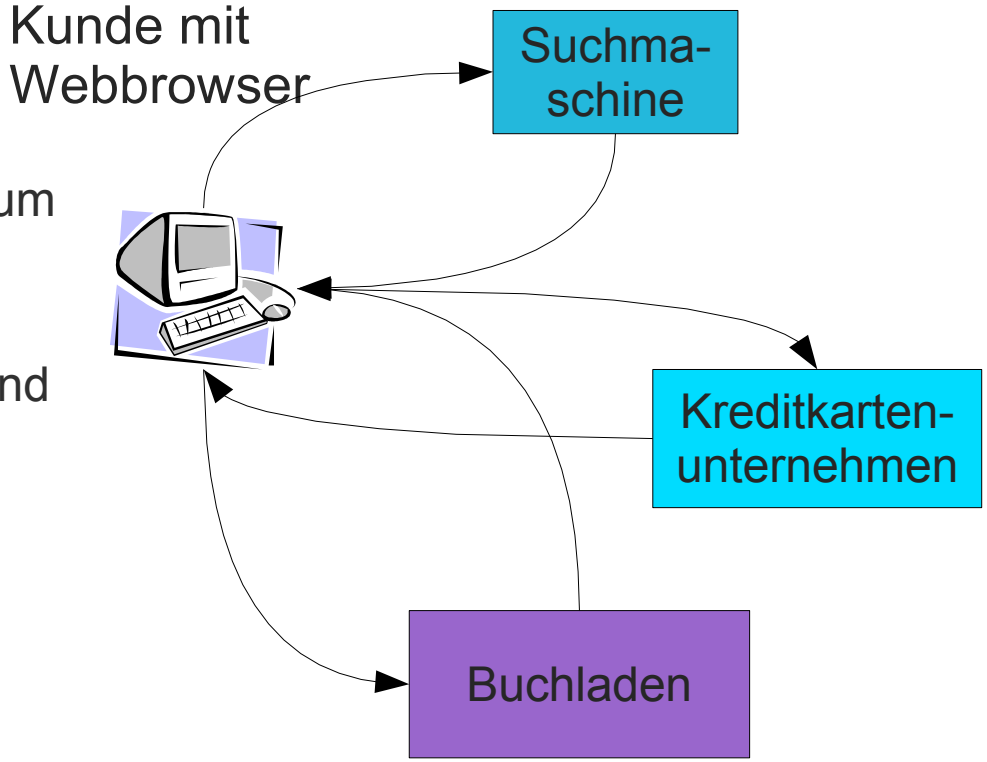
Webservices

Szenario

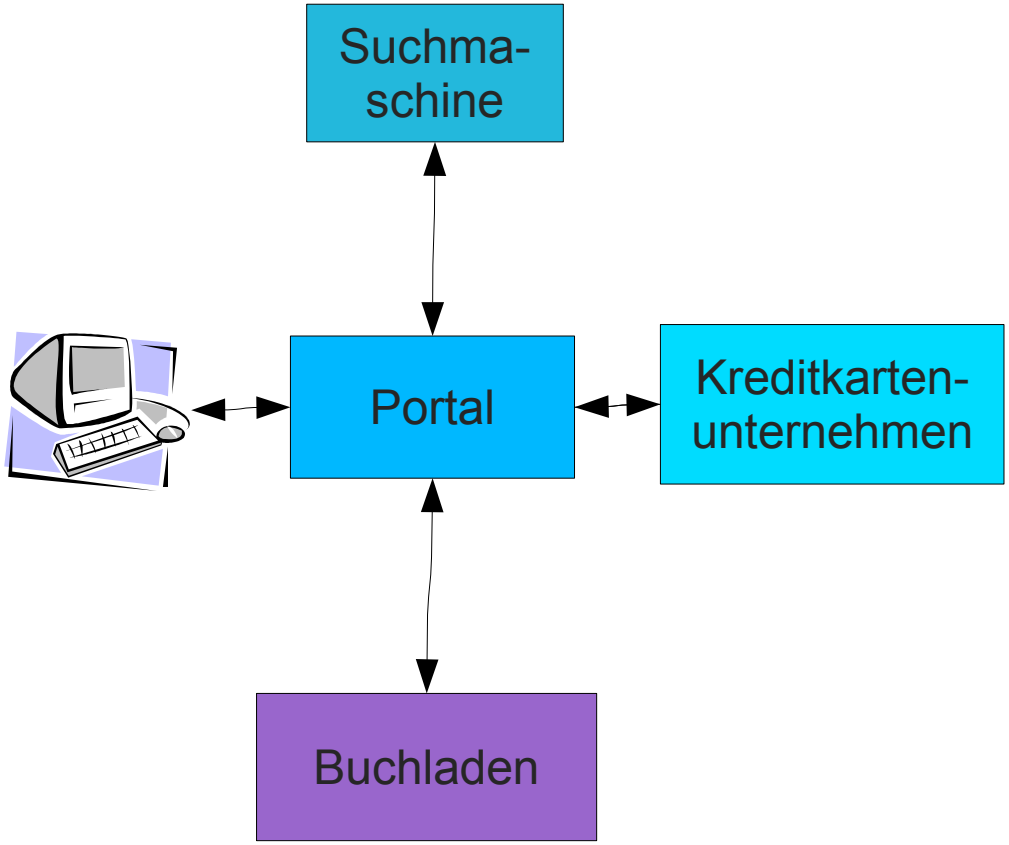
- Was bisher geschah
  - Kunde nutzt verschiedene Dienste, um den Prozess des Warenkaufs durchzuführen.
  - Kunde interpretiert die Ergebnisse und zieht daraus Schlüsse für den nächsten Schritt
  - Dienstleister sind voneinander getrennt



Nutzerzentrierte Anwendungskomposition

Szenario

- Baukastenprinzip für Dienste
  - Kunde nutzt einen Dienst, der verschiedene andere Dienste anspricht
  - Modellierung von Geschäftsprozessen; Kunde muss die Schritte nicht selbst ausführen
  - Dienstleister arbeiten zusammen



Webbasierte Anwendungskomposition

Webservices

- Problem
  - Informationen liegen im Web in Form von HTML-Dokumenten vor
  - Diese sind auf die Interpretation durch einen (intelligenten) Nutzer zugeschnitten.
  - Seiten können nicht ohne weiteres automatisch ausgewertet werden (z.B. die Trefferliste von Google)
- Ziel
  - Die bislang nur für Benutzer lesbaren Informationen auf dem Web auch für Programme zugänglich zu machen
  - Dabei soll aber die Art der Datenabfrage beibehalten werden
    - Webserver, HTTP

Webservices

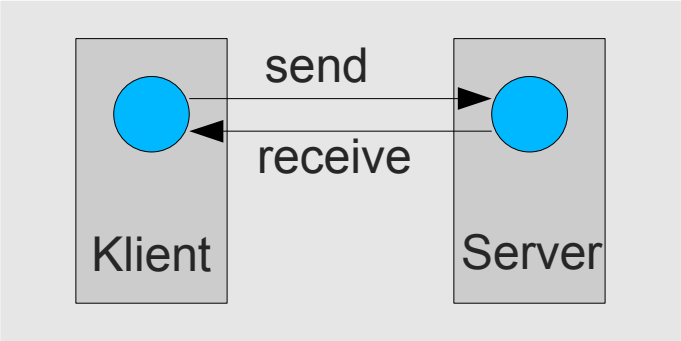
- Ansatz
  - Einheitliches Datenformat, frei von Darstellungsspezifikationen
    - Natürlich XML! (wie erwartet)
  - Aufrufschnittstelle definieren
    - sozusagen die bekannten Muster (RPC usw.) auf das Web abbilden
- Einsatzgebiete vielschichtig
  - geschäftsorientiert: Kreditkartenprüfung
  - kundenorientiert: Aktienkurse
  - systemorientiert: Benutzerauthentifikation
  - ...

→ ...alles Einsatzgebiete, in denen Inhalte und nicht die Darstellung gefragt ist.

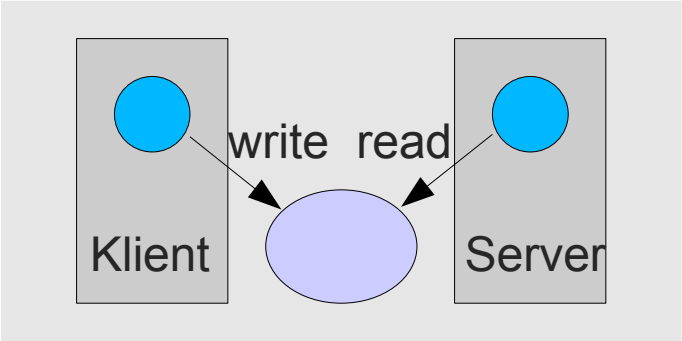
Interaktionsmodelle

- Kurzer Überblick über allgemeine Modelle

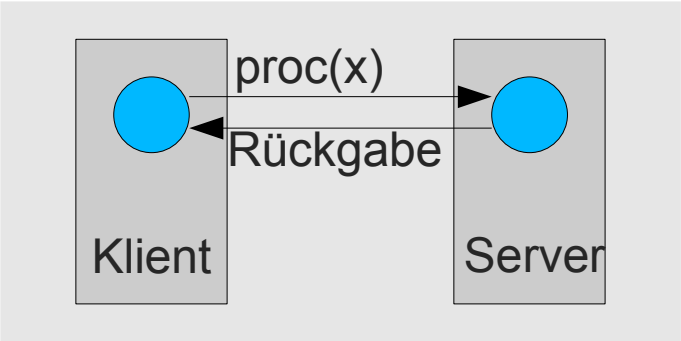
Nachrichten (Message passing)



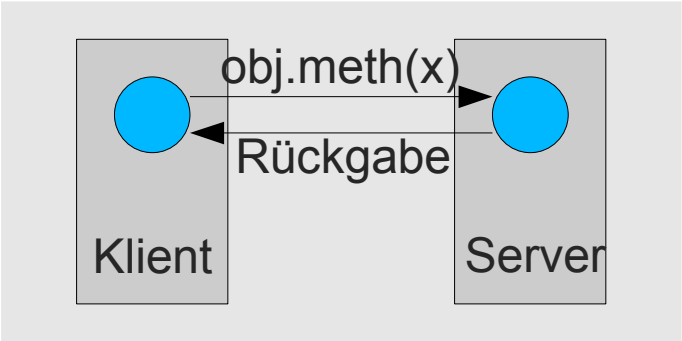
Virtueller gemeinsamer Speicher



Entfernte Prozeduren (RPC)



Verteilte Objekte



Objektaufrufprotokolle

- CORBA: IIOP
  - Binäres Protokoll
  - Komplex, meist nur mit kommerziellen Produkten
- RMI: Java Remote Method Invocation
  - binär
  - erfordert Java auf beiden Seiten, also nicht sprachunabhängig
- DCOM: Distributed Component Object Model
  - binär
  - hauptsächlich Windows-Welt (nicht plattformunabhängig)

All diese Protokolle wären nutzbar für die Kommunikation mit Diensten im Internet, aber warum ...

... muss es denn HTTP sein?

Innerhalb von zwei Jahren wird IIOP das HTTP im Internet ersetzt haben
Marc Andreesen, 1996

IIOP ist als Kommunikations-protokoll zwischen Klient und Server im Internet nicht zu empfehlen.
John Dawes, Netscape, 1997

Warum nicht IIOP oder DCOM?

- DCOM
  - ist stark verbindungsorientiert
    - Aufwand zum Betrieb von Sitzungen
  - nicht plattformunabhängig
    - Windows oder einige kommerzielle Unixe
- DCOM und IIOP sind sehr aufwändige Protokolle
  - brauchen dicke Laufzeitunterstützung
  - Administrationsaufwand
  - schwer zu portieren
- DCOM und IIOP werden meist an **Firewalls** abgeblockt

Schwer wiegendes Problem: Viele Firewalls blockieren fast alles, was nicht HTTP ist.

Warum HTTP?

- Hypertext Transfer Protocol
  - das meistverbreitete Protokoll im Internet
  - verfügbar für alle Plattformen
  - einfach aufgebaut
  - Sitzungssteuerung extern, aber einfach zu realisieren
  - Sicherheitsmechanismen
  - von Firewalls durchgelassen

Warum XML?

- Extensible Markup Language
  - textbasiert, selbsterklärende Repräsentation
  - einfache Handhabung
  - plattformunabhängig
  - unterstützt von wesentlichen Technologietreibern
  - Werkzeuge und APIs (wie DOM) verfügbar
  - anpassungsfähig (Transformation und Präsentation via XSL)

XML wäscht nicht Ihren Hund!

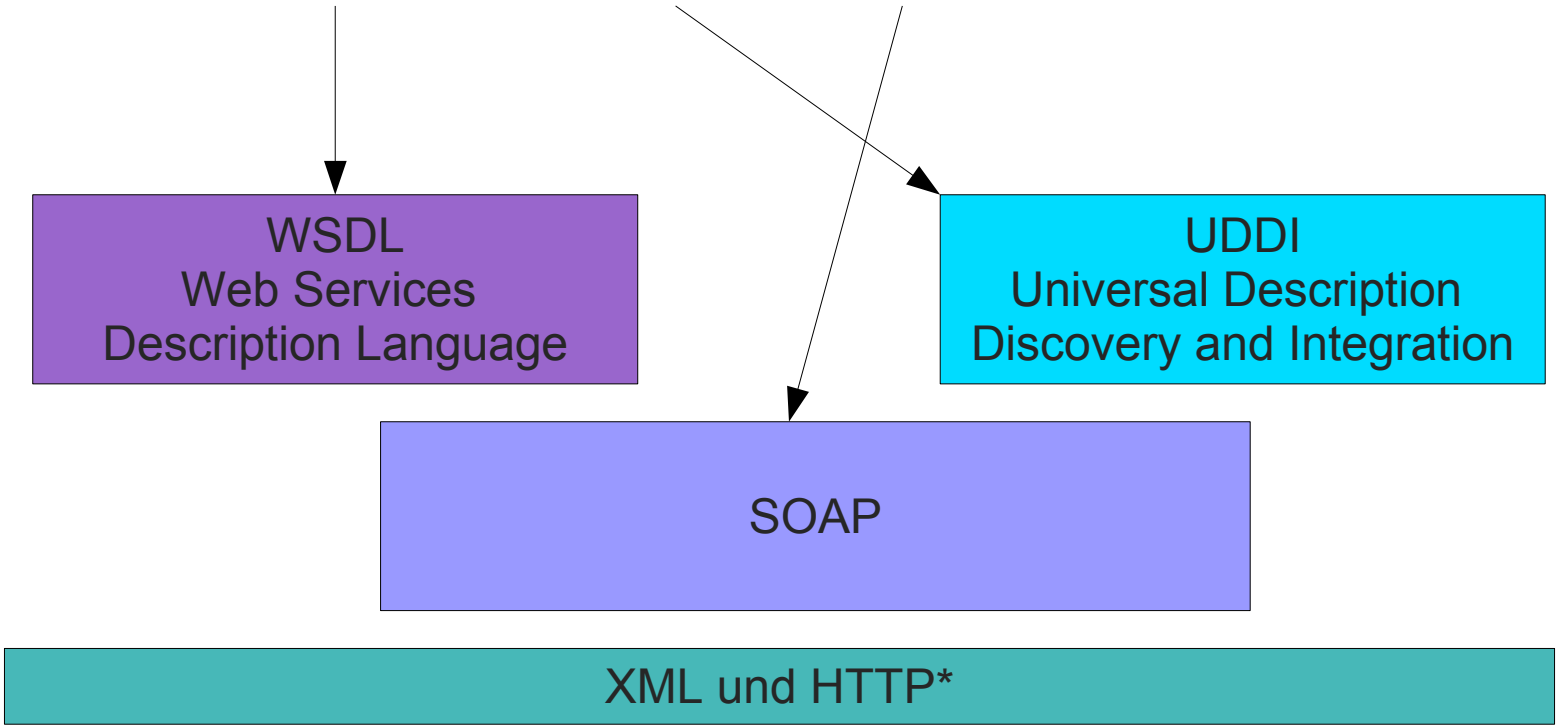
Aber zumindest ist es möglich, verhältnismäßig robuste Protokolle zu entwerfen. Man kann sich dafür verstärkt inhaltlichen Problemen widmen (welche immer noch bestehen).

Objektaufrufprotokolle

- SOAP
  - ehemals „Simple Object Access Protocol“, jetzt aber keine Abkürzung mehr
  - einfaches, leichtgewichtiges Protokoll zum Austausch von strukturierten und typisierten Informationen auf dem Web
  - Protokoll: HTTP, SMTP, ...
  - Format: XML
- Designziel: KISS (keep it simple, stupid!)
  - einfach implementiert
  - Minimum an vorgegebener Funktionalität
  - basierend auf akzeptierten Standards (HTTP / XML)

Webservices: WSDL, UDDI, SOAP

Ein Webservice ist eine eigenständige modulare Anwendungskomponente, welche im Web veröffentlicht, lokalisiert und aufgerufen werden kann.



* HTTP ist eine von vielen möglichen Bindungen

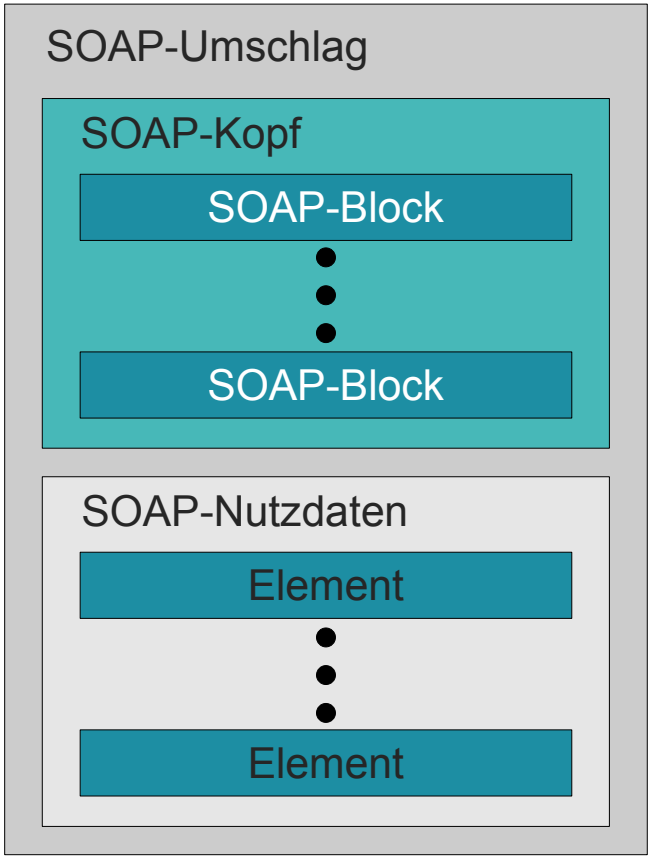
SOAP: Drei Sichtweisen

- SOAP ist ein Objektaufrufprotokoll
  - Anfragen beinhalten in- und inout-Parameter
  - Antworten können inout- und out-Parameter beinhalten
- SOAP ist ein Nachrichtenaustauschprotokoll
  - Anfragen beinhalten ein serialisiertes Anfrageobjekt
  - Antworten beinhalten ein serialisiertes Antwortobjekt
- SOAP ähnelt einem entfernten XSLT
  - Anfrage beinhaltet ein XML-Dokument
  - Server antwortet mit einer transformierten Version

Keine dieser Sichtweisen ist vom Standard vorgeschrieben

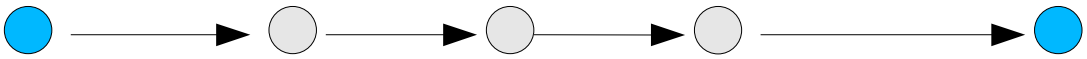
SOAP-Nachricht

- Aufbau einer SOAP-Nachricht
  - Umschlag („envelope“) enthält alle Teile der Nachricht
  - Kopf („header“) besteht aus einem oder mehreren Blöcken („header block“)
  - Jeder Kopfblock kann sich an einen anderen Empfänger (genauer: Rolle) in der Versendekette richten
  - Nutzdaten („body“) sind Ende-zu-Ende-Daten



SOAP-Kommunikationsmodell

- SOAP-Knoten
  - Anfangsknoten, Zwischenknoten (intermediary), Endknoten
- SOAP-Rollen
  - Knoten nehmen eine oder mehrere Rollen an, wenn sie eine Nachricht erhalten
  - Vordefinierte Rollen
    - none: Kein Knoten darf in dieser Rolle agieren
    - next: Jeder Zwischenknoten und der Endknoten müssen in dieser Rolle agieren
    - ultimateReceiver: Der endgültige Empfänger muss in dieser Rolle agieren.
  - Weitere Rollen sind anwendungsspezifisch definierbar



SOAP-Knoten

- Rollen werden mit URIs referenziert
  - z.B. http://www.w3.org/2003/05/soap-envelope/role/next
  - muss jedoch keine Zieladresse sein, sie kann die Rolle auch abstrakt bezeichnen: urn:support:account
- Wozu Rollen?
  - SOAP-Nachricht weist Blöcke im Kopf auf, welche sich auf Rollen beziehen
  - Nimmt ein Knoten eine Rolle R an, zu der es einen Kopfblock gibt, so muss er die darin stehenden Informationen verarbeiten.
  - Kopfblöcke, die an „none“ gerichtet sind, dürfen nicht verarbeitet werden (sie können aber Daten beinhalten, die zur Verarbeitung eines anderen Kopfblocks notwendig sind)

SOAP-Kommunikationsmodell

- Aktionen bei Empfang einer Nachricht
  - Der Knoten bestimmt, in welchen Rollen er agiert (anhand der Kopfdaten und auch ggf. anhand der Nutzlastdaten)
  - Der Knoten identifiziert die Kopfblöcke, die an ihn gerichtet sind (in einer der Rollen, die er annimmt)
  - Sind bestimmte Kopfblöcke als verpflichtend („**mustUnderstand=true**“) markiert, aber nicht unterstützt, dann sendet er eine Fehlermeldung (nicht-verpflichtende Kopfblöcke dürfen generell ignoriert werden)
  - Ist der Knoten nicht der Endknoten, so entfernt er alle an ihn adressierte Kopfblöcke, setzt ggf. neue ein und sendet die Nachricht weiter

SOAP-Weiterleitung

- Übersicht zum Verhalten von SOAP-Knoten

Rolle		Kopfblock	
Kurzbezeichnung	Annahme	Verstanden+ verarbeitet	Weitergeleitet
next	ja	ja	nein[1]
		nein	nein[2,3]
anwendungsspezifisch	ja	ja	nein[1]
		nein	nein[2,3]
	nein	-	Ja
ultimateReceiver	ja	ja	-
		nein	-
none	nein	-	Ja

(1) Der Block kann jedoch explizit wieder eingefügt werden
(2) Das Attribut relay="true" führt zu einer Weiterleitung dieses Blocks
(3) Ist mustUnderstand="true", dann ist die Angabe von relay="true" wirkungslos, es kommt zu einer Fehlermeldung.

SOAP-Knoten

- Zwischenknoten kennen zwei Verhaltensweisen
  - weiterleitend
    - Der Knoten verhält sich gemäß dem Kommunikationsmodell
    - Er schickt die Nachricht anhand der Kopfdaten weiter
  - aktiv
    - Der Knoten verhält sich gemäß dem Kommunikationsmodell
    - Er bearbeitet außerdem die Nutzdaten (z.B. verschlüsseln, Anfügungen vornehmen)
    - Die folgende Verarbeitung in der Knotenkette hängt vom Verhalten dieses aktiven Knotens ab (z.B. wegen der Modifikation der Kopfdaten)
- Endknoten *muss* den Nachrichteninhalt verstehen und verarbeiten
  - alle anderen *dürfen* ihn verarbeiten

SOAP-Kommunikationsmuster

- Kommunikationsmuster (Message Exchange Pattern) beschreiben Interaktion zwischen Knoten
- Vordefinierte Muster
  - http://www.w3.org/2003/05/soap/mep/request-response/
    - Genau eine SOAP-Nachricht als Anfrage, genau eine SOAP-Nachricht als Antwort
    - Bezieht sich auf unmittelbaren Sender und unmittelbaren Empfänger
  - http://www.w3.org/2003/05/soap/mep/soap-response/
    - Genau eine Nicht-SOAP-Nachricht als Anfrage, genau eine SOAP-Nachricht als Antwort

Einfaches Beispiel

```
<envelope>
    <Header>
       <transId>123</transId>
    </Header>
    <Body>
       <Add>
          <a>3</a>
          <b>4</b>
       </Add>
    </Body>
</envelope>
```

Aufbau einer einfachen SOAP-Anfrage

c=Add(3,4)

SOAP-Anfrage

... hier nun die genauere Version

```
<?xml version='1.0' ?>
<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:Header>
       <t:transId xmlns:t="http://a.com/trans">123</transId>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
       <m:Add xmlns:m="http://a.com/Calculator">
          <m:a xsi:type="integer">3</m:a>
          <m:b xsi:type="integer">4</m:b>
       </m:Add>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

(*)

*Die Nutzung von XML-Schema-Attributen in XML-Dokumenten wird über den Namensraum „XMLSchema-instance“ ermöglicht.

SOAP-Antwort

```
<?xml version='1.0' ?>
<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:Header>
       <t:transId xmlns:t="http://a.com/trans">123</transId>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
       <m:AddResponse xmlns:m="http://a.com/Calculator">
          <m:c xsi:type="integer">7</m:c>
       </m:AddResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP-Datenmodell

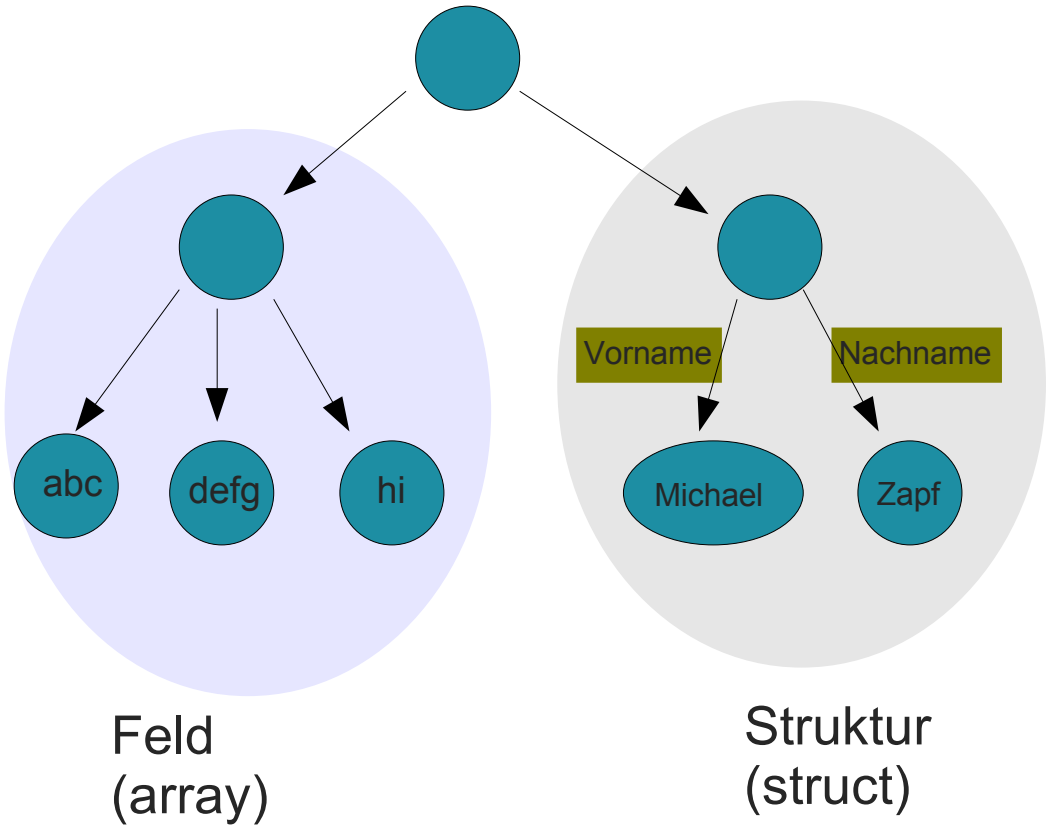
- Bislang definiert: Nachrichtenstruktur
- Jedoch findet auch Datenverarbeitung statt
  - nicht nur einfache Zahlen, Strings
  - XML-Schema definiert auch komplexe Typen
  - Datenmodell erforderlich, um anspruchsvolle Datenstrukturen auch mittels SOAP zu verarbeiten
- Ziel: Abbildung von Nicht-SOAP-Daten in ein transportables, SOAP-kompatibles Format
  - Außerdem Definition eines Kodierungsschemas sowie eines Prozeduraufrufformats (RPC)
- Verwendung des Datenmodells ist optional
  - Daten könnten schließlich bereits in XML-Form vorliegen

SOAP-Datenmodell

- Datenmodell in Form eines gerichteten Graphen
- Knoten haben einen (optionalen) Typnamen des Typs xs:QName
  - Namensraum xmlns:xs="http://www.w3.org/2001/XMLSchema"
  - XML Qualified Name (Angabe eines Namensraums und einer (dort) lokalen Bezeichnung
  - Beispiel: http://www.meinedomaene.dom/namespaces/ns1
- Knoten können eine oder mehrere Eingangskanten aufweisen
  - Einfachreferenz / Multireferenz
- Knoten haben einen
  - zugeordneten textuellen Wert, wenn sie keine ausgehende Kante aufweisen (einfacher Wert)
  - komplexen Wert in Form eines untergeordneten Graphen

SOAP-Datenmodell

- Kanten laufen zwischen Knoten
  - ausgehende / eingehende Kante
  - dürfen von einem Knoten zu sich selbst zurück laufen
- Kanten sind unterscheidbar
  - durch Beschriftung (xs:QName): Struktur
  - durch Position im Graph (implizite Nummerierung): Feld



SOAP-RPC

- Nachbildung des Remote Procedure Calls auf dem Web
- Voraussetzung: Gewisse Daten müssen vorliegen und in SOAP kodierbar sein
  - Adresse des Zielknotens
  - Prozedur-/Methodenname
  - Identitäten und Werte von Übergabeparametern
  - Trennung zwischen echten Funktionsargumenten und aufrufspezifischen Parametern
  - Kommunikationsmuster (Message exchange pattern)
  - optionale Daten für SOAP-Kopf
- Repräsentation nicht von SOAP selbst festgelegt!

SOAP-RPC

- Auswertung
  - Zielknoten muss Objekt/Methode/Prozedur aus einer URI bestimmen können
  - Aufrufparameter und Kommunikationsmuster wichtig für zu Grunde liegendes Protokoll (z.B. HTTP)
- Datenrepräsentation
  - wird bestimmt durch encodingStyle
  - kann vorher ausgehandelt worden sein

SOAP-RPC-Aufruf

Beispiel für optionale Daten für den SOAP-Kopf

Empfänger= ultimateRcv (keine role-Angabe)

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
 <env:Header>
   <t:transaction
          xmlns:t="http://thirdparty.example.org/transaction"
          env:encodingStyle="http://example.com/encoding"
          env:mustUnderstand="true" >sometranstype</t:transaction>
 </env:Header>
 <env:Body>
  <m:chargeReservation
      env:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
                xmlns:m="http://travelcompany.example.org/">
   <m:reservation xmlns:m="http://travelcompany.example.org/reservation">
    <m:code>FT35ZBQ</m:code>
   </m:reservation>
   <o:creditCard xmlns:o="http://mycompany.example.com/financial">
    <n:name xmlns:n="http://mycompany.example.com/employees">Michael Zapf</n:name>
    <o:number>123456789099999</o:number>
    <o:expiration>2006-02</o:expiration>
   </o:creditCard>
  </m:chargeReservation>
 </env:Body>
</env:Envelope>
```

reservation.code="FT35ZBQ";
creditCard={name="Michael Zapf", number="123...", expiration="2006-02"};

public void chargeReservation(in reservation, in creditCard);

RPC-Antwort

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
 <env:Header>
    <t:transaction
       xmlns:t="http://thirdparty.example.org/transaction"
         env:encodingStyle="http://example.com/encoding"
          env:mustUnderstand="true">sometranstype</t:transaction>
 </env:Header>
 <env:Body>
    <m:chargeReservationResponse
       env:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
           xmlns:rpc="http://www.w3.org/2003/05/soap-rpc"
             xmlns:m="http://travelcompany.example.org/">
       <rpc:result>m:status</rpc:result>
       <m:status>confirmed</m:status>
       <m:code>FT35ZBQ</m:code>
       <m:viewAt>http://travelcompany.example.org/reservations?code=FT35ZBQ</m:viewAt>
    </m:chargeReservationResponse>
 </env:Body>
</env:Envelope>
```

m:status ist vomTyp *QName*

Zusammen mit der Anfrage entspricht dies

public status chargeReservation(in reservation, in creditCard, out code, out viewAt)

Protokollbindung

- Bislang noch nicht erklärt, wie ein SOAP-Aufruf *übermittelt* wird
- Muster des Nachrichtenaustauschs sollte auf unterliegendes Protokoll passen
  - ggf. zusätzliche Hilfsdaten zur Korrelation von Nachrichten
- RPC ist normalerweise Anfrage-Antwort-Schema
  - passt gut auf HTTP Request – Response
  - kann aber auch andere Protokolle als HTTP nutzen

HTTP-Bindung

- Interessanteste Variante, auch Motivation für gesamte SOAP-Technik
- Besonderheiten von HTTP beachten
  - erfolgreiche Verarbeitung führt zu Antwort mit Statuscode 200
  - Probleme bei der Verarbeitung sollten mit 4xx oder 5xx gemeldet werden
- Zwischenknoten können aktiv sein
  - insbesondere, wenn sie nur HTTP/1.0 verstehen; sie können dann die Anfrage ändern

HTTP-Bindung

- Für reine Informationsabfrage kann die Abbildung auf HTTP GET erfolgen
  - Muster soap-response
  - Daten an Abfrage-URI hängen

```
GET /reservations?reservationCode=FT35ZBQ  HTTP/1.1
Host: travelcompany.example.org
Accept: application/soap+xml
```

Adresse stammt aus der Antwort des RPC-Aufrufs, siehe <viewAt>

- Für request-response muss die Abbildung auf HTTP POST erfolgen

```
POST /reservations HTTP/1.1
Host: travelcompany.example.org
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
    ...
```

hier entsprechend das XML-Dokument für den RPC-Aufruf

HTTP-Bindung

- Antwort über HTTP

```
HTTP/1.1 200 OK
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>
       ...
       ...
 </env:Header>
 <env:Body>
       ...
       ...
 </env:Body>
</env:Envelope>
```

HTTP-Bindung

- ... oder vielleicht auch

```
HTTP/1.1 500 Internal Server Error
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: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:Text xml:lang="en-US">Processing error</env:Text>
...
```

„Webkompatibles“ SOAP

- Problem
  - Ziel sollten „idempotente“ Aufrufe sein (man sollte sie ohne Änderung der Ressource wiederholen können)
    - normalerweise mit GET zu erreichen (Angaben in URI)
    - aber SOAP braucht häufig längere Nutzlast (eben das XML-Dokument), also meist nur POST verwendbar

```
POST /Reservations HTTP/1.1
Host: travelcompany.example.org
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:Body>
    <m:retrieveItinerary
        env:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
             xmlns:m="http://travelcompany.example.org/">
      <m:reservationCode>FT35ZBQ</m:reservationCode>
    </m:retrieveItinerary>
  </env:Body>
</env:Envelope>
```

Beispiel:
Reiseroute abfragen

Reine Abfrage, aber mit POST

Nicht empfohlen!

„Webkompatibles“ SOAP

- Abhilfe
  - Wichtige Daten (zur Identifikation der Ressource) in URI repräsentieren: Methodenname und Parameter

```
GET /Reservations/itinerary?reservationCode=FT35ZBQ HTTP/1.1
Host: travelcompany.example.org
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn
```

erspart die POST-Abfrage

- Nicht anwendbar, wenn man SOAP-Kopfblöcke oder komplexe Daten übertragen muss
  - doch POST verwenden, aber „möglichst webkompatibel“

```
POST /Reservations?code=FT35ZBQ HTTP/1.1
Host: travelcompany.example.org
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 ...
```

Möglichst genau die Ressource in URI adressieren

Ganz abgefahren...

SOAP über E-Mail!

```
From: Michael.Zapf@mycompany.example.com
To: reservations@travelcompany.example.org
Subject: Travel to LA
Date: Thu, 29 Nov 2001 13:20:00 EST
Message-Id: <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">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</reference>
   <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/employees"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
         env:mustUnderstand="true">
   <n:name>Michael Zapf</n:name>
  </n:passenger>
 </env:Header>
 <env:Body>
  <p:itinerary
     xmlns:p="http://travelcompany.example.org/reservation/travel">
   <p:departure>
     ...
   </p:departure>
   <p:return>
     ...
   </p:return>
  </p:itinerary>
  <q:lodging
     xmlns:q="http://travelcompany.example.org/reservation/hotels">
   <q:preference>none</q:preference>
  </q:lodging>
 </env:Body>
</env:Envelope>
```

SOAP über E-Mail

```
From: reservations@travelcompany.example.org
To: Michael.Zapf@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">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</reference>
   <m:dateAndTime>2001-11-29T13:35:00.000-05:00</m:dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/employees"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
        env:mustUnderstand="true">
   <n:name>Michael Zapf</n:name>
  </n:passenger>
 </env:Header>
 <env:Body>
  <p:itinerary
     xmlns:p="http://travelcompany.example.org/reservation/travel">
   <p:itineraryClarifications>
      ...
      ...
   </p:itineraryClarifications>
  </p:itinerary>
 </env:Body>
</env:Envelope>
```

Referenz über ID

Web Services Description Language

- XML-Schema zur Beschreibung von Webservices
  - Service-Schnittstellen-Beschreibung
  - Abstrakte Semantik für Webservices
  - Implementierungsdefinition
- Konkrete Endpunkte und Netzadressen, an denen der WS anzusprechen ist
- Kernelemente von WSDL
  - Dienst, Port, Porttyp
  - Operationen und Nachrichten

WSDL-Aufbau

- Beschreibung als XML-Datei; Inhalt:
  - *documentation:* Allgemeine Beschreibung
  - *types:* Welche Datentypen werden in den Anfragen und Antworten verwendet?
  - *interface:* Welche Aufrufe versteht der Dienst? Welche Datentypen gehören zu den Aufrufen?
  - *pattern:* Wie kommuniziert man mit dem Dienst?
    Beispiel: Nur Aufruf; Eingabe-Ausgabe; Ausgabe-Eingabe
  - *binding:* Welches Protokoll kommt zum Einsatz?
    Beispiel: SOAP über HTTP
  - *endpoint:* Über welchen URI kann man den Dienst ansprechen?

WSDL-Beispiel

```
<?xml version="1.0" encoding="UTF-8"?>
<description    xmlns="http://www.w3.org/2005/05/wsdl"
                targetNamespace="http://www.vs.uni-kassel.de/wsdl/regSrv"
                xmlns:vs="http://www.vs.uni-kassel.de/schemas/regSrv"
                xmlns:tns="http://www.vs.uni-kassel.de/wsdl/regSrv"
                xmlms:wsoap="http://www.w3.org/2005/05/wsdl/soap">
<documentation>Dies ist ein Beispiel für eine WSDL-Beschreibung</documentation>

<types>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.vs.uni-kassel.de/schemas/regSrv"
        xmlns="http://www.vs.uni-kassel.de/wsdl/regSrv">

        <xs:element name="Student" type="typStudent"/>
        <xs:complexType name="typStudent">
            <xs:sequence>
                <xs:element name="Name" type="xs:string"/>
                <xs:element name="Matrikelnummer" type="xs:positiveInteger"/>
            </xs:sequence>
        </xs:complexType>
    </xs:schema>
</types>
```

WSDL-Beispiel (Fortsetzung)

```
<interface name="Registrierungsschnittstelle">
        <operation name="registrieren" pattern="http://www.w3.org/2005/05/wsdl/in-only" ...>
                <input messageLabel="In" element="vs:Student"/>
        </operation>
</interface>
<binding name="bindKlausurReg"
        interface="tns:Registrierungsschnittstelle"
        type="http://www.w3.org/2005/05/wsdl/soap"
        wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP">

        <operation ref="tns:registrieren"
                wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>
</binding>
<service name="Klausuranmeldung" interface="Registrierungsschnittstelle">
        <endpoint name="anmeldungsadresse"
                        binding="tns:bindKlausurReg"
                        address="http://www.vs.uni-kassel.de/klausur/anmeldung"/>
</service>
</description>
```

mep=Message Exchange Pattern

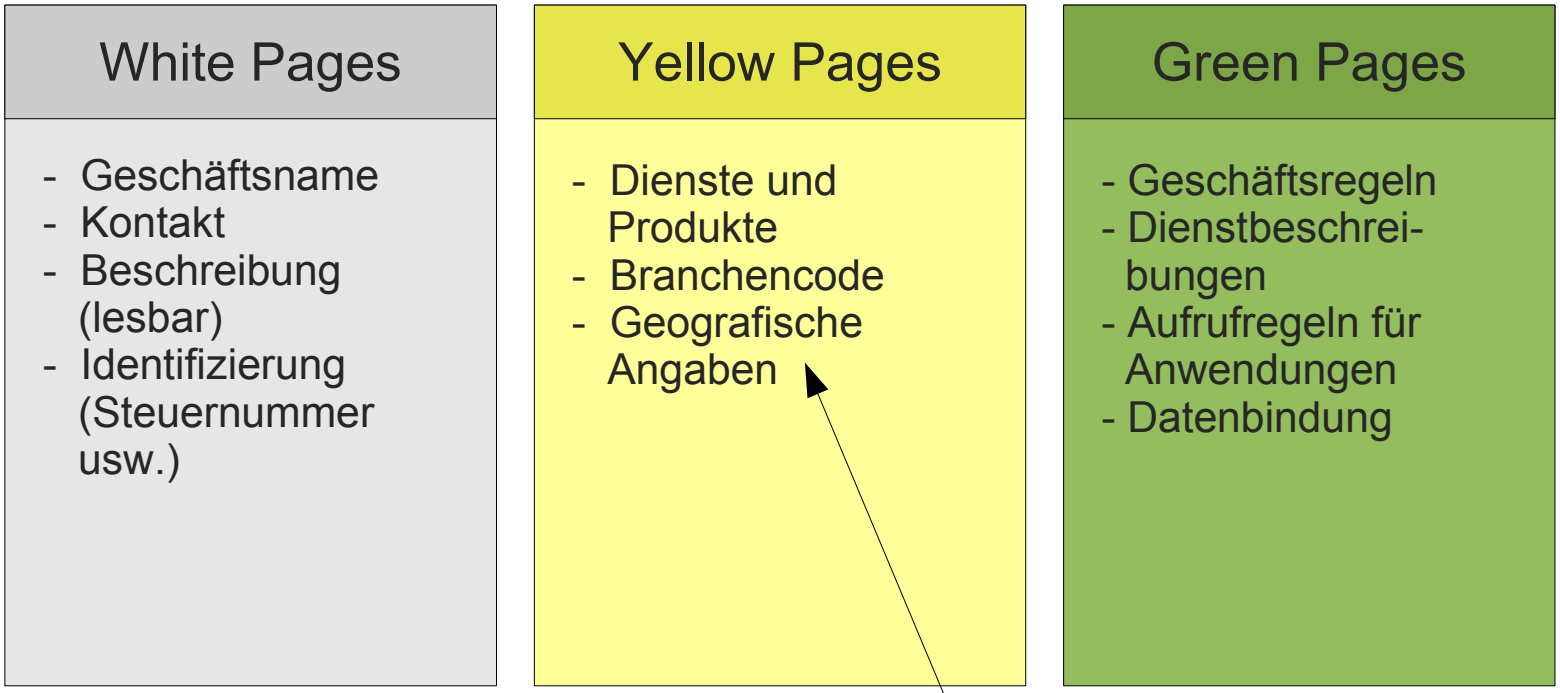
WSDL

- Große Flexibilität
  - Sieht auch andere Bindungen als HTTP und SOAP vor
- Recht aufwändige Beschreibung
  - Obiges Beispiel realisiert eigentlich nur einen Dienst mit einer einzigen Methode
- Nur selten per Hand erzeugt
  - Entwicklungsumgebungen können eine WSDL-Beschreibung aus der Dienstimplementierung automatisch generieren

UDDI

- Dienstverzeichnis
  - ermöglich Abfragen von WDSL-Beschreibungen
  - damit Vermittlung von Diensten anhand von Beschreibungen
- Spezifiziert über OASIS (www.oasis-open.org)
  - UDDI API
  - UDDI Data Structure
  - UDDI XML Schema
  - UDDI Replication Specification / XML Schema / Custody schema
  - UDDI Operator's Specification
  - UDDI WSDL Service Interface Descriptions
  - UDDI tModels

UDDI

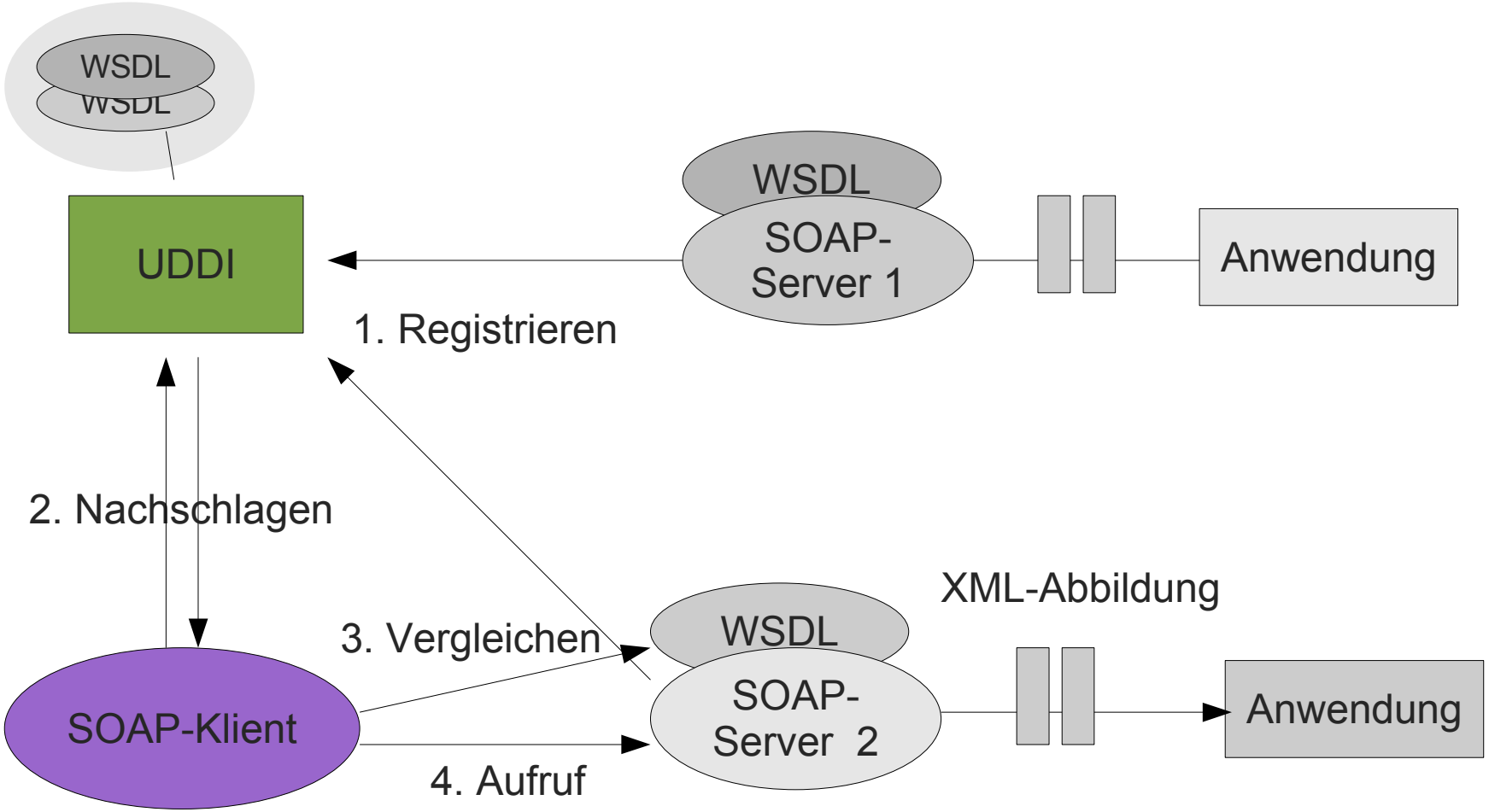


können bei der Dienstauswahl
von Interesse sein (Beispiel: Taxidienst)

UDDI

- Einträge durch XML-Schemata definiert
- SOAP-APIs zur Registrierung und Suche
  - **Publisher API**: Erzeugen, Sichten, Löschen von Registrierungen
  - **Inquiry API**: Registrierung finden (Matching), Details bekommen (bei bekanntem Dienst)
- Liefert URLs auf WSDL- und andere Webservice-Beschreibungen

SOAP, UDDI und WSDL



Anwendung von UDDI

- Ursprünglich gedacht, um verschiedene Geschäftspartner zusammenzubringen
- Vision ist nicht eingetreten. Stattdessen:
  - Dienste mit spezifischen Beschreibungen
  - Anbieter kodieren URLs in SOAP-Endpunkt hart ein
  - keine Vermittlung!
- Beispiel für eine aufwändige Spezifikation ohne erwiesen nützliches Geschäftsmodell?

Zusammenfassung

- Fundamentaler Trend: Verteilte Verarbeitung auf dem Internet
- Webservices bieten einen offenen, flexiblen, standardisierten Integrationsansatz für Anwendungen auf dem Web

Zusammenfassung

- SOAP – was heißt „Simple“?
  - zustandslose Objekte (wie häufig im Internet)
  - keine Callback-Objekte (keine bidirektionale Kommunikation)
  - keine verteilte Speicherbereinigung
  - keine Objektreferenzverarbeitung
  - kein entfernter Objektaufruf
  - kein Zustands- und Sitzungsmanagement
  - keine verteilten Transaktionen
  - einfach zu implementieren, installieren, administrieren
  - glücklichere Firewalls

Zusammenfassung

- Nachteile
  - Sicherheit
    - Kein Thema. (Warum gab es schnell nochmal Firewalls?)
    - Wenn auf SOAP aufgesetzt, behindert sie die Interoperabilität
    - Wenn auf HTTP aufgesetzt, erhöht sich der Adminstrationsaufwand
  - Effizienz
    - üble Performanz
    - die meiste Zeit entfällt auf das Erzeugen und Lesen der Nutzlast
  - Fehlen von Infrastrukturdiensten

Literatur

- Einführung
  - http://www.w3.org/TR/soap12-part0/
- Kommunikation
  - http://www.w3.org/TR/soap12-part1/
- Datenmodell und weiteres
  - http://www.w3.org/TR/soap12-part2/