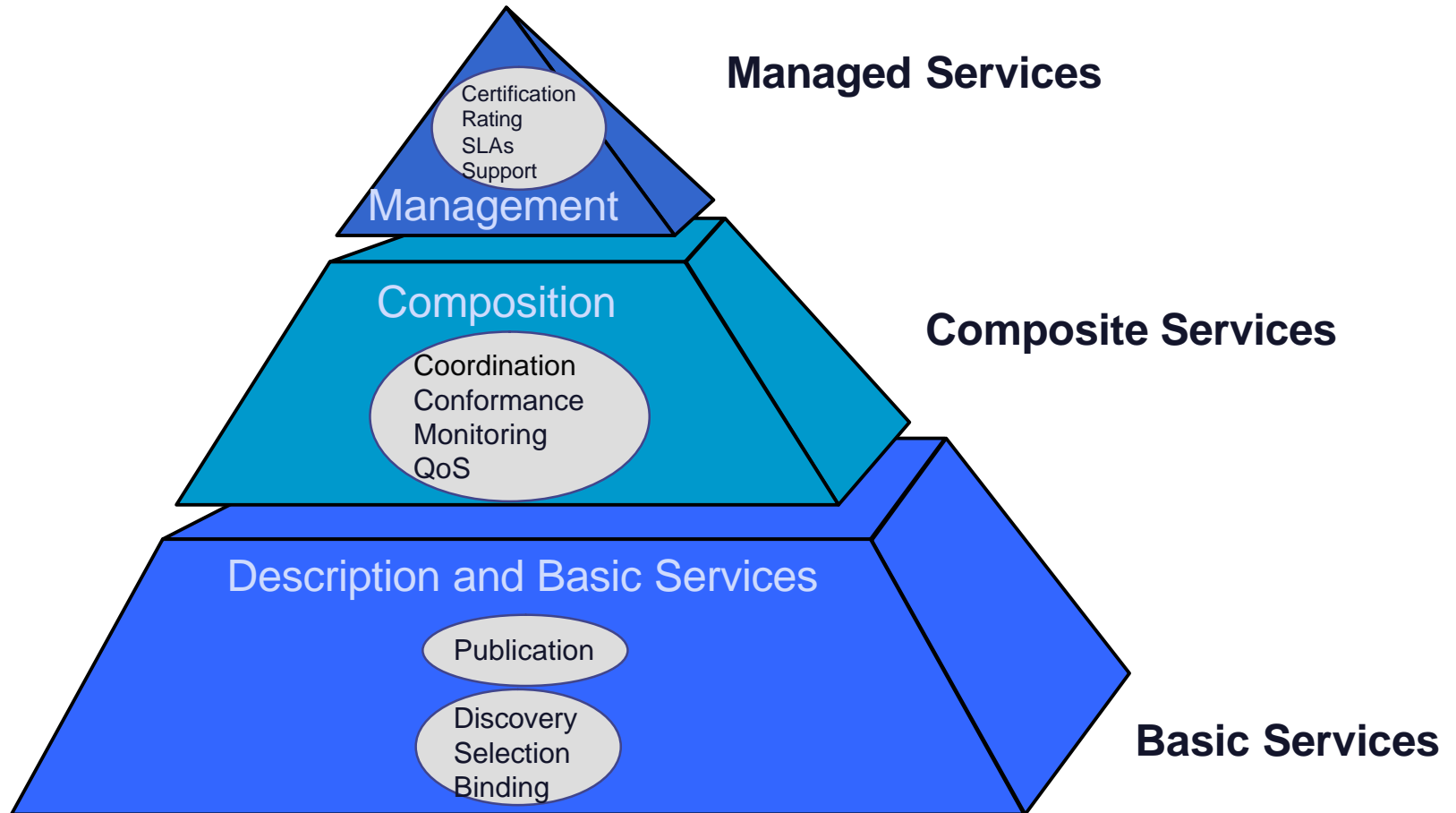


Verteilte Systeme: Web Services vs. CORBA

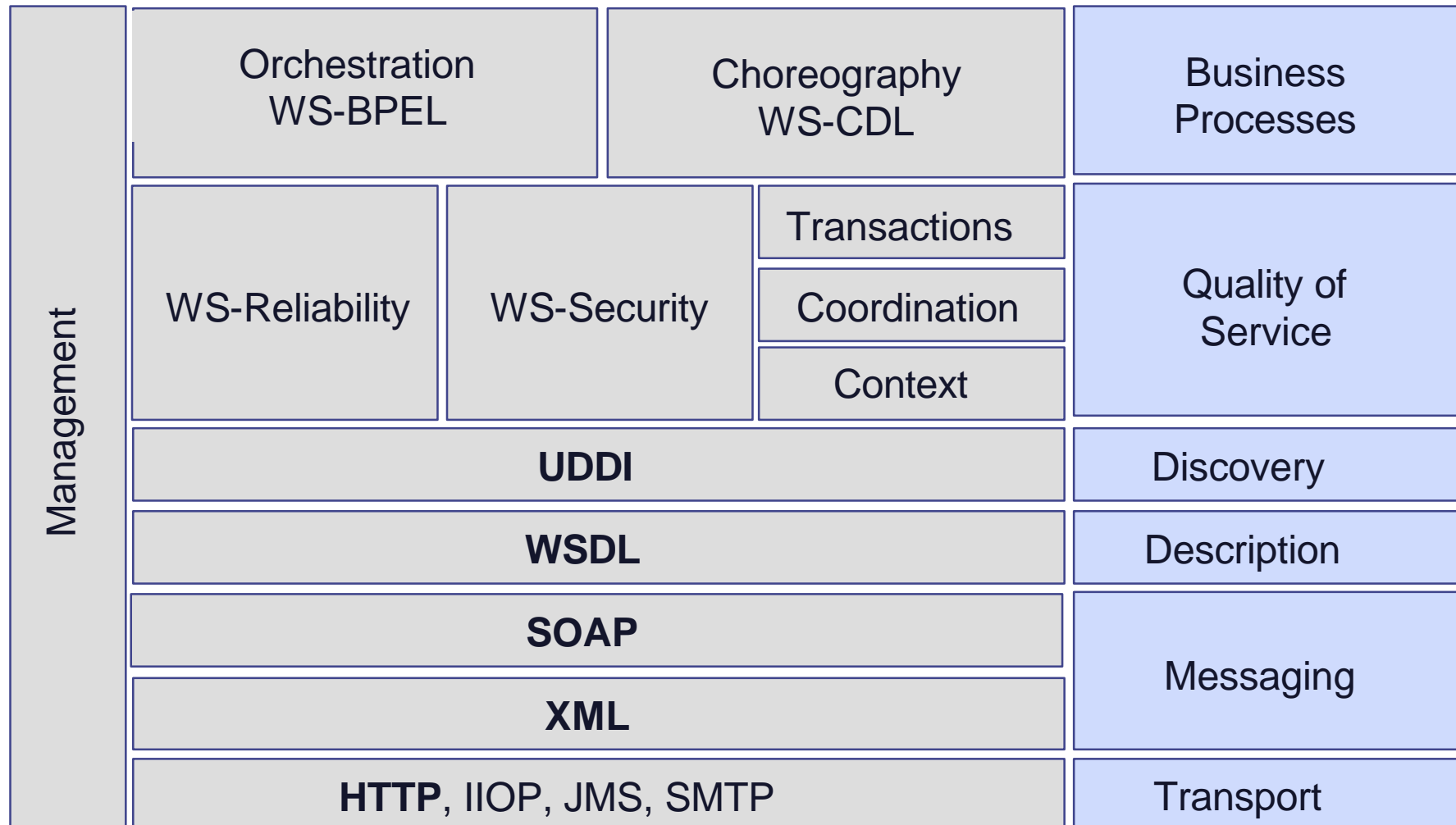
- SOAP, HTTP, XML
- URLs
- WSDL
- UDDI
- Fehlende Standards für Infrastrukturdienste
- Nicht standardisiert
- Kein statisches Type-Checking
- Unterstützung der großen Softwarehersteller
- IIOP, GIOP
- IORs, URLs
- IDL
- Naming und Trading Service, Interface Rep.
- Security, Notification, Transaction Service, ...
- Standardisierte Language Bindings
- Statisches Type-Checking während der Compilierung
- Zahlreiche Softwareherst. ohne Microsoft

Aspect	CORBA	Web services
Data model	Object model	SOAP message exchange model
Client-Server coupling	Tight	Loose
Location transparency	Object references	URL
Type system	IDL	XML schemas
	static + runtime checks	runtime checks only
Error handling	IDL exception	SOAP fault messages
Serialization	built into the ORB	can be chosen by the user
Parameter passing	by reference	by value (no notion of objects)
	by value (<i>valuetype</i>)	
Transfer syntax	CDR used on the wire	XML used on the wire
	binary format	Unicode
State	stateful	stateless
Request semantics	at-most-once	defined by SOAP
Runtime composition	DII	UDDI/WSDL
Registry	Interface Repository	UDDI/WSDL
	Implementation repository	
Service discovery	CORBA naming/trading service	UDDI
	RMI registry	
Language support	any language with an IDL binding	any language
Security	CORBA security service	HTTP/SSL, XML signature
Firewall Traversal	work in progress	uses HTTP port 80
Events	CORBA event service	N/A

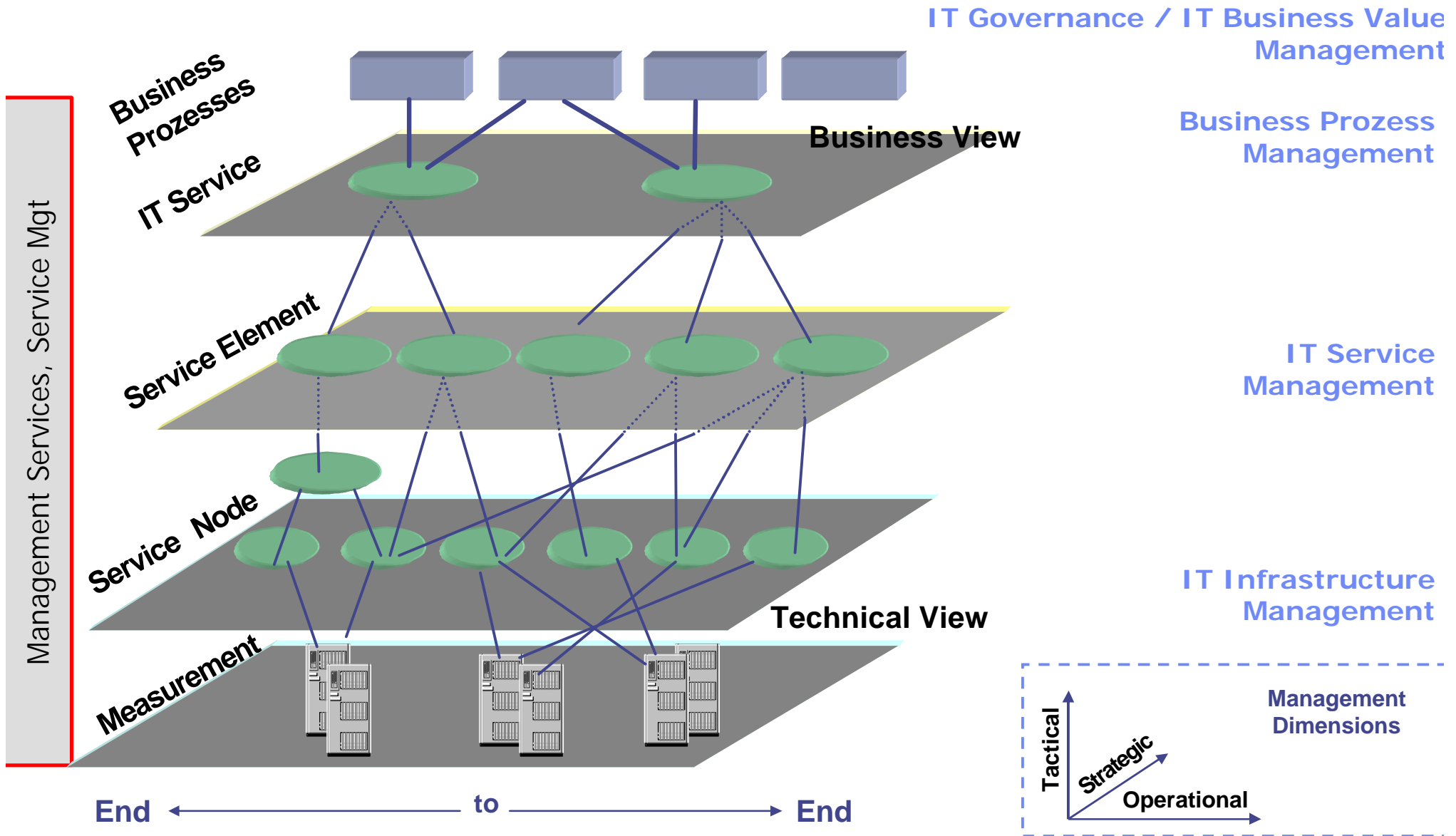
Extended Service-Oriented Architecture (SOA)



WS-Stack und -Standards



IT Service Management (ITSM) and Business Process Mgt. (BPM)



Literatur

Bücher: Einführungskapitel sowie Kapitel über SOAP, WSDL, UDDI

- „**Web Services Platform Architectures**“, **S. Weerawarana et al.**
- „Service-orientierte Architekturen mit Web Services“, Dostal et al.
- „Building Web Services with Java“, S. Graham et al.
- „Web Services mit Java“, T. Langer

Internet Quellen

- Specifications:W3C Web Services
 - **WSDL (1.1)** - <http://www.w3.org/2002/ws/desc/>
 - **SOAP** - www.w3.org/TR/soap/
 - **UDDI** - <http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3>
- Tutorials zu SOAP und WSDL
 - <http://www.w3schools.com/soap/default.asp>
 - <http://www.w3schools.com/wsdl/default.asp>
- Tools und Information
 - <http://www-106.ibm.com/developerworks/webservices/>
 - <http://ws.apache.org/>
 - <http://java.sun.com/webservices/>

Praktische Tipps für die Uebung

- Liste aller Web Services ausgeben

```
java org.apache.axis.client.AdminClient -l http://<Ihre IP>:<Ihr Port>/axis/servlet/AxisServlet list
```
- Zugriff auf SOAP-Nachricht(en) über MessageContext Objekt

```
Call rpc ...  
MessageContext context = rpc.getMessageContext()
```
- Über Integration zusätzlicher Handler können Funktionalitäten wie Logging oder Sicherheit auf der Serverseite integriert werden.
 - Handler verarbeiten Request- und Response-Nachrichten
 - Mehrere Handler können zu Ketten (Chains) zusammengefasst werden
 - Handler erweitern die Klasse `BasicHandler()`
 - Handler-Anmeldung über Deployment Descriptor: `Handler-Element`