



Netzprogrammierung Web Services

Prof. Dr.-Ing. Robert Tolksdorf, Klaus Schild,
Malgorzata Mochol, Lyndon Nixon
Freie Universität Berlin
Institut für Informatik
Netzbasierte Informationssysteme
mailto: tolk@inf.fu-berlin.de
<http://www.robert-tolksdorf.de>

- Was sind Web Services?
- Web Services – Basistechnologien:
 - SOAP
 - WSDL
 - UDDI
- Enterprise Application Integration
- Dienstorientierte Architektur (SOA)



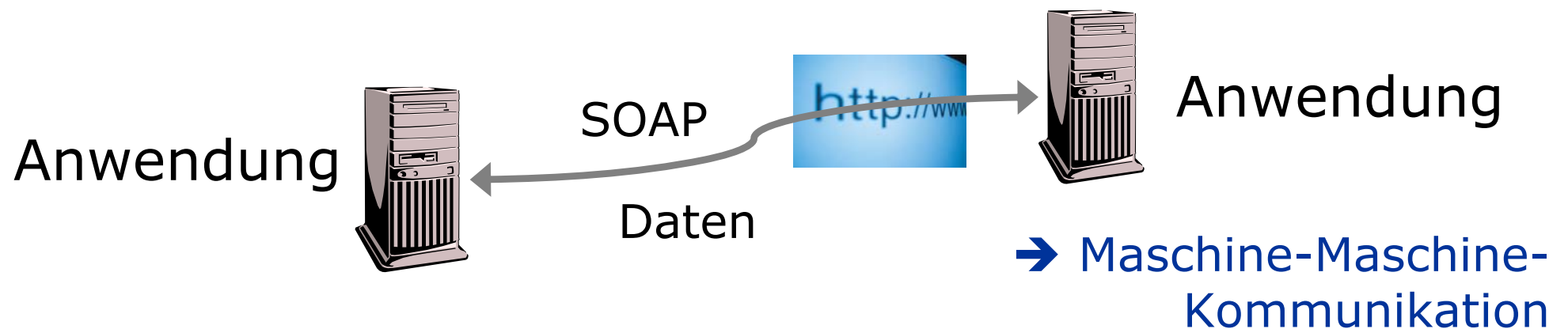
Was sind Web Services?

- Email: Mensch-Mensch-Kommunikation
 - Versendung von uninterpretiertem Text
- WWW: Mensch-Computer-Kommunikation
 - Mensch – aktiv: konsumiert & gibt an, was er als nächstes lesen möchte.
 - der Computer – passiv
- Web Service: Computer-Computer-Kommunikation
 - Computer erbringt für einen anderen eine Dienstleistung

traditionelle Web-Anwendung



Web Service





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

Google als Web Service

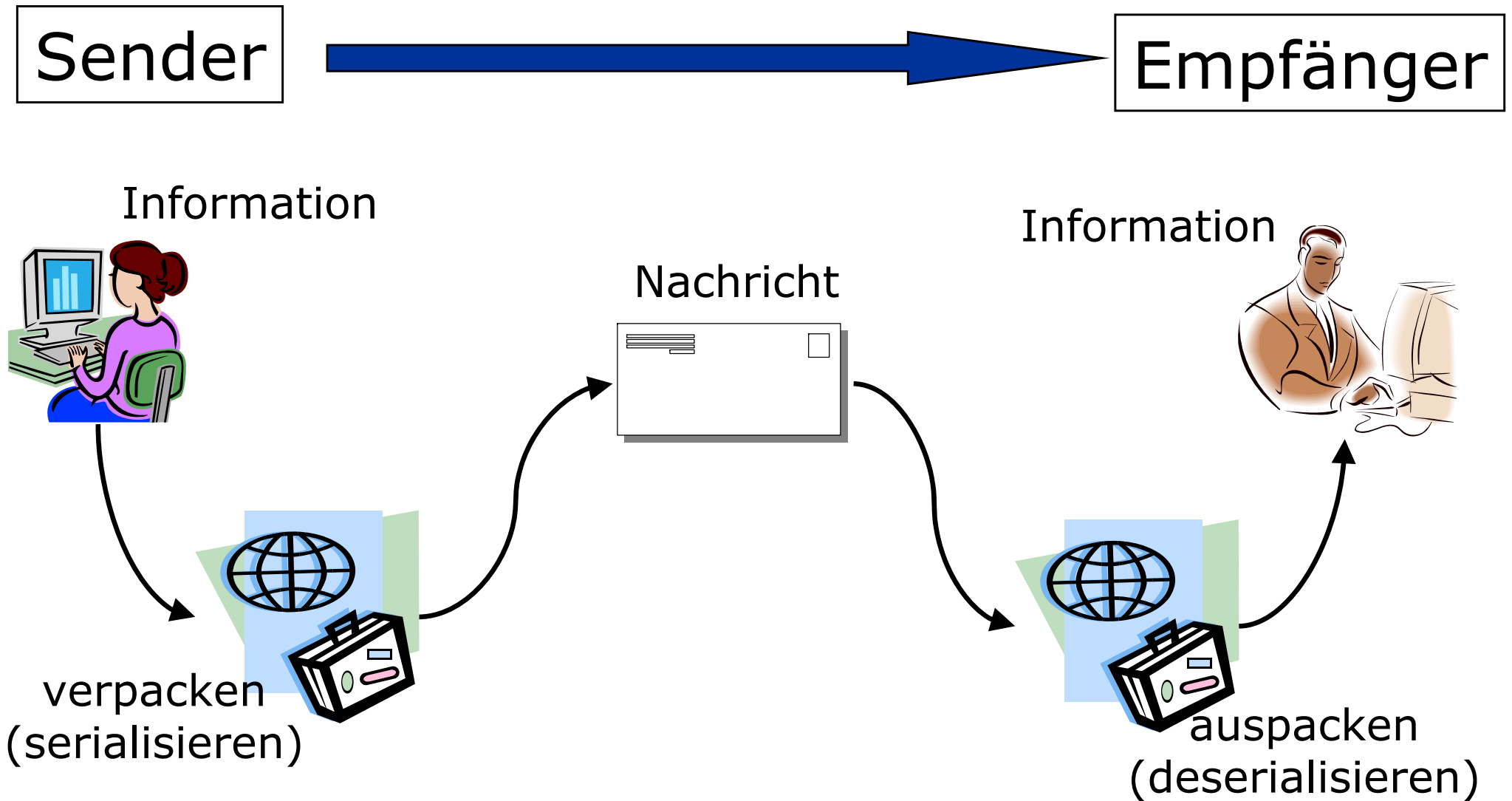
- Suche
- Rechtschreibkorrektur
- Zugriff auf Web-Cache

Suche als Web Service

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

→ <http://www.google.com/apis/>

Austausch einer SOAP-Nachricht



- Google-Suche kann aus Anwendungsprogramm heraus aufgerufen werden

mögliche Anwendungen

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

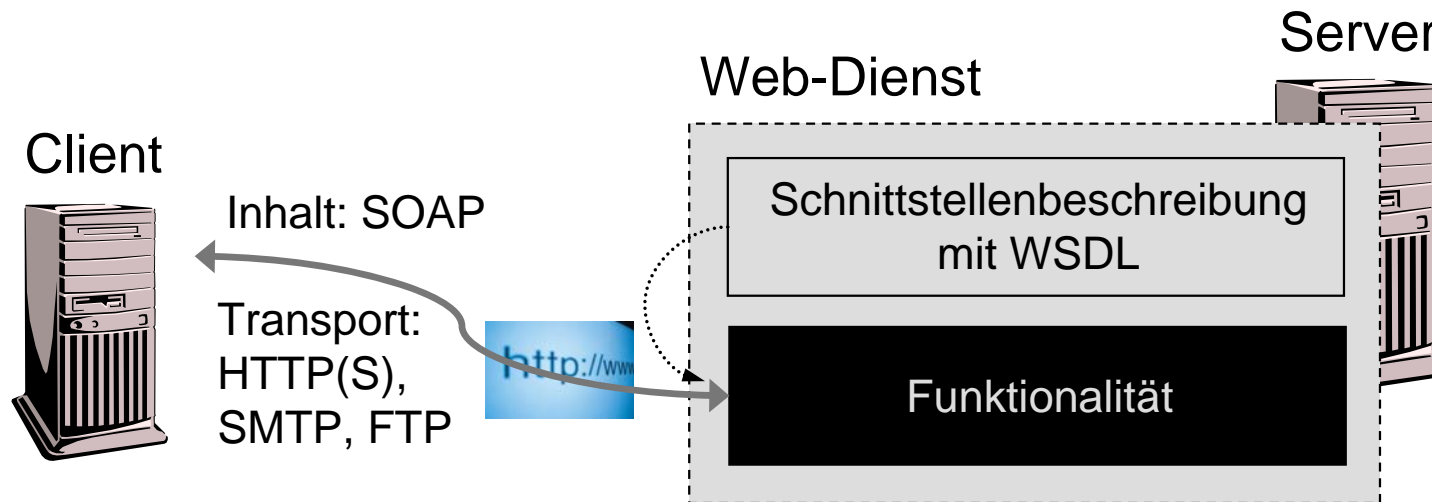
Web Alert

search-for: "XSLT 2.0 Recommendation"

notify new web page: mymail@inf.fu-berlin.de

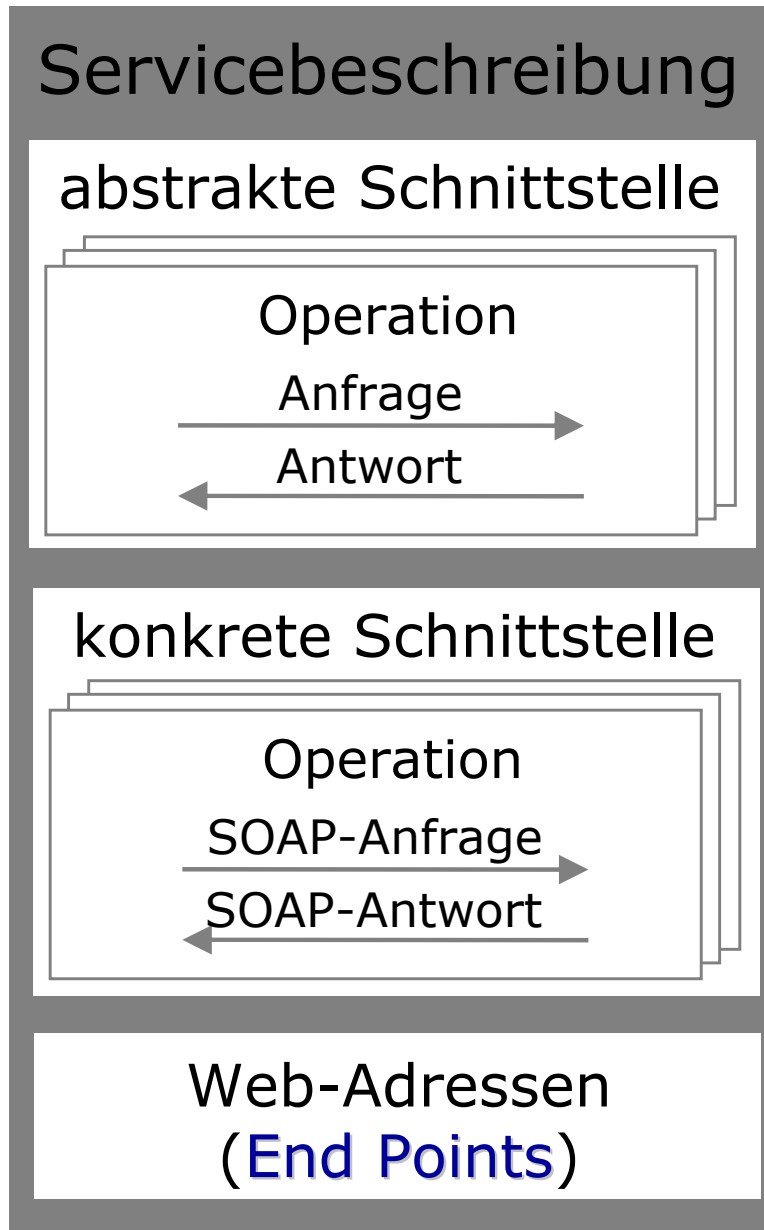
- automatisch neue Trends im WWW zu identifizieren





- Ein **Web Service** ist eine Softwareanwendung, die
1. mit einer **URI** eindeutig identifizierbar ist,
 2. über eine **WSDL**-Schnittstellenbeschreibung verfügt,
 3. nur über die in ihrer WSDL beschriebenen Methoden zugreifbar ist und
 4. über **gängige Internet-Protokolle** unter Benutzung von XML-basierten Nachrichtenformaten wie z.B. **SOAP** zugreifbar ist.

Web Services Description Language



- beschreibt Interface (IDL)
- XML-basierter Standard
- W3C Note von 2001
- abstrakte Schnittstelle (port type)**
- Schnittstelle unabhängig von Nachrichtenformaten
- Beschreibung mit XML-Schema
- konkrete Schnittstelle (binding)**
- Abbildung der abstrakten Schnittstelle auf unterstützte Nachrichtenformate

- implementieren häufig keine neuen Systeme, sondern sind **Fassade** für bestehende Systeme
- abstrahieren von Programmiersprache und Plattform mit der die Anwendung realisiert ist:
Virtualisierung von Software
- zwei Erscheinungsformen:
 - **RPCs (synchron)**
 - **Messaging (asynchron)**

- Web Services keine revolutionär neue Technologie
- große Ähnlichkeiten mit CORBA

Neu ist jedoch:

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

Außerdem neu:

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

Layer	Protokoll/Standards
Messaging	HTML,...
Transport	HTTP, FTP, SMTP
Network	TCP/IP, UDP

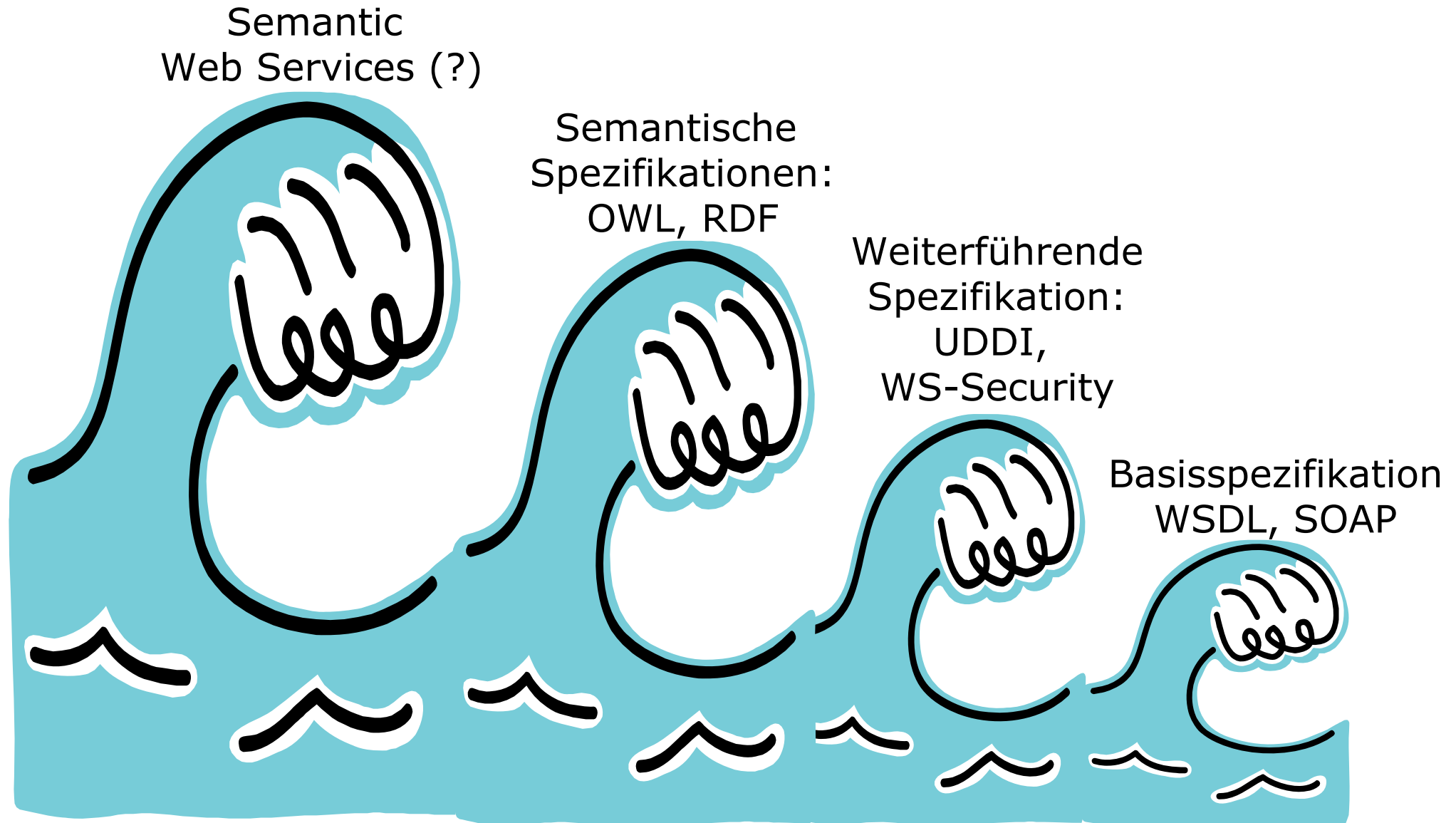
Die Protokolle und Standards aller Layer nur in Auswahl
Quelle: http://www.aki-dpg.de/Dokumente/Bad_Honnef_2003/webservicestutorial.pdf

Layer	Protokoll/Standards
Content	Content Information
Messaging	SOAP, XML-RPC
Transport	HTTP, FTP, SMTP
Network	TCP/IP, UDP

Die Protokolle und Standards aller Layer nur in Auswahl
Quelle: http://www.aki-dpg.de/Dokumente/Bad_Honnef_2003/webservicestutorial.pdf

Layer	Protokoll/Standards
Discovery	UDDI, DISCO, WSIL
Description	WSDL, RDF
Messaging	SOAP, XML-RPC
Transport	HTTP, FTP, SMTP
Network	TCP/IP, UDP

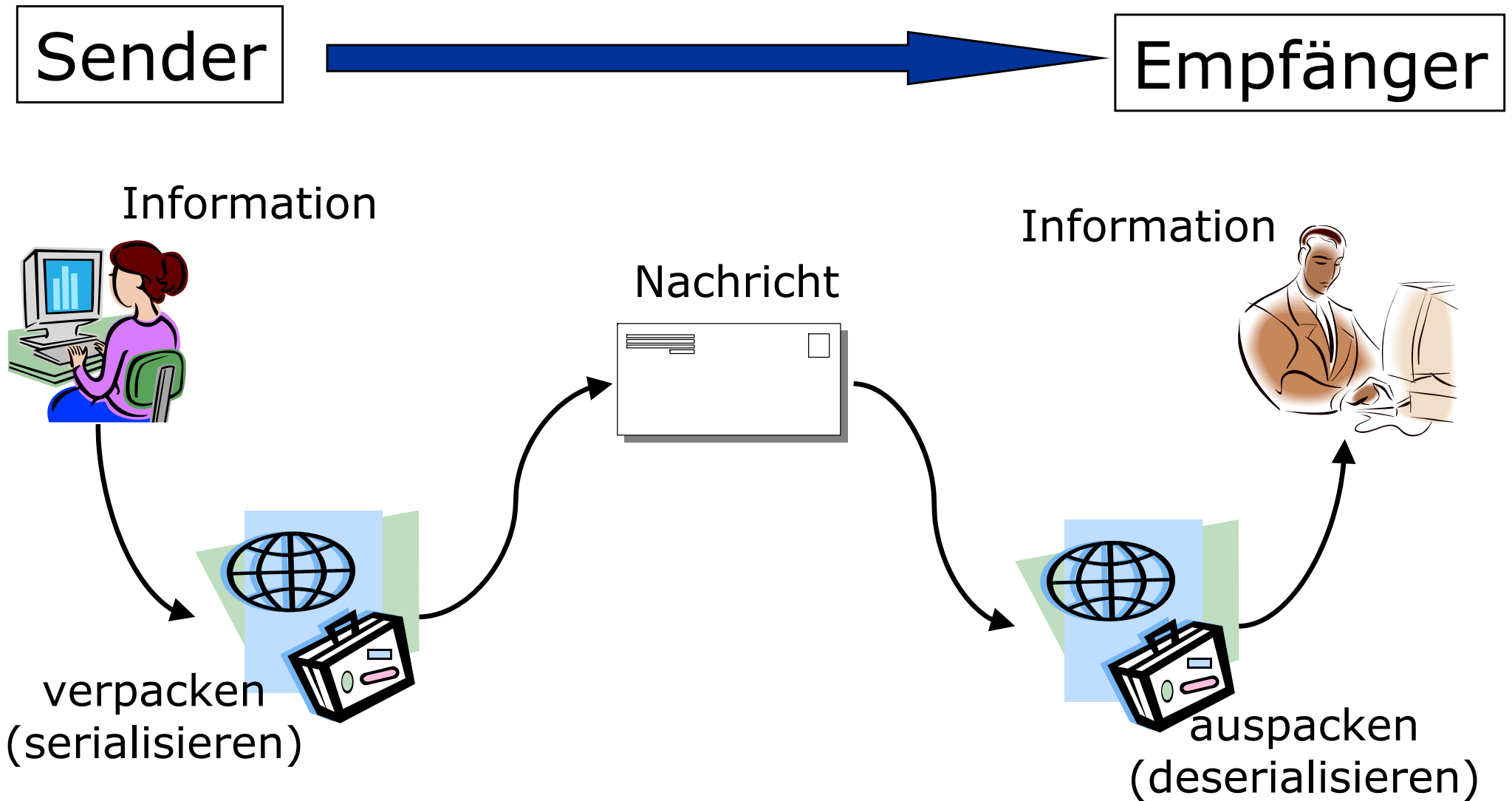
Die Protokolle und Standards aller Layer nur in Auswahl
Quelle: http://www.aki-dpg.de/Dokumente/Bad_Honnef_2003/webservicestutorial.pdf



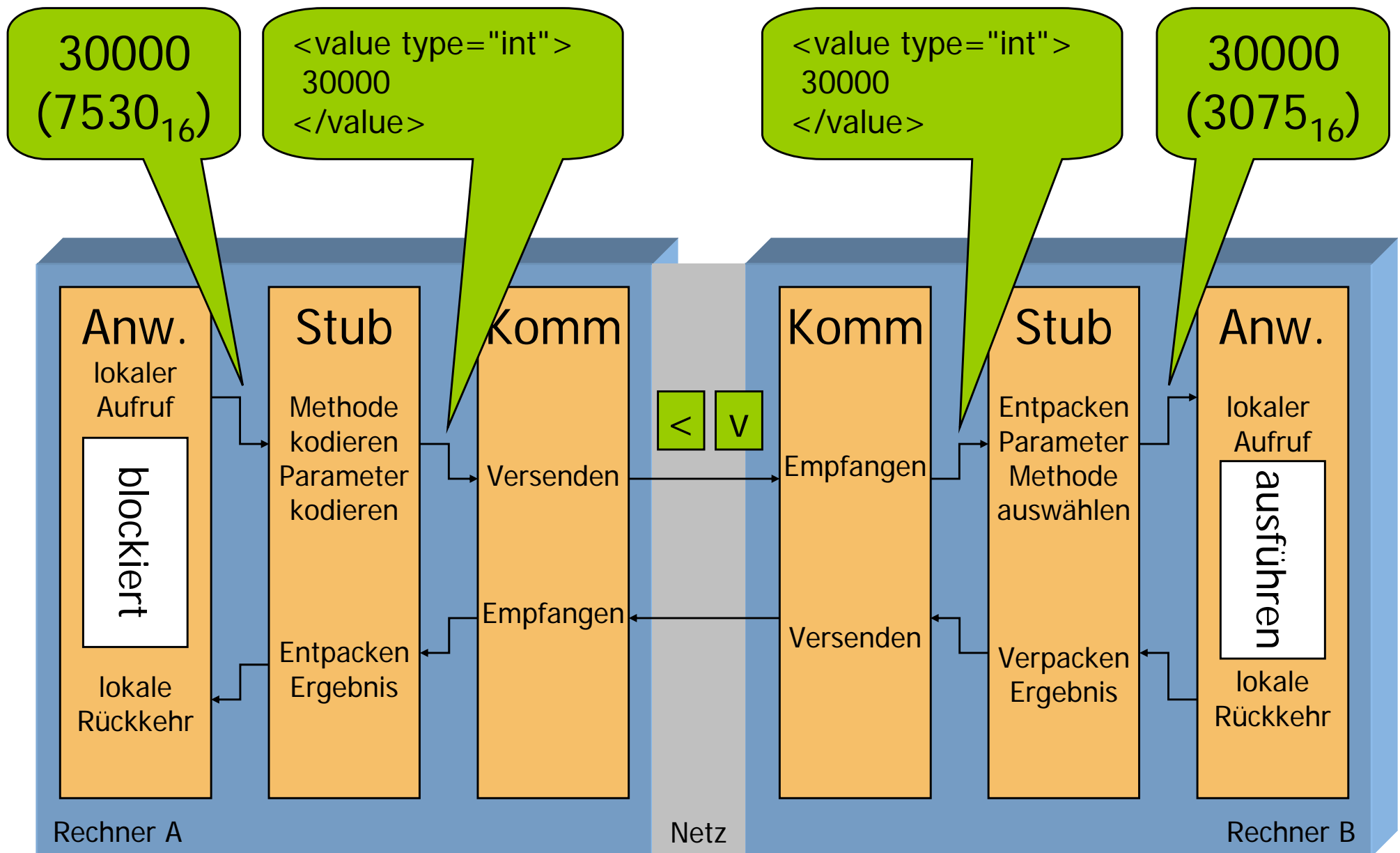


Web Service Basiskomponenten: SOAP

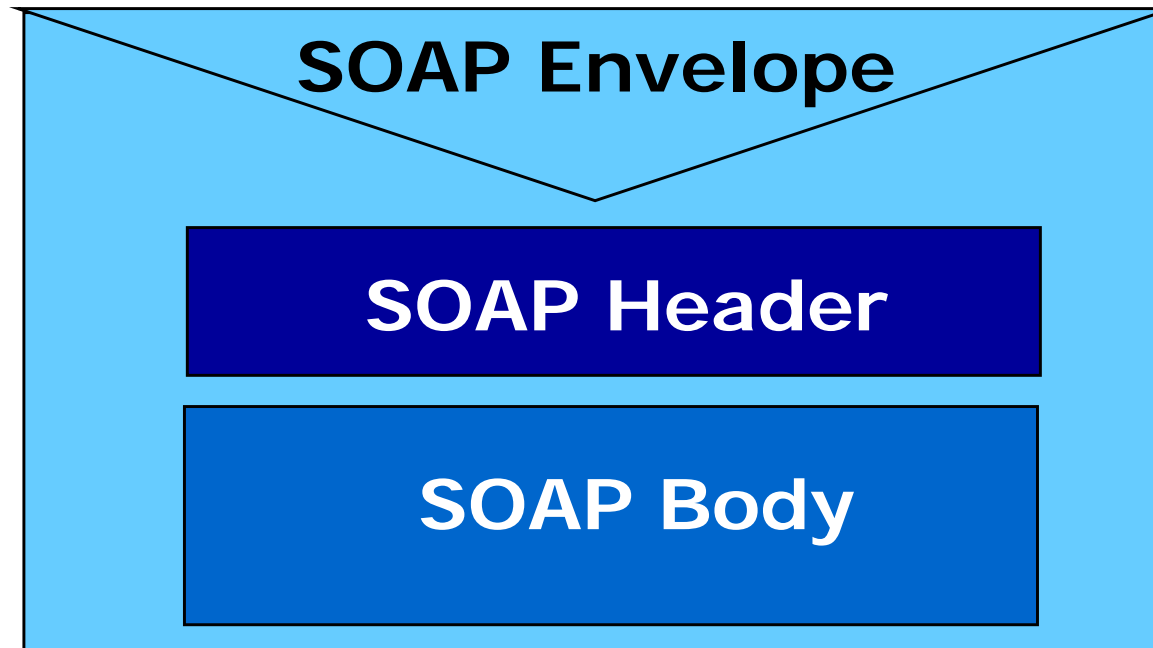
Austausch einer SOAP-Nachricht



Marshalling textuell



Aufbau einer SOAP-Nachricht



```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap/envelope/">  
  <!-- SOAP Header -->  
  <!-- SOAP Body -->  
</env:Envelope>
```

SOAP Version 1.2

SOAP Version 1.1:

<http://schemas.xmlsoap.org/soap/envelope/>

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 1.1: W3C Note von 2000
 - SOAP 1.2: W3C Recommendation von 2003
- seit SOAP 1.2: SOAP ≠ Simple Object Access



Kodierung von RPCs

- SOAP auch Nachrichtenformat für entfernte Prozeduraufrufe (RPCs)
- eigentlichen RPCs werden aber von Middleware realisiert
- SOAP selbst unterstützt nur Einweg-Kommunikation
- Anfrage-Antwort-Muster:
 1. SOAP mit HTTP übertragen
 - ⇒ auf Ebene des Transportprotokolls
 2. eindeutige Referenz im SOAP-Briefkopf
 - ⇒ auf Ebene von SOAP, dadurch unabhängig vom Transportprotokoll

- Anfrage (Request)
 - Methodenaufruf
 - Parameterübergabe
- Antwort (Answer)
 - fehlerfreie Bearbeitung
 - Ergebnisübergabe
- Fehlerfall (Fault)
 - Fehlerübergabe

- Anfrage enthält eindeutige Referenz („Mein Zeichen“), worauf anschließend verwiesen werden

```
<env:Header>
```

```
  <wsu:identifizier xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility/">
```

```
    URI als eindeutige Referenz
```

```
  </wsu:identifizier>
```

```
</env:Header>
```

Vorteile

- unabhängig von Transportprotokoll
- bel. komplexe Interaktionen möglich
- hilfreich bei Archivierung und Weiterleitung

Procedure(Param-1="val-1",...,Param-n="val-n")

```
<env:Envelope ...>
  <env:Body>
    <m:Procedure xmlns:m="URI">
      <m:Parameter-1>val-1</m:Parameter-1>
      ...
      <m:Parameter-n>val-n</m:Parameter-n>
    </m:Procedure>
  </env:Body>
</env:Envelope>
```

- Name der Prozedur: Kind-Element von Body
- Eingangsparemeter: Kind-Elemente der Prozedur
- Beachte: Reihenfolge der Parameter relevant!
- Beachte: grundsätzlich **Call-by-Value!**

Aufruf-Adresse

Wo soll die Prozedur aufgerufen werden (URI)?

- entweder außerhalb von SOAP im Transportprotokoll (z.B. HTTP) spezifiziert
- besser im SOAP-Briefkopf angeben:

```
<env:Header>
```

```
  <wsa:EndpointReference
```

```
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing">
```

```
  <wsa:Address>http://api.google.com/search/beta2</wsa:Address>
```

```
  <wsa:PortType>ns1:GoogleSearchPort</wsa:PortType>
```

```
</wsa:EndpointReference>
```

```
</env:Header>
```

```
public Parameter-i Procedure(...,Parameter-j,...)
```

```
<env:Envelope ...>
  <env:Body>
    <m: ProcedureResponse xmlns:m="URI"
      xmlns:rpc="http://www.w3.org/2003/05/soap-rpc">
      <rpc:result> m:Parameter-i</rpc:result>
      <m:Parameter-i>...</m:Parameter-i>
      ...
      <m:Parameter-j>...</m:Parameter-j>
    </m:ProcedureResponse>
  </env:Body>
</env:Envelope>
```

- **rpc:result**: ausgezeichnetes Ergebnis (optional)
- Namensraum **.../soap-rpc** Teil der SOAP-Spezifikation

public Parameter-i Procedure(...,Parameter-j,...)

```
<env:Envelope ...>
  <env:Body>
    <m:ProcedureResponse xmlns:m="URI"
      xmlns:rpc="http://www.w3.org/2003/05/soap-rpc">
      <rpc:result> m:Parameter-i</rpc:result>
      <m:Parameter-i>...</m:Parameter-i>
      ...
      <m:Parameter-j>...</m:Parameter-j>
    </m:ProcedureResponse>
  </env:Body>
</env:Envelope>
```

- *Wahl von ProcedureResponse* beliebig
- Kind-Elemente von *ProcedureResponse*: Rückgabewerte
- In-Out-Parameter = erscheinen im Aufruf & Antwort

```
<env:Envelope ...>  
  <env:Body>  
    <env:Fault>  
      <env:Code>  
        <env:Value>env:Sender</env:Value>  
      </env:Code>  
      <env:Reason>  
        <env:Text xml:lang="en-US">Processing error</env:Text>  
        <env:Text xml:lang="de">Verarbeitungsfehler</env:Text>  
      </env:Reason>  
    </env:Fault>  
  </env:Body>  
</env:Envelope>
```

- Code und Reason obligatorisch
- **Code**: für maschinelle Verarbeitung
- **Reason**: zusätzliche Information, nicht für maschinelle Verarbeitung

```
<env:Envelope ...>
  <env:Body>
    <env:Fault>
      <env:Code>
        <env:Value>env:Sender</env:Value>
      </env:Code>
      <env:Reason>
        <env:Text xml:lang="en-US">Processing
error</env:Text>
        <env:Text
xml:lang="de">Verarbeitungsfehler</env:Text>
      </env:Reason>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

- Value obligatorisch
- **env:Sender**: Anfrage nicht korrekt, nochmalige korrigierte Anfrage erwartet

```
<env:Envelope ... xmlns:rpc="http://www.w3.org/2003/05/soap-rpc">
  <env:Body>
    <env:Fault>
      <env:Code>
        <env:Value>env:Sender</env:Value>
        <env:Subcode>
          <env:Value>rpc:BadArguments</env:Value>
        </env:Subcode>
      </env:Code>
      <env:Reason>...</env:Reason>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

- Subcode optional
- Subcode genauso strukturiert, wie Code: Value (obligatorisch) + Subcode (optional)
- **rpc:BadArguments**: standardisierter Fehlercode

```
<env:Envelope ...>
  <env:Body>
    <env:Fault>
      <env:Code>
        <env:Value>env:Sender</env:Value>
      </env:Code>
      <env:Reason>
        <env:Text xml:lang="en-US">Processing error</env:Text>
        <env:Text xml:lang="de">Verarbeitungsfehler</env:Text>
      </env:Reason>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

- mindestens ein Text-Element
- xml:lang-Attribut obligatorisch

```
<?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
- Beachte: Web Service-Aufruf kann als URL kodiert werden (REST)

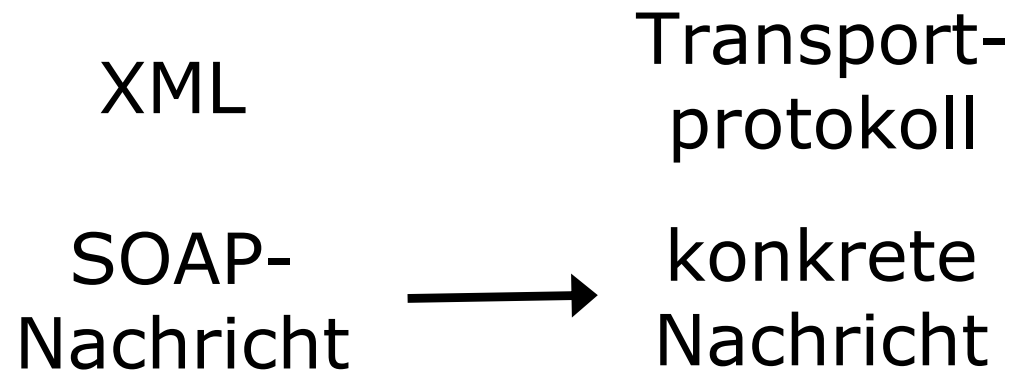
```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="..." xmlns:xsi="...">
  <env:Body>
    <ns1:doGoogleSearchResponse xmlns:ns1="urn:GoogleSearch" ...>
      <return xsi:type="ns1:GoogleSearchResult">
        ...
      </return>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

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

- 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 (Messaging) möglich



Übertragung von SOAP-Nachrichten



- Wie werden SOAP-Nachrichten mit bestimmten Transportprotokoll übertragen?
- Wie SOAP-Nachrichten serialisieren?
- z.B. für HTTP GET nicht trivial
- SOAP-Spezifikation schreibt nicht vor, **wie** Protokoll-Bindung spezifiziert wird

- konkrete Nachricht meist XML, kann aber auch beliebig anderes Format sein:
z.B. komprimiertes Binärformat
- einzige Bedingung:
Serialisierung ohne Informationsverlust
- Serialisierung s muss also symmetrisch sein:
 $s^{-1}(s(N)) = N$, für alle SOAP-Nachrichten N

- HTTP-Binding bisher als einzige Protokoll-Bindung für SOAP standardisiert
- zwei unterschiedliche HTTP-Bindungen:
 - HTTP-POST
 - HTTP-GET

HTTP GET

- URL → Antwort
 - Parameter können in URL kodiert werden, z.B.:
 - `http://google.com/doGoogleSearch?q=Beginning+XML`
- = Aufruf `doGoogleSearch(q="Beginning XML")`

HTTP POST

- URL + Datenanhang → Antwort

SOAP über HTTP POST: Anfrage

```
POST /search/beta2/doGoogleSearch HTTP/1.1
Host: api.google.com
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn
```

HTTP Header

```
<?xml version='1.0' encoding='UTF-8'?>
<env:Envelope ...>
  <env:Body>
    <doGoogleSearch xmlns="urn:GoogleSearch">
      <key
xsi:type="xsd:string">3289754870548097</key>
      <q xsi:type="xsd:string">Eine Anfrage</q>
      ...
    </doGoogleSearch>
  </env:Body>
</env:Envelope>
```

SOAP-
Nachricht

SOAP über HTTP POST: Antwort

HTTP/1.1 200 OK

Content-Type: application/soap+xml; charset="utf-8"

Content-Length: nnnn

HTTP Header

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

```
<env:Envelope ...>
```

```
<env:Body>
```

```
<ns1:doGoogleSearchResponse xmlns:ns1="urn:GoogleSearch"
```

```
env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
```

```
<return xsi:type="ns1:GoogleSearchResult">...</return>
```

```
</ns1:doGoogleSearchResponse>
```

```
</env:Body>
```

```
</env:Envelope>
```

SOAP-Response

SOAP über HTTP GET

**GET /search/beta2/doGoogleSearch?q=Beginning+XML
HTTP/1.1**

Host: api.google.com

Accept: application/soap+xml

- ruft doGoogleSearch(q="Beginning XML") auf
- gesamte SOAP-Nachricht als URL kodiert
- Antwort wie bei HTTP POST
- Amazon bietet HTTP-GET-Schnittstelle an, Google jedoch nicht
- sehr beliebt weil leichtgewichtig

- **REST-Architektur** des WWW (Fielding 2000):
jede Web-Ressource soll eindeutig über eine URI identifiziert werden
- Beispiel: online gebuchte Reise = Web-Ressource
- gebuchte Reise sollte daher auch über eine URI eindeutig identifiziert werden

HTTP GET

⇒ entspricht REST-Grundsatz

- würde z.B.
travel.com/Reservations/itinerary?reservationCode=FT35ZBQ
anfragen
⇒ URL identifiziert eindeutig gebuchte Reise

HTTP POST

- würde z.B. ⇒ widerspricht REST-Grundsatz
travel.com/Reservations/
anfragen mit SOAP-RPC als Datenhang:
itinerary(reservationCode="FT35ZBQ")
⇒ URL identifiziert nicht gebuchte Reise

- SOAP-Spezifikation empfiehlt HTTP GET, wenn Parameter Web-Ressourcen identifizieren
- ⇒ Übereinstimmung mit REST-Grundsatz
- Problem: Wie komplexe Parameter als URI kodieren?
- Beispiel: reservationCode könnte aus Bezeichner + Datum bestehen
- so kodieren?
`travel.com/Reservations/itinerary?reservationCode=FT35ZBQ+22/6/2005`
- oder so?
`travel.com/Reservations/itinerary?reservationId=FT35ZBQ&reservationDate=22/6/2005`

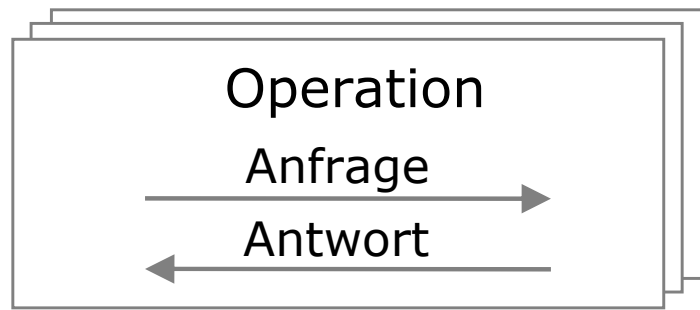


Web Service Basiskomponenten: WSDL

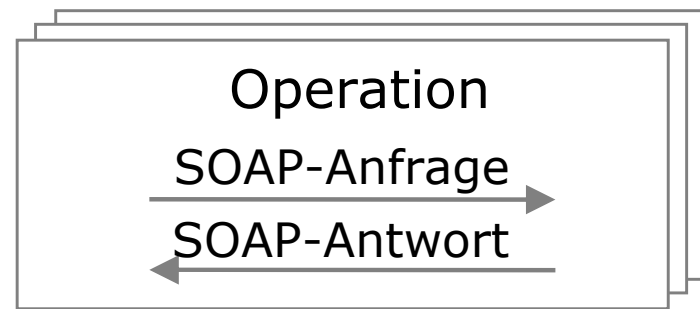
Web Services Description Language

Servicebeschreibung

abstrakte Schnittstelle

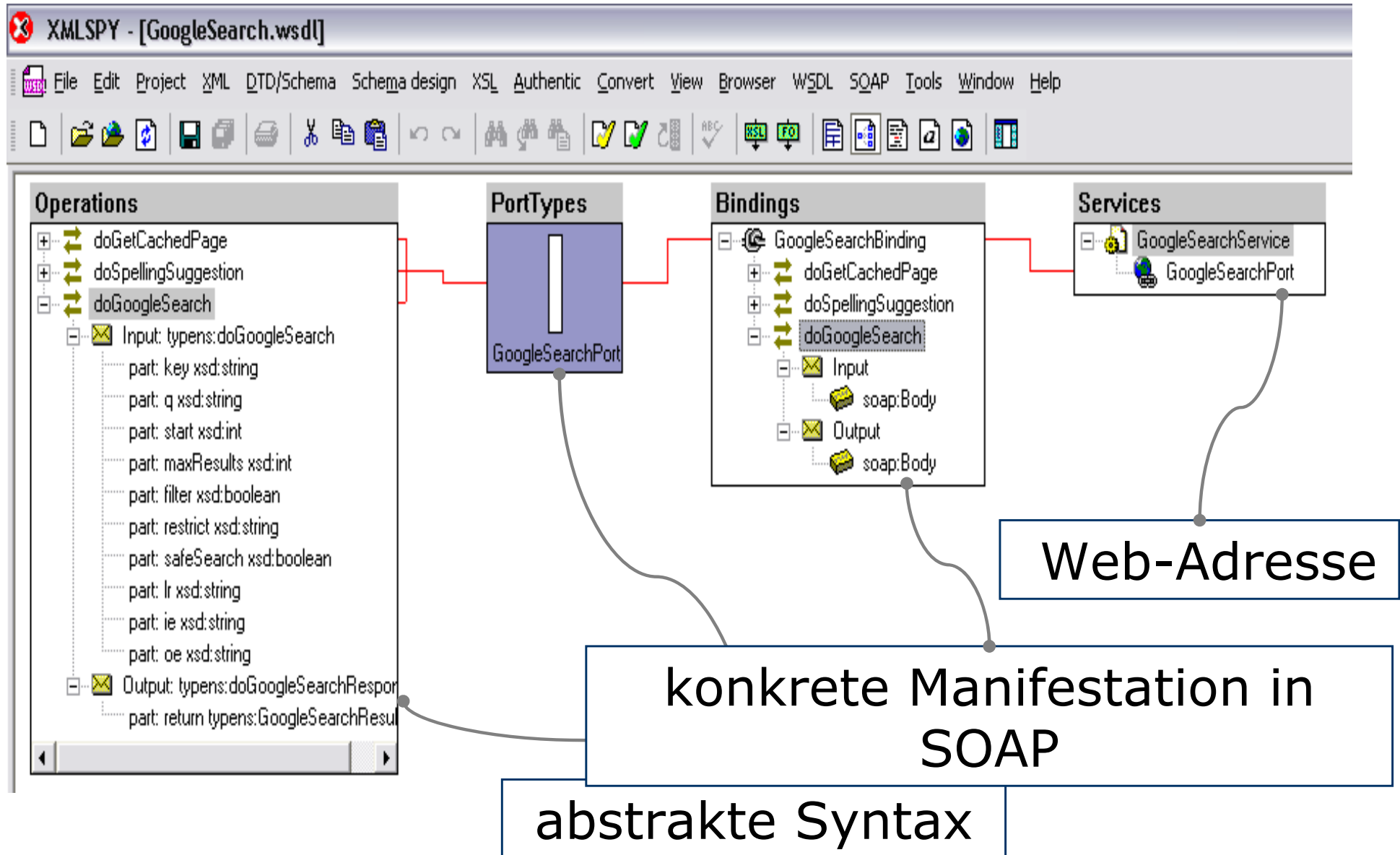


konkrete Schnittstelle



Web-Adressen (End Points)

- beschreibt Interface (IDL)
 - XML-basierter Standard
 - W3C Note von 2001
- abstrakte Schnittstelle (port type)**
- Schnittstelle unabhängig von Nachrichtenformaten
 - Beschreibung mit XML-Schema
- konkrete Schnittstelle (binding)**
- Abbildung der abstrakten Schnittstelle auf unterstützte Nachrichtenformate



Eigenschaften von WSDL

- beschreibt Schnittstelle(n) 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.



Web Service Basiskomponenten: UDDI

- Universal (Service) Description, Discovery, and Integration
- entwickelt seit Herbst 2000 von Ariba, Microsoft & IBM
- beschreibt einen Verzeichnisdienst für Web Services
- ein SOAP basierter Dienst

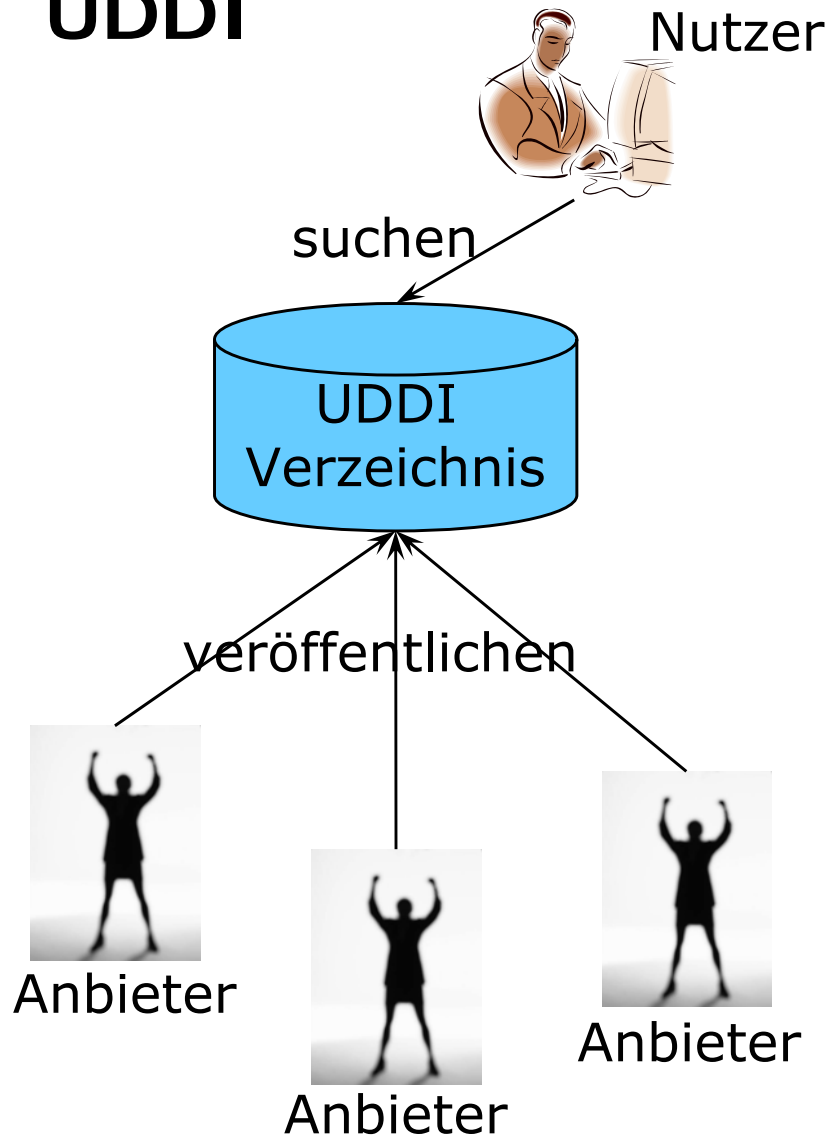
UDDI Komponenten

- UDDI-XML-Schema
 - Business-Entity
 - Business-Service
 - Binding-Template
 - Technisch Modelle (TModel)

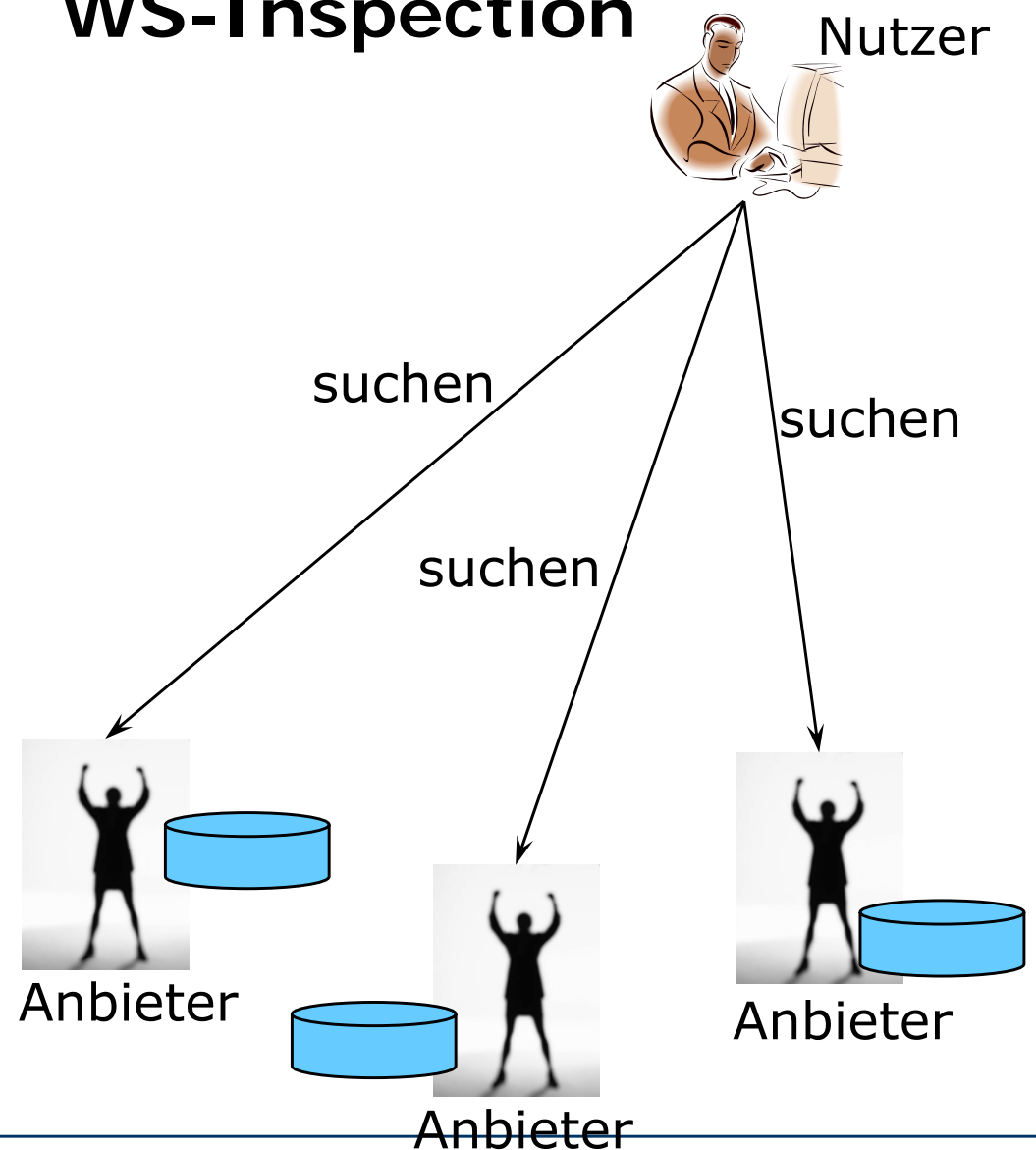
- UDDI-API
 - Anwendungsschnittstelle in Form von Web Services

UDDI vs. WS-Inspection

UDDI



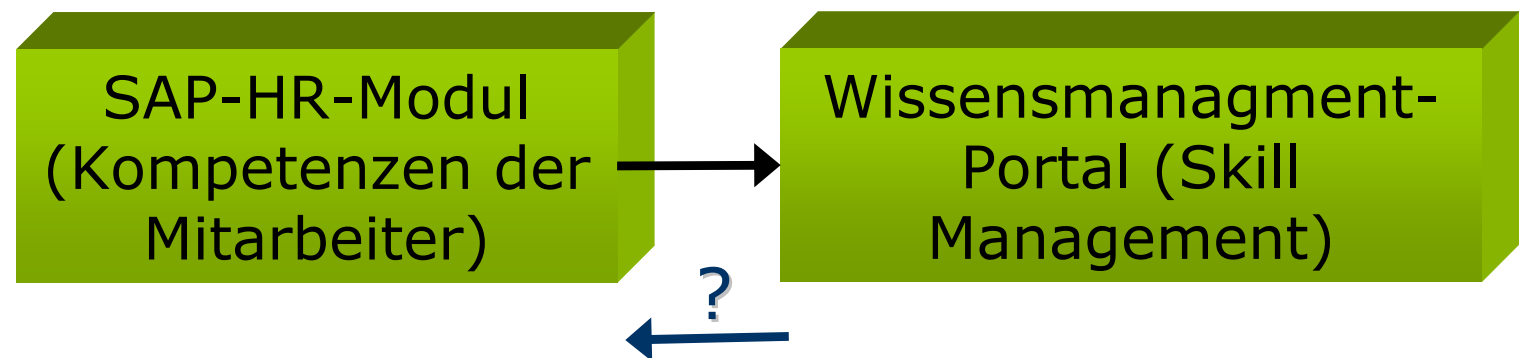
WS-Inspection





Enterprise Application Integration

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



unternehmensintern

- Beispiel Mercedes-Benz-Werk
 - mehr als 200 EDV-Systeme
 - sehr gut vernetzt (LAN)
 - Systeme also prinzipiell integrierbar
- Neue Systeme müssen Alt-Systeme integrieren.

unternehmensübergreifend

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

Systemintegration bindet

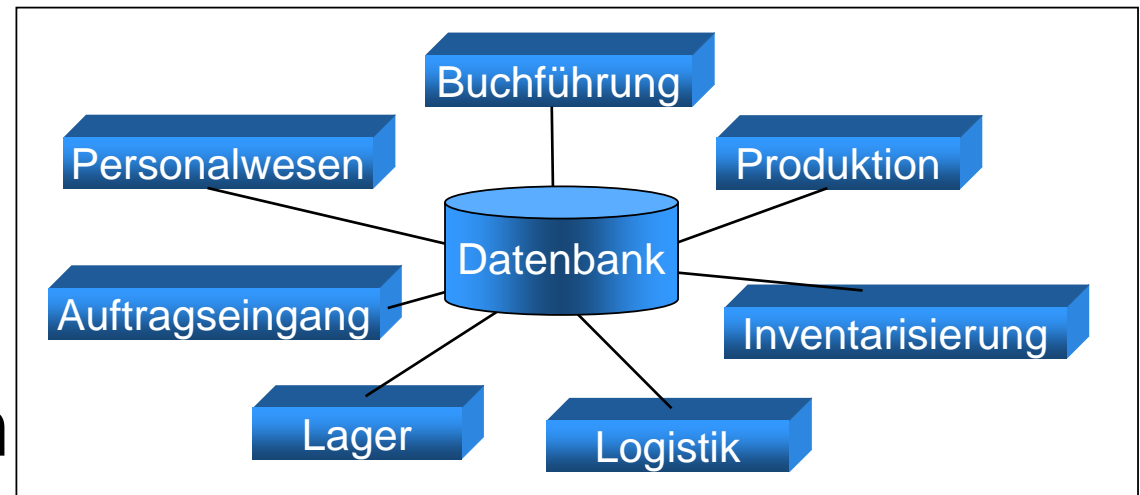
- **35%** der IT-Personal-Ressourcen eines Unternehmens (Forrester Research, 2002)
- **65%** der Arbeitszeit eines Programmierers (Gartner)

Enterprise Resource Planning

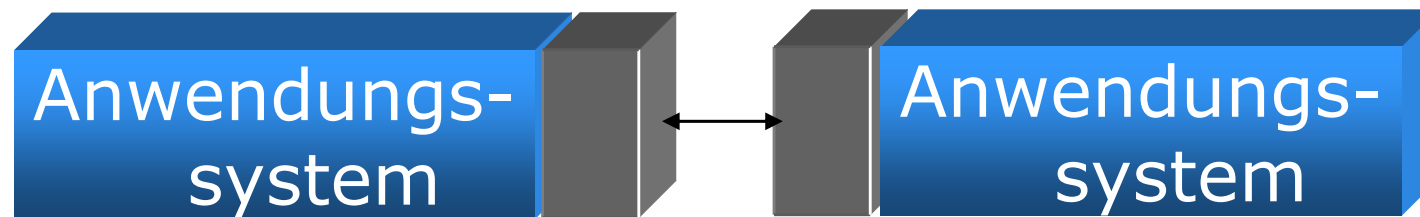
E-Commerce

?

- Beispiel: SAP/R3
- großer Fortschritt für Datenintegration
- sehr teuer:
 - > 53.000 \$ (TCO) pro Nutzer für ersten 2 Jahre (Meta Group, 2002)
- viele Datensilos durch ein großes Datensilo ersetzt:
- schwierige Integration externer Systeme



- Anwendungssysteme durch standardisierte Schnittstelle erweitern



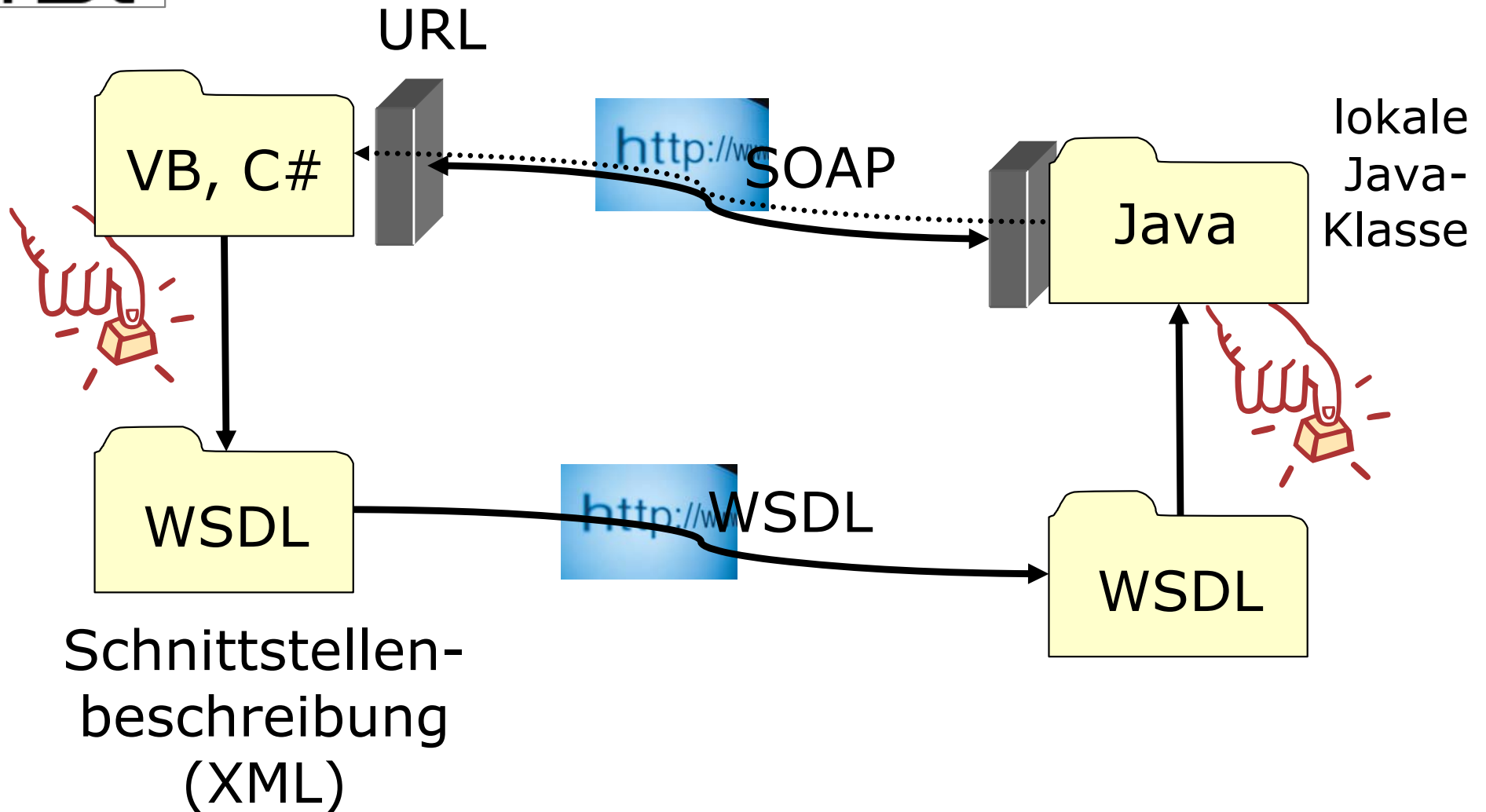
- Schnittstelle muss allgemein akzeptiert sein
- bei Web Services ist dies der Fall
 - ✓ SOAP
 - ✓ WSDL
 - ✓ Internet-Protokolle
- bei n Systemen:
statt n^2 Schnittstellen nur n Erweiterungen!

Systemintegration per Knopfdruck!

MS XP



Unix



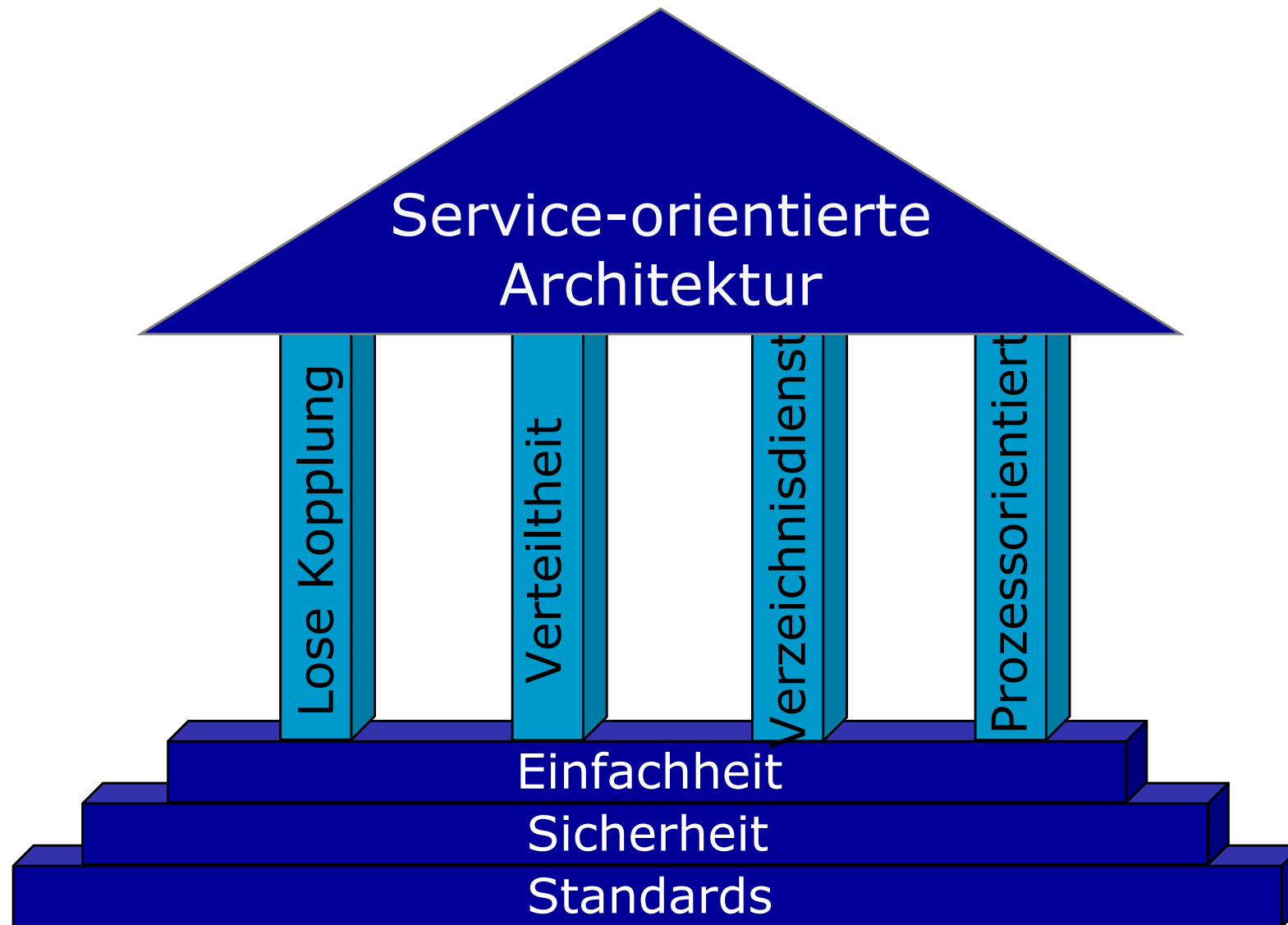


Dienstorientierte Architektur

- engl. **service-oriented architecture**, kurz **SOA**
- statt Anwendungen isoliert zu entwickeln, nur um sie später zu integrieren:
- neue Anwendungen von Anfang an auf existierenden Web Services aufbauen
- neue Anwendung wiederum als Web Service anbieten

*„... eine Systemarchitektur, die vielfältige, verschiedene und eventuell inkompatible Methoden oder Applikationen als wiederverwendbare und offen zugreifbare Dienste repräsentiert und dadurch eine plattform- und sprachunabhängige Nutzung und Wiederverwendung ermöglicht.“ **

*Quelle: „ Service-orientierte Architekturen mit Web Services: Konzepte – Standards – Praxis“, W. Dostal, M. Jeckle, I. Melzer, B. Zengler; Spektrum Akademischer Verlag, 2005



- konsequente Realisierung einer dienstorientierten Architektur:

If you hit the Amazon.com gateway page, the application calls more than 100 services to collect data and construct the page for you
(Werner Vogels, CTO von Amazon)

Beispiele für (interne) Web Services:

- Webseite generieren
- Kaufempfehlungen generieren
- „Käufer, die diesen Artikel gekauft haben, haben auch folgende Artikel gekauft...“
- Angebote anderer Anbieter

Vorteile aus Sicht von Amazon

- komplexe Anwendung aus relativ einfachen Web Services aufbauen
- aus komplexer Anwendung wiederum einfachen Web Service machen
- verteilte, skalierbare, robuste Architektur
- bestimmte Gruppe für Entwicklung und Betrieb eines Web Services verantwortlich

Battle-Tested Web Services

Amazon has spent 11 years and over \$2 billion building the infrastructure, technical knowledge, and operational excellence to operate a world class web-scale computing platform. Amazon Web Services has now released a variety of web services (programmatic access to its open APIs) that enable developers to leverage Amazon's data and robust infrastructure, easily and inexpensively. These fundamental services allow external developers and businesses to build their web applications in a reliable, scalable, and cost-effective manner.

Web-Scale Computing

Amazon Web Services (AWS) provides customers the opportunity to replace existing infrastructure and scale up or down based on resource demands. This flexibility allows you to run your business at "web-scale"--uninhibited by growth in the number of geographic markets. On average, businesses spend 70% of their time building and maintaining and worrying about infrastructure, and 30% of their time focused on the ideas that will propel their business forward. Web-scale computing is helping to invert the 70/30 ratio, enabling you to spend your energy creating the difference that will make your business successful.

Web-scale computing characteristics:

- Elastic capacity both up and down
- Fast response time
- 24/7 availability
- Rock solid reliability

Only Pay For What You Use

With AWS offerings, you only pay for what you use. You dynamically scale your system up and down, depending on your immediate requirements and only pay for resources when you need them. This means there are no upfront costs, so your developers can get started using web services without huge investment. Developers don't need to give up equity or incur huge capital expenses, because costs scale along with usage.

Focus on the Idea

Free up your most valuable resource: time. Don't spend time worrying about scaling data centers, buying hardware, negotiating lease contracts, dealing with downed servers, or backing-up customer data. Amazon Web Services takes care of those issues and does the "heavy lifting". Shifting the focus from managing infrastructure to driving your business allows you to concentrate on what matters. This business approach also levels the playing field so everyone can now compete on ideas, not resources.

Amazons Web Services

Amazon Associates Web Service (A2S)

The Amazon Associates Web Service (A2S) exposes Amazon's product data and e-commerce functionality. This allows developers, web site owners and merchants to leverage the data and functionality that Amazon uses to power its own e-commerce business. A2S 4.0 makes it extremely easy for developers to build rich, highly effective web sites and applications.

Amazon Elastic Compute Cloud (Amazon EC2) - Limited Beta

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud. It is designed to make web-scale computing easier for developers. Just as Amazon Simple Storage Service (Amazon S3) enables storage in the cloud, Amazon EC2 enables "compute" in the cloud. Amazon EC2's simple web service interface allows you to obtain and configure capacity with minimal friction. It provides you with complete control of your computing resources and lets you run on Amazon's proven computing environment. Amazon EC2 reduces the time required to obtain and boot new server instances to minutes, allowing you to quickly scale capacity, both up and down, as your computing requirements change. Amazon EC2 changes the economics of computing by allowing you to pay only for capacity that you actually use.

Amazon Flexible Payments Service (Amazon FPS) - Limited Beta

Amazon Flexible Payments Service (Amazon FPS) is the first payments service designed from the ground up specifically for developers. The set of web services APIs allows the movement of money between any two entities, humans or computers. It is built on top of Amazon's reliable and scalable payment infrastructure.

Amazon Mechanical Turk (Beta)

Today, humans still significantly outperform the most powerful computers at completing such simple tasks as identifying objects in photographs - something children can do even before they learn to speak. However, when we think of interfaces between human beings and computers, we usually assume that the human being is the one requesting that a task be completed, and the computer is completing the task and providing the results. What if this process were reversed and a computer program could ask a human being to perform a task and return the results? What if it could coordinate many human beings to perform a task?

Amazon Mechanical Turk does this, providing a web services API for computers to integrate Artificial Intelligence directly into their processing.

Amazon SimpleDB - Limited Beta

Amazon SimpleDB is a web service for running queries on structured data in real time. This service works in close conjunction with Amazon Simple Storage Service (Amazon S3) and Amazon Elastic Compute Cloud (Amazon EC2), collectively providing the ability to store, process and query data sets in the cloud. These services are designed to make web-scale computing easier and more cost-effective for developers.

Amazon Simple Storage Service (Amazon S3)

Amazon S3 is storage for the Internet. It is designed to make web-scale computing easier for developers. Amazon S3 provides a simple web services interface that can be used to store and retrieve any amount of data, at any time, from anywhere on the web. It gives any developer access to the same highly scalable, reliable, fast, inexpensive data storage infrastructure that Amazon uses to run its own global network of web sites.

Amazon Simple Queue Service (Amazon SQS)

Amazon Simple Queue Service (Amazon SQS) offers a reliable, highly scalable hosted queue for storing messages as they travel between computers. By using Amazon SQS, developers can simply move data between distributed application components performing different tasks, without losing messages or requiring each component to be always available.

ohne dass Nutzer des Web Services es bemerkt,
kann

- + neue Version freigeschaltet werden
- + bei Ausfall ein Web Service durch einen anderen Web Service mit gleicher Schnittstelle ersetzt werden
- + Lastverteilung durchgeführt werden

allerdings

- Vertrauen nötig, dass Web-Service-Anbieter WSDL-Beschreibung im Sinne des Nutzers realisiert