

---

# Web Services

## Block Web Services

---

### heutige Vorlesung

- Was sind Web Services?
- Nachrichtenformat SOAP
- Schnittstellenbeschreibung WSDL
- Anwendungen
- RPC vs. Messaging

### 22.6.

- SOAP im Detail

### 29.6.

- WSDL im Detail

### 6.7.

- Web Services in der Praxis

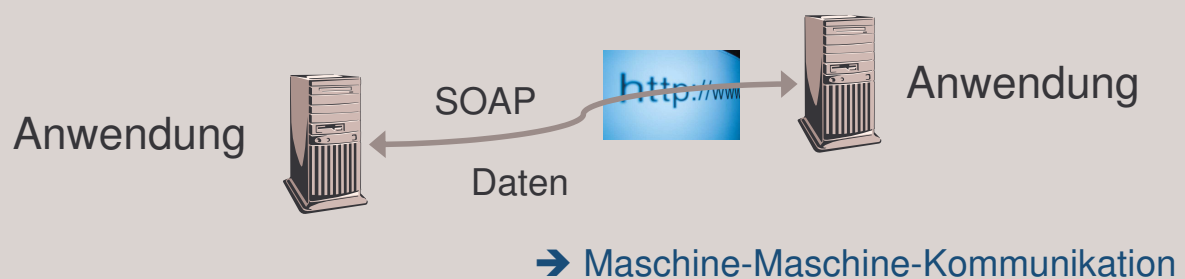
# Was sind Web Services?

## Was sind Web Services?

### traditionelle Web-Anwendung



### Web Service



# Beispiel: Google ohne Browser



With Google Web APIs, your computer can do the searching for you.

## Google als Web Service

- Suche
- Rechtschreibkorrektur
- Zugriff auf Web-Cache

## Suche als Web Service

- Suchanfrage als SOAP-Nachricht.
- Suchergebnis als SOAP-Nachricht.

➔ <http://www.google.com/apis/>

# Google-Suche ohne Browser



- Google-Suche kann aus Anwendungsprogramm heraus aufgerufen werden

## mögliche Anwendungen

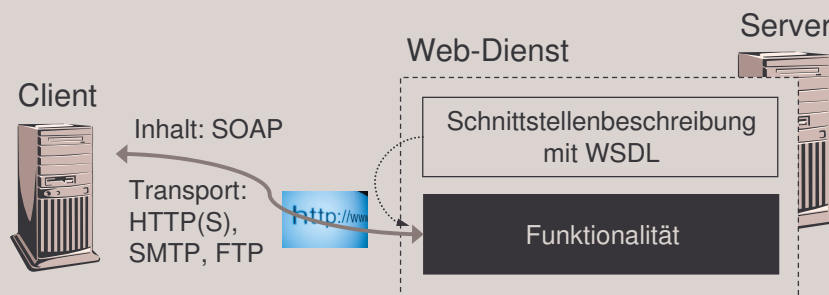
- in periodischen Abständen zu bestimmten Thema nach neuen Webseiten zu suchen:



### Web Alert

search-for: "XSLT 2.0 Recommendation"  
notify new web page: mymail@inf.fu-berlin.de

- automatisch neue Trends im WWW zu identifizieren



Ein **Web Service (Web-Dienst)** ist eine Software, die

1. auf einem Server bereitgestellt wird,
2. eine bestimmte Funktionalität als Blackbox zur Verfügung stellt,
3. über gängige Internet-Protokolle unter Benutzung von SOAP zugreifbar ist und
4. über eine mit WSDL beschriebene Schnittstelle verfügt.

- implementieren *keine neuen* Systeme
- Fassade für bestehende Systeme, um diese einfach zuzugreifen
- nutzen gängige Internet-Protokolle wie HTTP(S) und SMTP
- verwenden XML-Standards SOAP und WSDL
- unabhängig von Programmiersprachen und Betriebssystemen
- zwei Erscheinungsformen: RPCs (synchron) oder Messaging (asynchron)

# Alter Wein in neuen Schläuchen?



- Web-Dienste *keine* revolutionär neue Technologie
- große Ähnlichkeiten mit CORBA

## Neu ist jedoch:

„What would happen if Microsoft, IBM, and Sun could all agree on something for a change? Well, they have. [...] All three are ganging up on poor customers with one thought in mind: Sell them Web services.“ (David Coursey, 2002)

- alle bedeutenden IT-Unternehmen auf Standards geeinigt: XML, SOAP und WSDL
- CORBA hingegen nie von Microsoft unterstützt

# Alter Wein in neuen Schläuchen?



## Außerdem ist neu:

- statt proprietäre Protokolle (wie IIOP und DCOM) gängige Internet-Protokolle
- nicht nur RPCs, sondern auch Messaging



## Das Nachrichtenformat SOAP

**HTML**

```
<html>  
  <head>  
    Zusatzinformationen  
  </head>  
  <body>  
    Inhalt: Webseite  
  </body>  
</html>
```

**SOAP**

```
<Envelope>  
  <Header>  
    Zusatzinformationen  
  </Header>  
  <Body>  
    Inhalt: XML-Daten  
  </Body>  
</Envelope>
```

- XML-basierter W3C-Standard
- SOAP-Nachricht enthält Daten und keine Webseiten.
- Seit SOAP 1.2 steht SOAP *nicht* mehr für Simple Object Access Protocol!

# Eine SOAP-Anfrage an Google



```
<?xml version='1.0' encoding='UTF-8'?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="..." xmlns:xsi="...">
  <env:Body>
    <doGoogleSearch xmlns="urn:GoogleSearch">
      <key xsi:type="xsd:string">3289754870548097</key>
      <q xsi:type="xsd:string">Eine Anfrage</q>
      <start xsi:type="xsd:int">0</start>
      <maxResults xsi:type="xsd:int">10</maxResults>
      ...
    </doGoogleSearch>
  </env:Body>
</env:Envelope>
```

- doGoogleSearch(key, q, start, maxResults,...)
- hier kein Header
- Datentypen mit xsi:type spezifiziert
- vordefinierte Datentypen aus XML-Schema

© Klaus Schild, 2005

# Und die Antwort von Google



```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="..." xmlns:xsi="...">
  <env:Body>
    <ns1:doGoogleSearchResponse xmlns:ns1="urn:GoogleSearch" ...>
      <return xsi:type="ns1:GoogleSearchResult">
        ...
      </return>
    </ns1:doGoogleSearchResponse>
  </env:Body>
</env:Envelope>
```

- Antwort: doGoogleSearchResponse(return(...))
- Datentyp ns1:GoogleSearchResult in WSDL-Beschreibung definiert

© Klaus Schild, 2005

# Übertragung von SOAP-Nachrichten

---



- heute meist über HTTP oder HTTPS
- Request-Response-Verhalten von HTTP unterstützt entfernte Prozeduraufrufe (RPCs).
- mit HTTP auch RPCs über eine Firewall hinweg
- Übertragung aber auch z.B. mit SMTP möglich (*Messaging*).

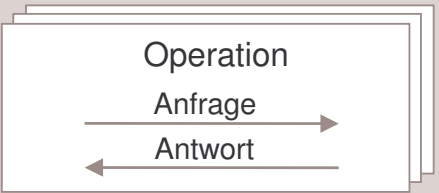


---

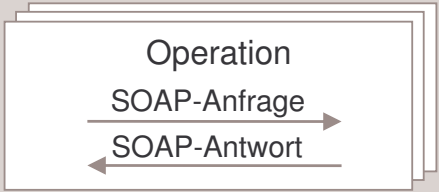
# WSDL

## Servicebeschreibung

### abstrakte Schnittstelle



### konkrete Schnittstelle



### Web-Adressen (End Points)

- beschreibt Interface (→ IDL)
- XML-basierter Standard
- W3C Note, keine Recommendation

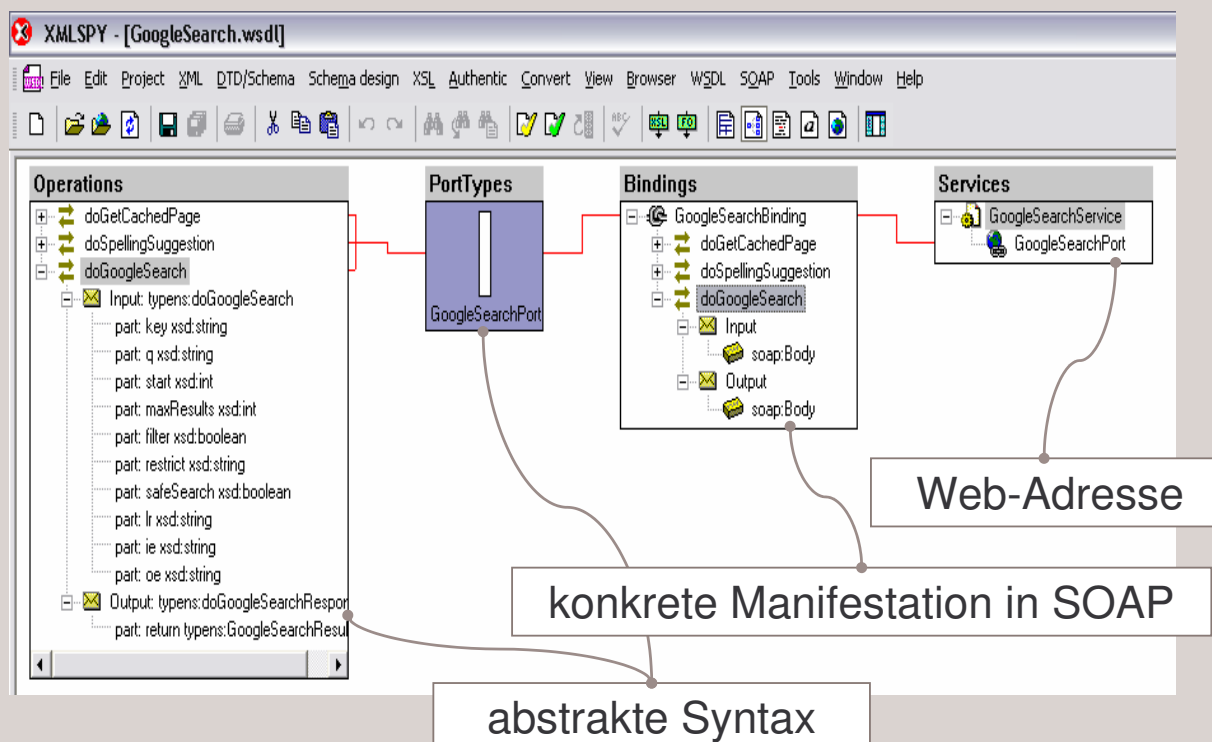
### Abstrakte Schnittstelle (*port types*)

- Schnittstelle unabhängig von Nachrichtenformaten
- mit XML-Schema beschrieben

### konkrete Schnittstelle (*binding*):

- Abbildung der abstrakten Schnittstelle auf unterstützte Nachrichtenformate

# Die WSDL-Beschreibung von Google



- beschreibt Schnittstellen eines Web Services und wo dieser abgerufen werden kann
- baut auf XML-Schema auf
- Syntax einer Schnittstelle kann bis ins kleinste Detail festgelegt werden.
- Grundlegende Interaktionsmuster (wie Anfrage-Antwort) können beschrieben werden.
- Semantische Eigenschaften können nicht beschrieben werden:
  - Funktionalität
  - Verfügbarkeit
  - etc.

# Anwendungen

## Enterprise Application Integration

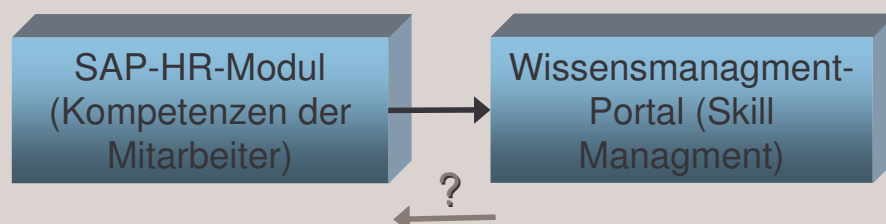
- Systemintegration per Knopfdruck

### diensteorientierte Architektur

- Beispiel Amazon Light 4

## Enterprise Application Integration

- **technologisch**: inkompatible IT-Systeme miteinander verbinden
- **inkompatibel** kann bedeuten:
  - unterschiedliche Betriebssysteme
  - unterschiedliche Programmiersprachen
  - unterschiedliche Kommunikationsprotokolle
- **organisatorisch**: Geschäftsprozesse optimieren
- Beispiel:



# Zwang zur Systemintegration



## unternehmensintern

- Beispiel Mercedes-Benz-Werk
  - mehr als 200 EDV-Systeme
  - sehr gut vernetzt (LAN)
  - Systeme also prinzipiell integrierbar
- Neue Systeme werden nur dann eingeführt, wenn Alt-Systeme integriert werden.

## unternehmensübergreifend

- E-Business setzt Zusammenarbeit von heterogenen Systemen voraus:
  - Unternehmen ↔ Unternehmen
  - Unternehmen ↔ Portal

# Kosten der Systemintegration



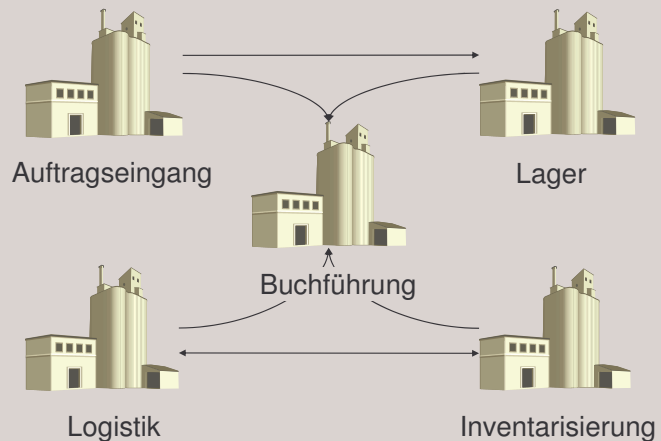
## Systemintegration bindet

- **35%** der IT-Personal-Ressourcen eines Unternehmens (Forrester Research, 2002)
- **65%** der Arbeitszeit eines Programmierers (Gartner)

⇒ Systemintegration häufig als **notwendiges Übel** betrachtet

- Systemintegration aber auch Chance zur Flexibilisierung eines Unternehmens

# Entstehung von Datensilos



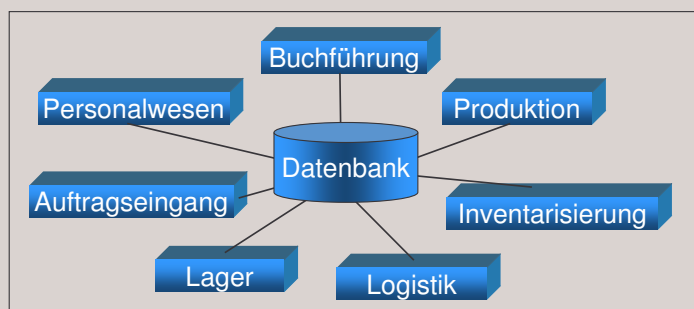
- jede Abteilung eigenes Informationssystem
- Informationsfluss zwischen Systemen:
  - Schnittstelle Mensch  
Medium: Ausdruck, Diskette, Telefon, Drehstuhl
  - Screen scraping

# Enterprise Resource Planning



E-Commerce ?

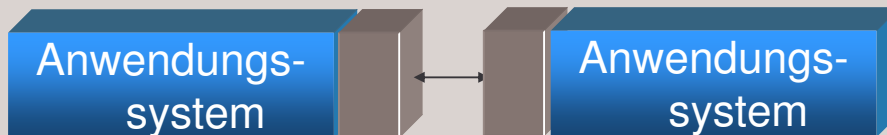
- Beispiel: SAP/R3
- großer Fortschritt bei der Datenintegration
- sehr teuer:
  - > 53.000 \$ (TCO) pro Nutzer für ersten zwei Jahr seit Installation (Meta Group, 2002)
- ein System für alles
- viele Datensilos durch ein großes Datensilo ersetzt:
- schwierige Integration von externen Systemen



# Web Services



- statt jedes Anwendungssystem mit jedem anderen zu integrieren ( $n^2$  Integrationsprozesse):
- Anwendungssysteme durch standardisierte Schnittstelle erweitern ( $n$  Erweiterungen):



- Schnittstelle muss allgemein akzeptiert sein
- bei Web Services ist dies der Fall
  - Nachrichtenformat: SOAP
  - Schnittstellenbeschreibung: WSDL
  - Übertragung: Internet-Protokolle

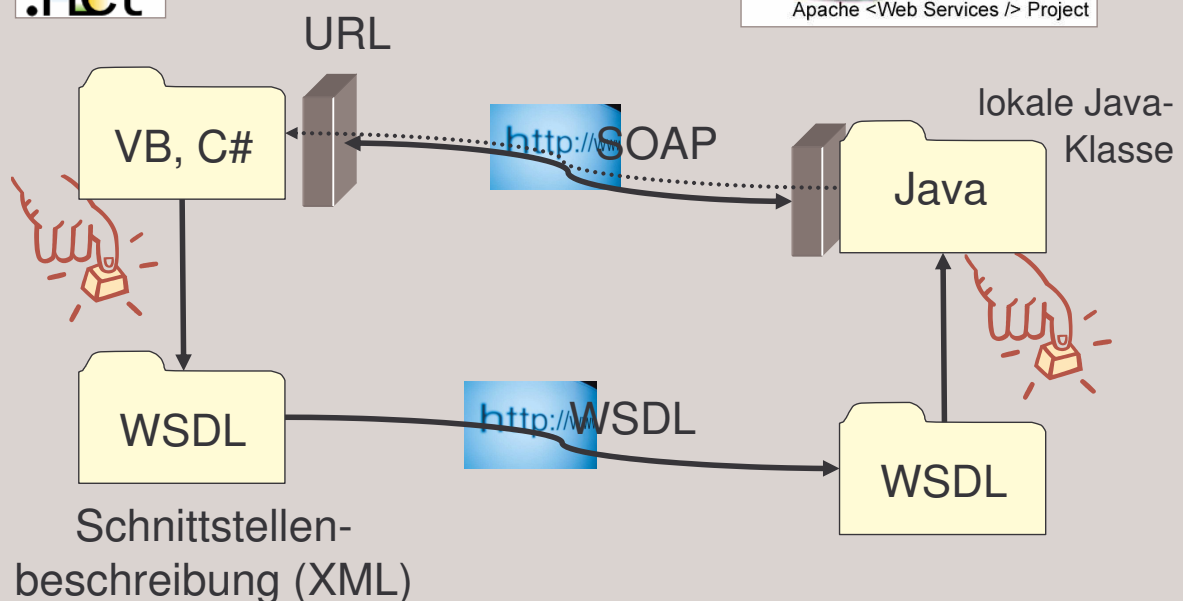
# Systemintegration per Knopfdruck



MS XP



Unix



## Enterprise Application Integration

- Systemintegration per Knopfdruck

## diensteorientierte Architektur ←

- Beispiel Amazon Light 4

# Systemintegration auf den Kopf stellen!

## Diensteorientierte Architekturen (service-oriented architecture, SOA)

- neue Art, IT-Systeme zu entwickeln
- statt IT-Systeme isoliert zu entwickeln, nur um sie später zu integrieren:
- Funktionen von allgemeinem Interesse als Web Service anbieten
- neues System auf anderen Web Services aufbauen
- neues System kann wieder als Web Service angeboten werden

# Beispiel: Amazon Light 4



## Amazon für alle!

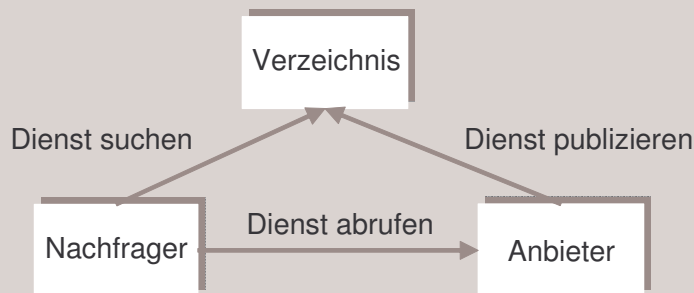
- Amazon bietet auch Web-Service-Schnittstelle an
- ⇒ neue Systeme können auf Amazon-Web-Service aufbauen und mit anderen Web Services kombinieren:
  - Ausleihmöglichkeit in lokaler Stadtbibliothek
  - Angebote auf eBay

The screenshot shows the Amazon website interface for the book 'Beginning XML (Programmer to Programmer)'. The browser title is 'Amazon Light 4 - Beginning XML (Programmer to Programmer) - Mozilla Firefox'. The URL is 'http://www.kklogical.com/amazon/details.asp?SearchIndex=Books&keywords=Beginning%20XML&asin=0764570773'. The search bar contains 'Beginning XML' and '90'. The page layout includes a sidebar with 'Links/Help', 'Same Search at', and 'Yahoo News' sections. The main content area displays the book title, authors, a 'Buy This' button, and a 'Sales Rank' of 9506. It also features an 'Editorial Review' and 'Customer Reviews' section, with a 'Similar Products' section at the bottom showing related books like 'XML: A Programmer's Reference' and 'XML in a Nutshell, 2nd Edition'.

## Vor- und Nachteile

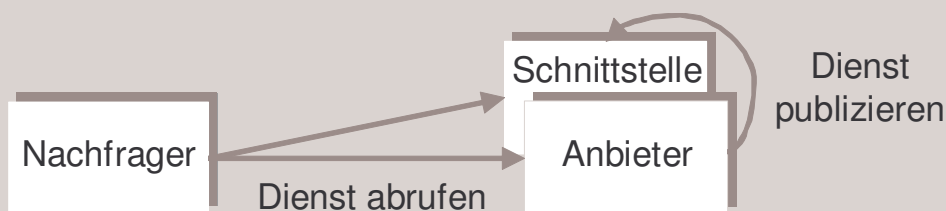


- + ohne dass Nachfrager es bemerkt, kann
  - neue Version eines Web Services freigeschaltet werden
  - Web Service bei Ausfall ersetzt werden
  - Lastverteilung durchgeführt werden
- Vertrauen in korrekte Implementierung nötig



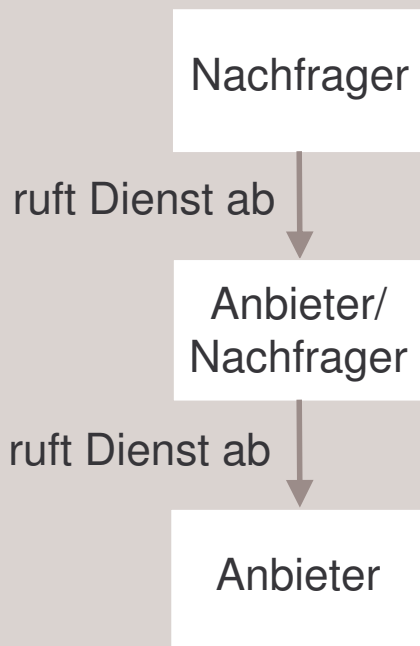
- **publizieren** (*publish*): Beschreibung eines Dienstes in einem Verzeichnis (*registry*) veröffentlichen.
- **suchen** (*find*): Beschreibung eines Dienstes suchen, entweder dynamisch oder zur Entwicklungszeit
- **abrufen** (*bind*): Beschreibung des Dienstes verwenden, um Dienst abzurufen, entweder dynamisch oder zur Entwicklungszeit

## Öffentliches Verzeichnis



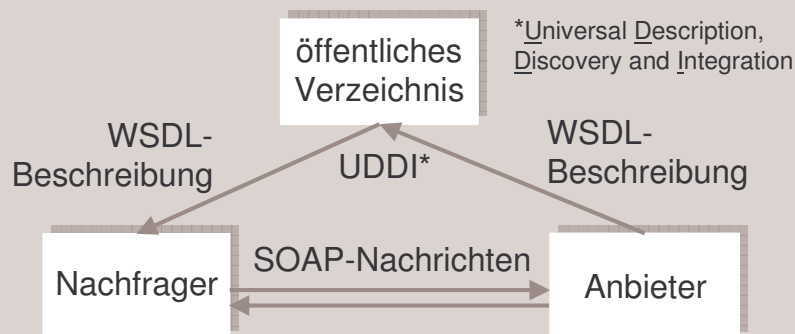
- Öffentliches Verzeichnis *nicht* zwingend notwendig:
- Auch *ohne* öffentliches Verzeichnis kann Dienst gefunden werden.
- Schnittstelle kann z.B. auf Webseite des Anbieters veröffentlicht werden.
- öffentliche Verzeichnisse heute auch kaum genutzt

# Nachfrager vs. Anbieter von Diensten



- **Dienst-Anbieter** (*service provider*) bietet Dienste an.
  - **Dienst-Nachfrager** (*service requestor*) nutzt Dienste anderer Anbieter.
  - Anbieter von Diensten kann *gleichzeitig* Dienste anderer nutzen (und Nachfrager) sein.
- ⇒ Diese Begriffe sind relativ.

# Standards



- SOAP und WSDL allgemein akzeptiert
- allerdings umstritten, ob WSDL-Beschreibungen ausreichen
- UDDI umstritten und wenig genutzt

# RPC vs. Messaging

## Wie SOAP-Nachrichten übertragen?

### über HTTP

- heute üblich
- Request-Response-Verhalten von HTTP unterstützt **RPCs**.
- verbindungsorientiert
- Übertragung: Sender und Empfänger müssen präsent sein.
- typischerweise synchron
- enge Kopplung

### über SMTP

- heute selten
- realisiert **Messaging**
- persistente Kommunikation
- Übertragung: Weder Sender noch Empfänger muss präsent sein.
- typischerweise asynchron
- lose Kopplung
- erlaubt Lastverteilung und Priorisierung

⇒ schwerwiegende Designentscheidung!

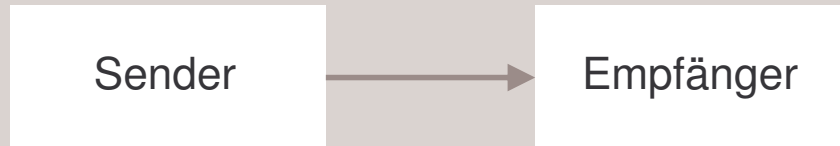
- Anwendungen interagieren durch Austausch von Nachrichten miteinander:
- Kommunikation in jeweiligen Anwendungen sichtbar
- verschiedene Formen des Messaging:
  1. Kommunikationsstruktur
  2. Interaktionsmuster
  3. flüchtig vs. persistent
  4. Synchronität
  5. Qualität (*quality of service*)

Kriterien auch für  
Web Services  
relevant!

## 1. Kommunikationsstruktur

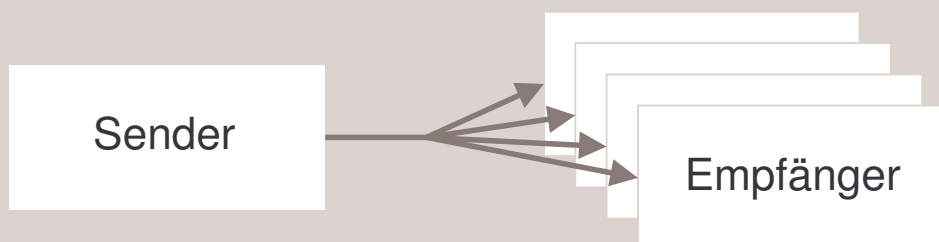
- Anzahl und Organisation der Kommunikationspartner
- wichtigste Kommunikationsstrukturen:
  - Eins-zu-Eins-Kommunikation
  - One-to-Many-Kommunikation

# Eins-zu-Eins-Kommunikation



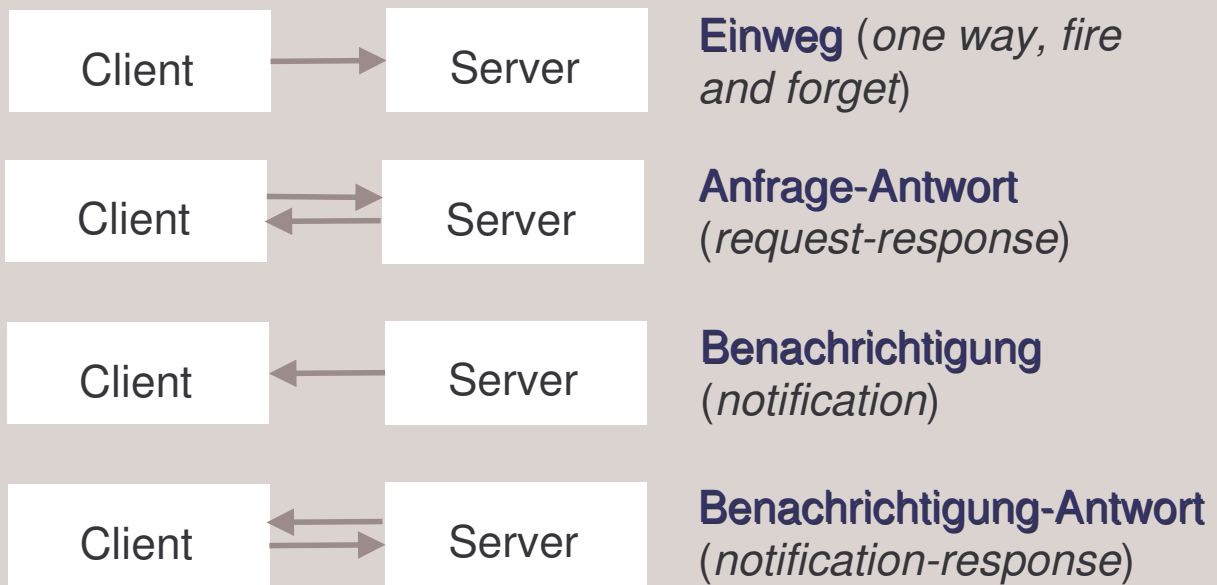
- Sender sendet Nachricht an bestimmten Empfänger.
- Beispiel: Kunde sendet Bestellung per E-Mail an Firma

# One-to-Many-Kommunikation



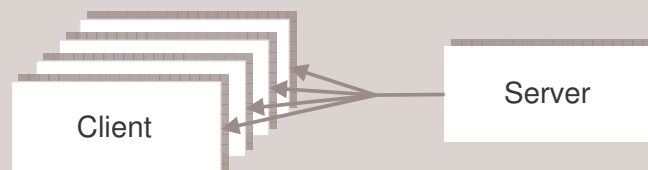
- Sender sendet identische Kopie gleichzeitig an mehrere Empfänger.
- Sender (*publisher*) veröffentlicht Nachricht zu einem bestimmten Thema (*topic*), zu dem sich die Empfänger (*subscriber*) angemeldet haben.
- auch *publish-subscribe* oder *topic-based messaging* genannt
- Beispiel: Mailing-Liste

## 2. Interaktionsmuster



Beachte: „Antwort“ bezieht sich auf jeweilige *Anwendung*, nicht auf das Netzwerk.

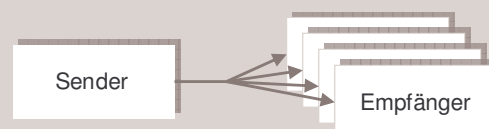
## Beispiel Börsenticker



- Interaktionsmuster: Benachrichtigung (*notification*)



- Kommunikationsstruktur: One-to-Many (*publish-subscribe*)



# Komplexe Interaktionsmuster



- Registrierung
- ← Bestätigung der Registrierung
- ← aktuelle Aktienkurse (Benachrichtigung)

## Börsenticker



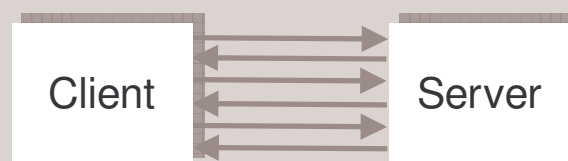
- Abfolge der Nachrichten wichtig
- z.B. keine Benachrichtigung ohne vorherige Registrierung
- auch als Konversation (*conversation*) bezeichnet

# Weiteres Beispiel

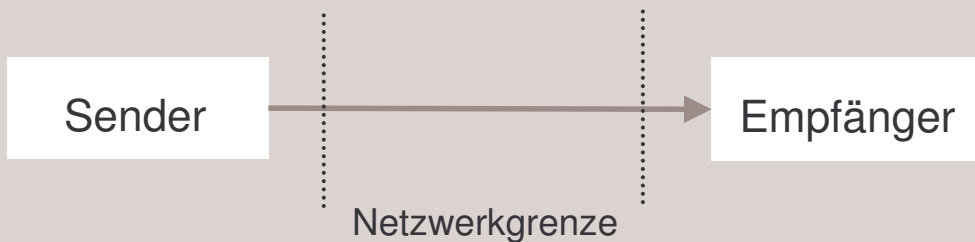


- Bestellanfrage mit spätestem Liefertermin
- ← Angebot mit zugesichertem Liefertermin
- Bestellung
- ← Bestätigung des Eingangs der Bestellung
- Bestätigung der Lieferung
- ← Rechnung

## Bestellvorgang



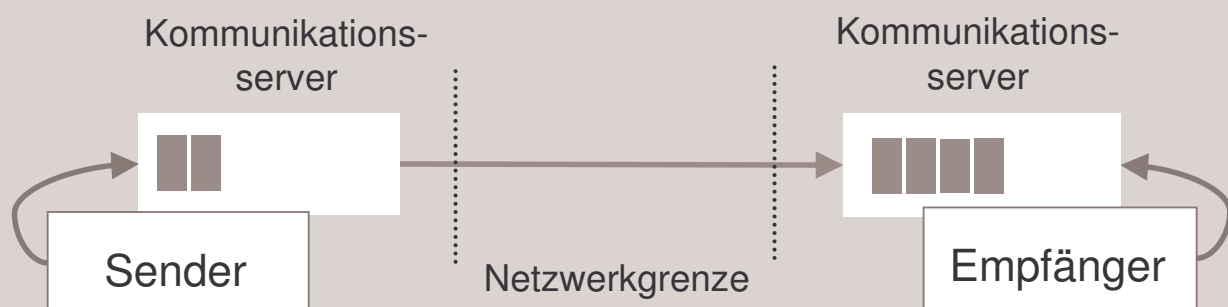
### 3. Flüchtig vs. persistent



#### flüchtige Kommunikation

- Sender und Empfänger kommunizieren direkt ohne Puffer miteinander.
- Sender und Empfänger müssen während der gesamten Übertragung präsent sein
- engl. *transient*
- Beispiel: HTTP, UDP

### Flüchtig vs. persistent



#### persistente Kommunikation

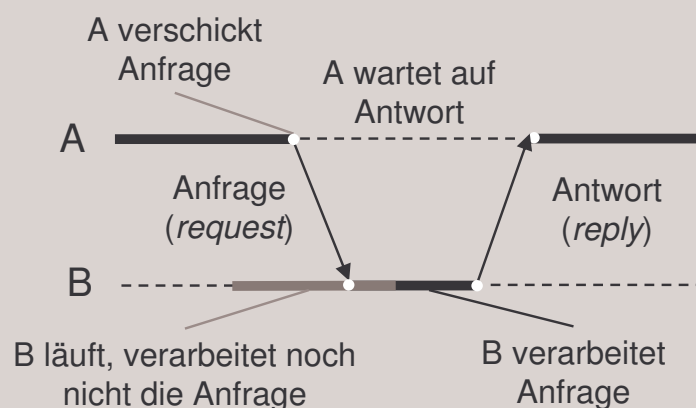
- Nachricht solange gespeichert, bis sie tatsächlich zugestellt wurde.
- Weder Sender noch Empfänger müssen während Übertragung präsent sein.
- Beispiel: E-Mail, MQSeries (IBM), JMS (J2EE)

## 4. Synchronität



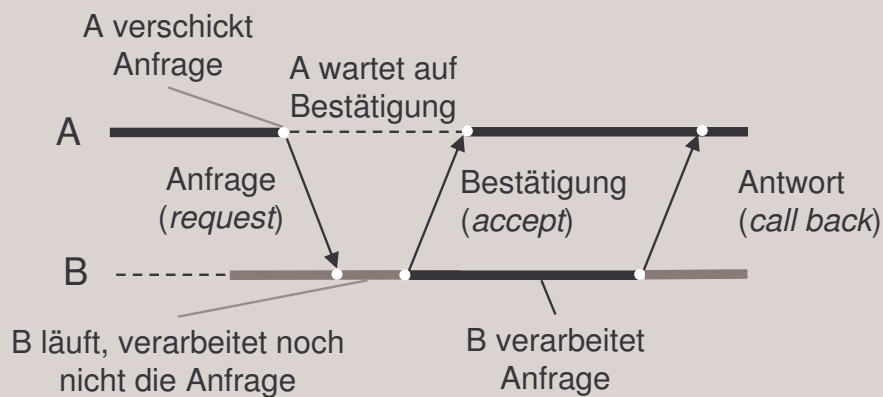
- auch beim Messaging Unterscheidung zwischen synchron oder asynchron
- persistente Kommunikation
  - typischerweise asynchron (⇒ Puffer)
  - theoretisch aber auch synchron: Sender solange blockiert, bis Empfang der Nachricht bestätigt
- flüchtige Kommunikation
  - auch in der Praxis sowohl synchron als auch asynchron

## Beispiel 1



- flüchtige, synchrone Kommunikation
- auch **antwortorientiert** (*response-based*) genannt
- implementiert synchrone RPCs
- jedoch ≠ RPCs, da Kommunikation für Prozesse sichtbar

## Beispiel 2



- flüchtige, asynchrone Kommunikation
- auch **bestätigungsorientiert** (*delivery-based*) genannt
- implementiert asynchrone RPCs

## 5. Qualität (*quality of service*)



- Verfügbarkeit  
z.B. 7 Tage/Woche, 24 Stunden/pro Tag
- maximale Verzögerung (*latency*)
- maximale Last
- Toleranz gegenüber Ausfällen  
z.B. ob Puffer persistent Datenbank gespeichert
- Verschlüsselung
- Authentifizierung
- Nachrichten erreichen auf jeden Fall den Empfänger
- Nachrichten werden höchstens einmal zugestellt

## RPC

→ eng gekoppelte, starre Systeme

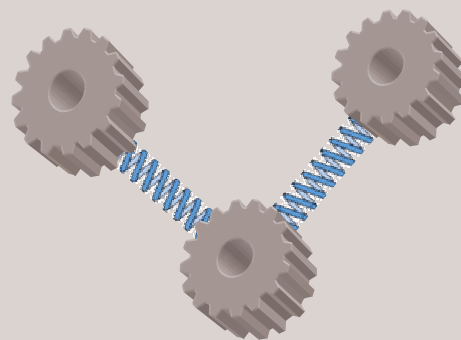
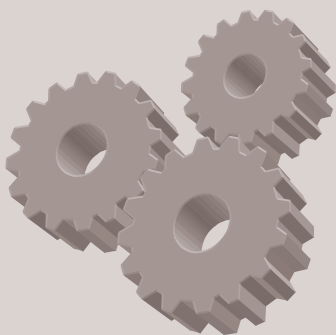
- + einfach, abstrahiert von Kommunikation
- nur Eins-zu-Eins-Kommunikation
- Client und Server müssen präsent sein.
- weniger gut skalierbar

## Messaging

→ lose gekoppelte, robuste Systeme

- abstrahiert nicht von Kommunikation
- + erlaubt auch One-to-Many-Kommunikation
- + Weder Sender noch Empfänger müssen präsent sein.
- + erlaubt Priorisierung und Lastverteilung
- + sehr gut skalierbar

# Enge vs. lose Koppelung



## enge Kopplung

- direkte, synchrone Kommunikation
- z.B. SOAP über HTTP

## lose Kopplung

- indirekte (gepufferte), asynchrone Kommunikation
- z.B. SOAP über SMTP
- robuster, aber auch komplexer zu entwerfen

## heutige Vorlesung

- ☑ Was sind Web Services?
- ☑ Nachrichtenformat SOAP
- ☑ Schnittstellenbeschreibung WSDL
- ☑ Anwendungen
- ☑ RPC vs. Messaging

### **22.6.**

- SOAP im Detail

### **29.6.**

- WSDL im Detail

### **6.7.**

- Web Services in der Praxis