Einführung in Web Services

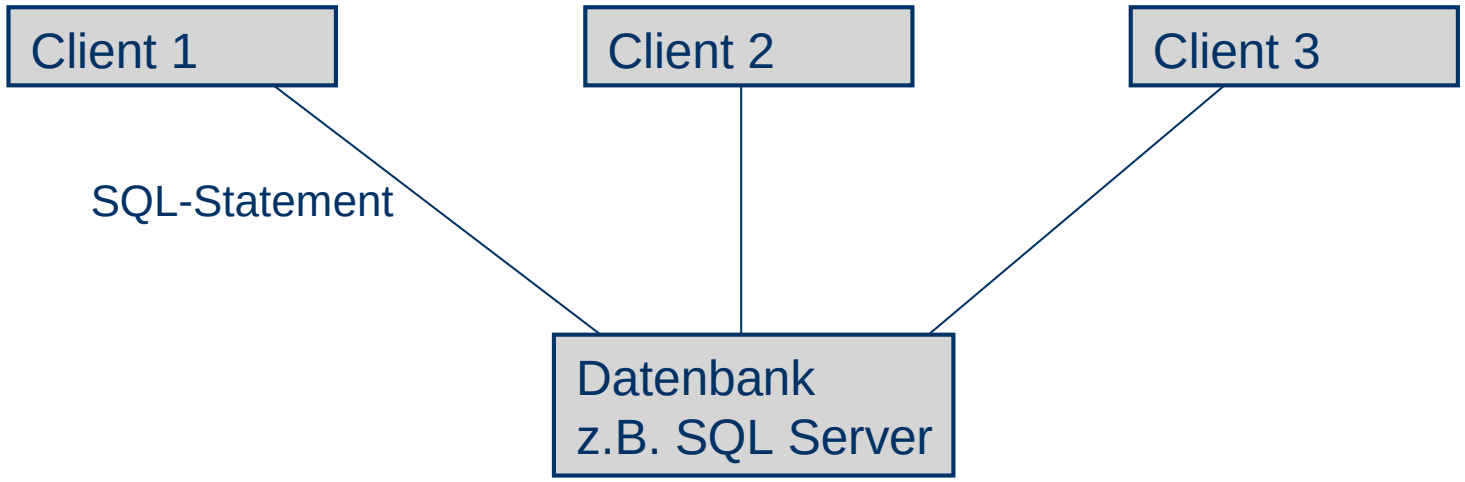
- Seminar Web Services
- am 18.05.2004
- von Can Okutan und
- Sascha Schnaufer

- Betreuer: Andreas Kamper

Agenda

- Einführung
  - Aufbau von Mehrbenutzer-Anwendungen
  - RMI
  - Corba
  - Warum Webservices ?
- Webservice im Detail
  - Architektur
  - XML
  - HTTP
  - SOAP
  - WSDL
  - UDDI
- Webservice Minibeispiel in .NET

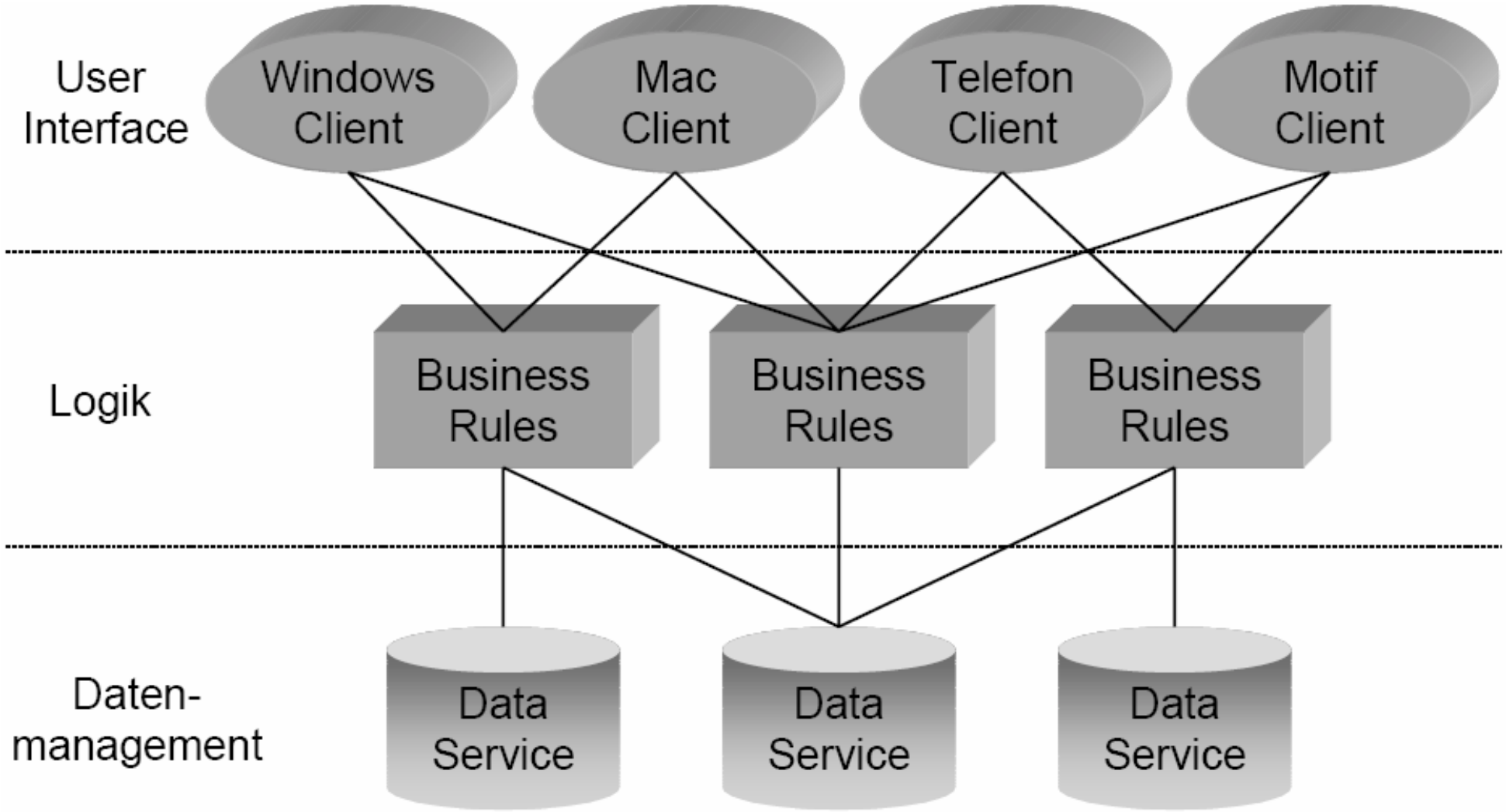
Anwendungen mit gemeinsamer Datenbank



- User Interface und Logik bei jedem Client
- Clients greifen direkt über SQL-Statements auf die Datenbank zu
- Auf dem Server läuft nur die Datenbank

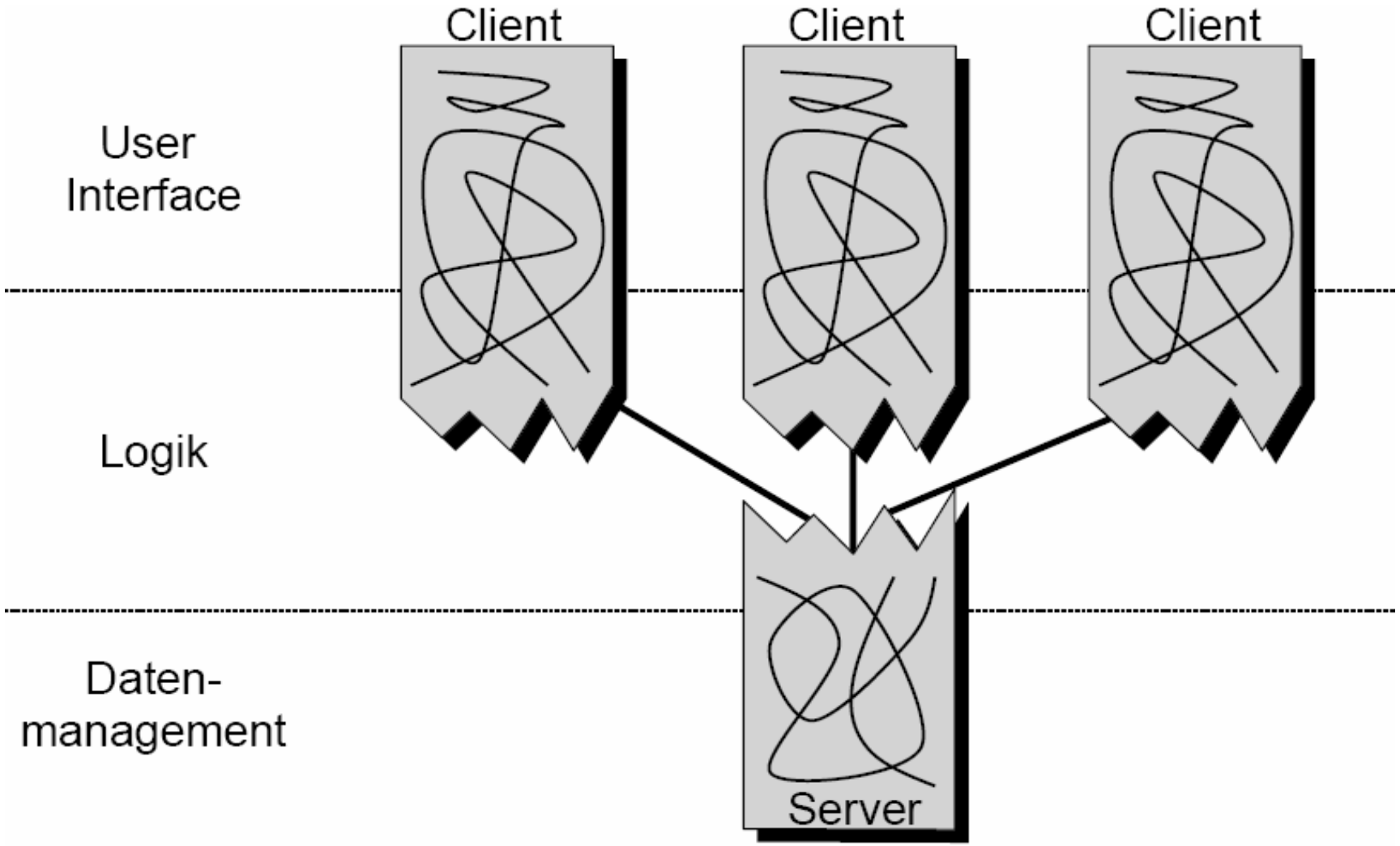
- Probleme:
  - Eventuell hohe Netzwerkbelastung z.B. bei der Suche von Duplikaten
  - Nur auf Datenbankebene können Daten zwischengespeichert werden
  - Updates müssen auf allen Clients eingespielt werden
  - …

„Three-Tiered“ Ansatz



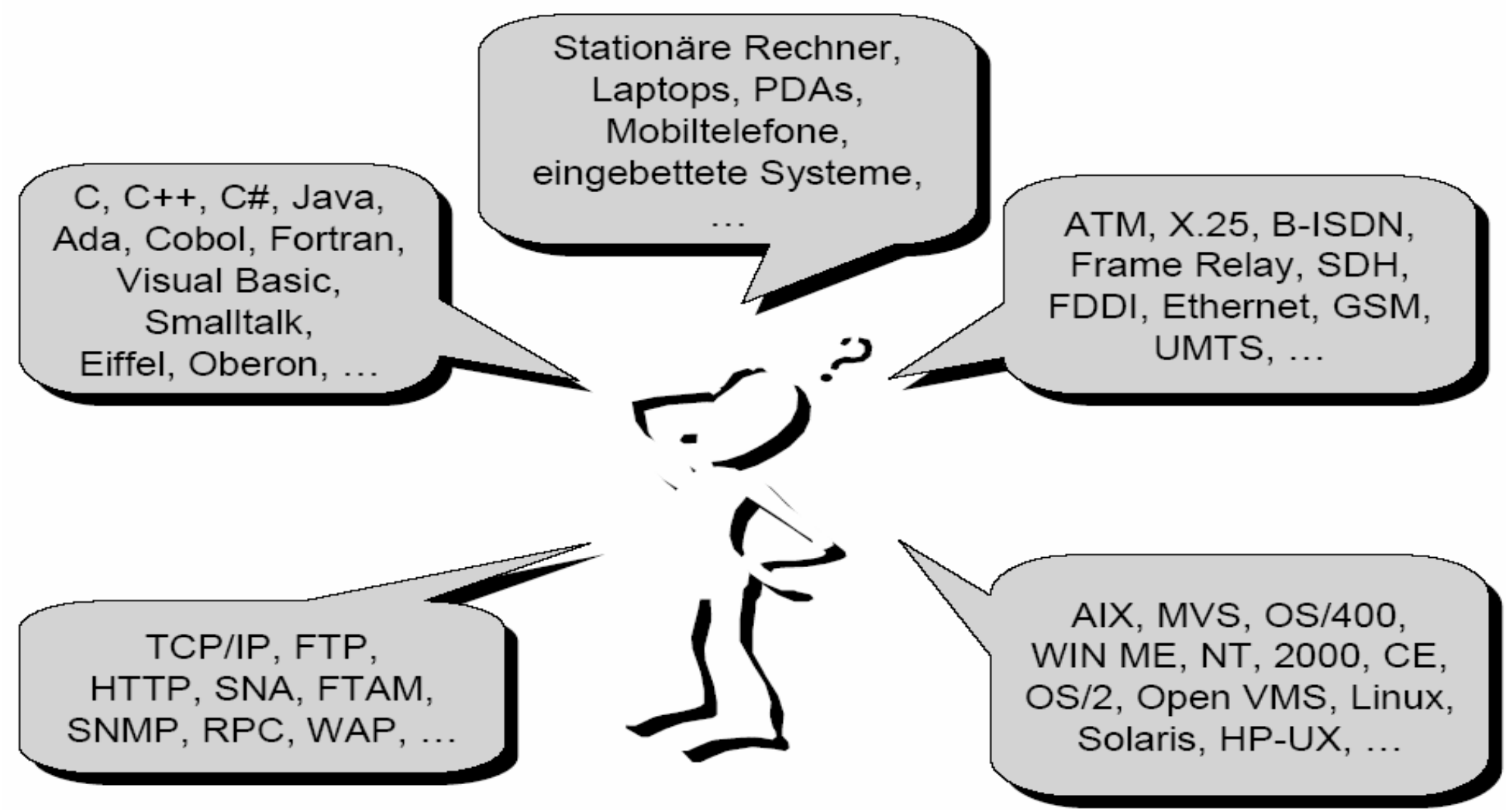
Quelle: Vorlesung Verteilte Objekte WS02 von Stefan Kuhlins

Logische Monolithen (Client / Server)



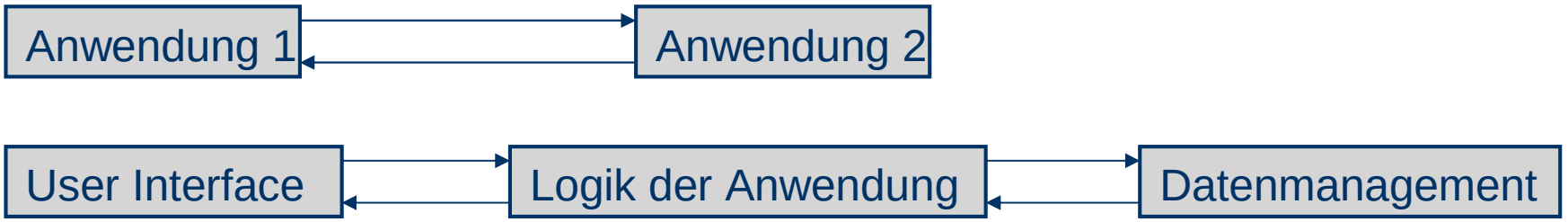
Quelle: Vorlesung Verteilte Objekte WS02 von Stefan Kuhlins

Heterogenität



Quelle: Vorlesung Verteilte Objekte WS02 von Stefan Kuhlins

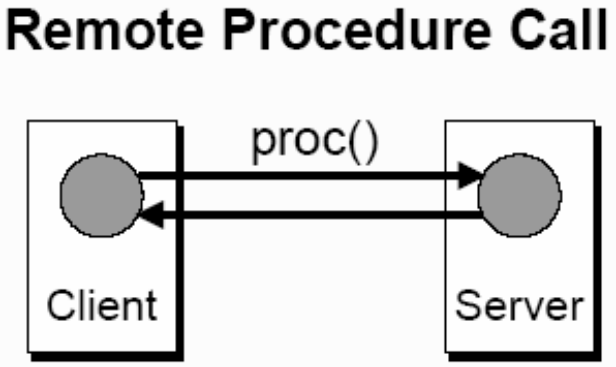
Kommunikation zwischen Anwendungen/Schichten (1)



- Kommunikation über Netzwerk, aber wie genau ?
- z.B. Eigene Implementation
  - über TCP auf einem festgelegten Port
  - Aufbau der Nachrichten definieren
  - z.B. jeder Funktion eine Nummer zuordnen
    - **`-> 1 = String getKundenName( String Kundennummer )`**
    - **`Aufbau des Anfragepaketes '1, K50015'`**
    - **`Aufbau des Antwortpaketes '1, Müller'`**

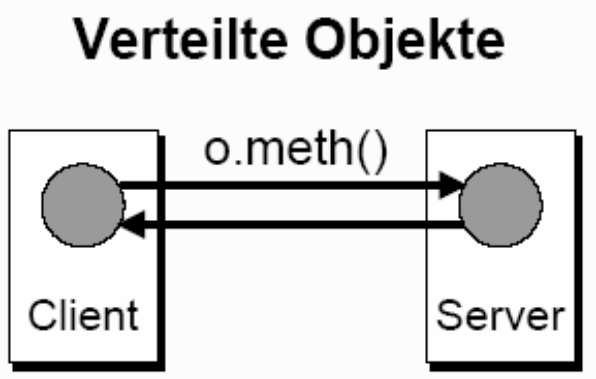
Kommunikation zwischen Anwendungen/Schichten (2)

- Probleme:
  - Extrem unflexibel
  - Viel zu definieren
  - Der gesamte Kommunikationscode muss selbst geschrieben werden
  - Fehleranfällig



RMI („Remote Method Invocation“) (1)

- Basiert auf der Java Virtual Machine (JVM)
- Einfache Entwicklung verteilter Java-Anwendungen
- Typprüfung
- Sicherheit (Security Manager)
- Natürliche Integration in die Sprache



RMI („Remote Method Invocation“) (2)

- Server:
  - Erzeugen „entfernte“ Objekte
  - Warten auf Methodenaufrufe für die Objekte
- Client:
  - Bekommen Referenzen auf entfernte Objekte
  - Rufen Methoden für die entfernten Objekte auf
- RMI Steuert die Kommunikation

RMI („Remote Method Invocation“) (3)

- Vorgehen:
  - 1. Remote Java-Interface einer Klasse definieren
  - 2. Interface implementieren
  - 3. Server implementieren
  - 4. Client implementieren

- 1. Remote Interface definieren:

```
import java.rmi.*;

public interface RMICounter extends Remote
{
  void reset() throws RemoteException;
  void increment() throws RemoteException;
  int value() throws RemoteException;
}
```

RMI („Remote Method Invocation“) (4)

- 2. Interface implementieren:

```
import java.rmi.*;
import java.rmi.server.*;

public class RMICounterImpl
  extends UnicastRemoteObject
  implements RMICounter
{
  private int counter;
  public RMICounterImpl() throws RemoteException { super(); }
  public void reset() throws RemoteException { counter = 0; }
  public void increment() throws RemoteException { ++counter; }
  public int value() throws RemoteException { return counter; }
}
```

RMI („Remote Method Invocation“) (5)

- 3. Server implementieren

```
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;

public class RMICounterServer {
  public static void main(String args[]) {
    try {
      LocateRegistry.createRegistry(1099);
      RMICounter myCount = new RMICounterImpl();
      Naming.rebind("Counter", myCount);
      System.out.println("Server ready");
    }
    catch(RemoteException x) { … }
    catch(java.net.MalformedURLException x) { … }
  }
}
```

RMI („Remote Method Invocation“) (6)

- 4. Client implementieren

```
import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.*;

public class RMICounterClient
{
  public static void main(String args[]) {
    try {
      RMICounter myCount = (RMICounter) Naming.lookup("rmi://" +
                           args[0] + "/Counter");
      myCount.reset();
      myCount.increment();
      System.out.println("Result = " + myCount.value());
    }
    catch(NotBoundException x) { … }
    catch(java.net.MalformedURLException x) { … }
    catch(RemoteException x) { … }
  }
}
```

RMI („Remote Method Invocation“) (7)

- Mit rmic RMICounterImpl werden Hilfsklassen generiert
  - RMICounterImpl_Skel.class
  - RMICounterImpl_Stub.class
- Vorteile:
  - Relativ einfach zu implementieren
  - Natürliche Integration in Java
  - RMISecurityManager
- Nachteile:
  - Ist von der Programmiersprache Java abhängig
  - Relativ langsam und Fehleranfällig (gilt aber für alle Verteilten Systeme)

OMG/CORBA (1)

- Ähnliches System wie RMI
- Interface wird mit IDL spezifiziert

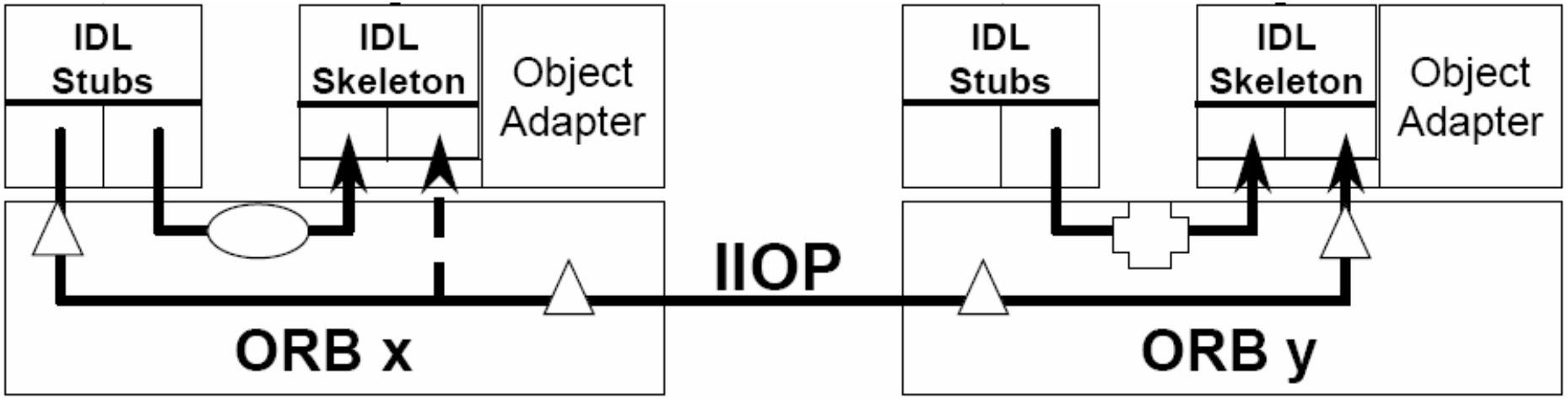
```
module CounterApp {
  interface Counter {
    void reset();
    void increment();
    long value();
  };
};
```

- IDL ist von der benutzten Programmiersprache unabhängig
- Parallel zu RMI werden Hilfsklassen erzeugt (IDL Compiler)

OMG/CORBA (2)

- Der Object Request Broker (ORB) beschreibt den Mechanismus, der das
  - hardwareunabhängige,
  - betriebssystemunabhängige und
  - programmiersprachenunabhängige

  Aufrufen von Diensten ermöglicht.



OMG/CORBA (3)

- Vorteil gegenüber RMI:
  - Abstrakte Definition eines ORBs und deren Kommunikation untereinander über IIOP ->
    - Unabhängig von einer Programmiersprache, da es für fast jede einen passenden ORB gibt
  - Corba bietet weitere Dienste für verteilte Anwendungen

- RMI Objekte können mit einen Java ORB über das IIOP „freigegeben“ werden.

Was bieten RMI und Corba nicht ?

- Kein allgemeines, offenes, freies und verbreitetes Protokoll (sondern Eigenentwicklungen wie IIOP)
- Sind weniger dafür ausgelegt „fremde“ Objekte zu integrieren (in Normalfall kennen sich die Partner)
- Lassen sich nicht gut in das Internet integrieren
- Es gibt keine Suchmaschine für RMI / Corba Dienste

Webservices

Def:

Ein Dienst, der mit Hilfe von XML auf der Basis von Internet-Netzwerkprotokollen erbracht wird. Sie bilden die drei wichtigsten Teile der Zusammenarbeit ab:

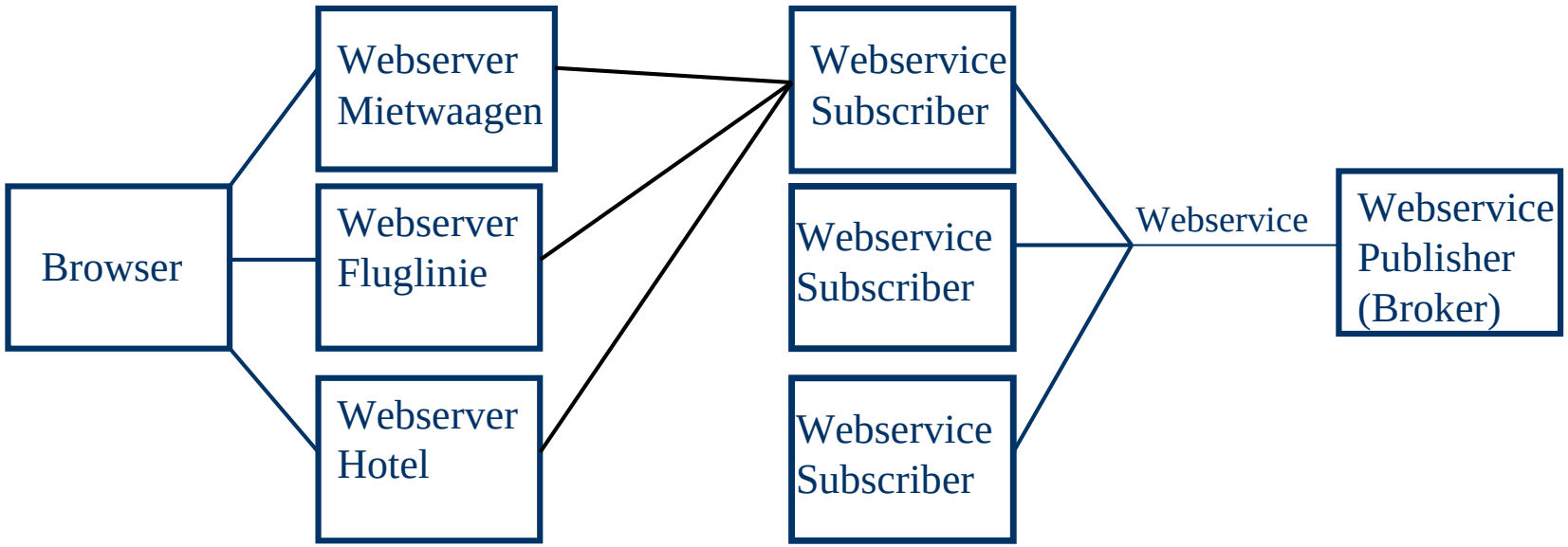
- Zusammenfinden
- Binden
- Datenaustausch

Def:

Webservices sind Methoden, die über ein Netzwerk auf entfernten Rechnern zur Verfügung stehen.

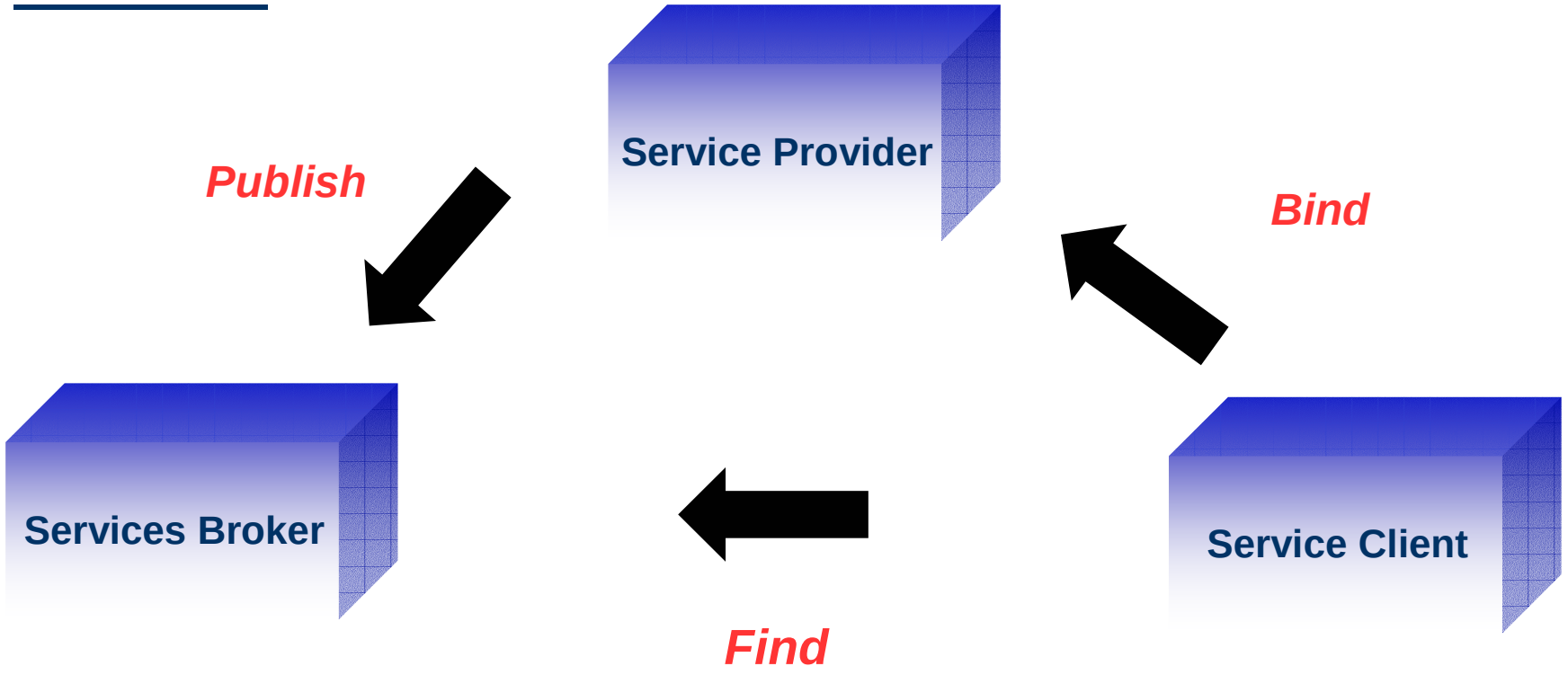
Architektur

- Web Services im engeren technischen Sinn meint automatisierte Kommunikation zwischen Applikationen übers Internet. Es werden also nicht HTML-Seiten zu einem Webbrowser geschickt, die von einem Menschen betrachtet werden, sondern Programme tauschen Daten und starten auf entfernten Rechnern Funktionen



Webservices

- Schema



System Modell

- Webservices ermöglichen zu können bedient man sich Standards.

Die weltgrößten Softwarehersteller (Microsoft, IBM,…)
haben sich auf Prozeduren und Protokolle für
Web Services geeinigt, nämlich:

SOAP (Simple Object Access Protocol)
WSDL (Web Services Description Language)
UDDI (Universal Description, Discovery, and Integration)

XML (Extensible Markup Language)

Datenaustausch: Wird mit Hilfe von SOAP und HTTP durchgeführt.

Zusammenfinden: Ein Webservice kann man automatisch via UDDI Finden. UDDI selbst basiert auf SOAP über HTTP.

Binden: Für das Binden von Webservices dominiert der WSDL Standard. Hier kann man alle wichtigen Informationen ablegen, die man benötigt als Nutzer und Anbieter. **Es beschreibt den Webservice formal.**

Während bisher bei verteilten Systemen die elektronische Kommunikation über Rechnergrenzen hinweg meistens per CORBA, RMI oder DCOM erfolgte, nutzen Web Services einfaches XML, meistens über HTTP.

HTML contra XML

- ...ist die Technologie für Web Applicationen und Services!
- Seit Anfang 1998 offizieller Standard des W3C.

XML statt HTML

HTML Beispiel:

```
<p> <b> Herr Arnold Braun</b>
<br>
Kaiserallee: 5<br>
Karlsruhe , D-76133</p>
```

Herr Arnold Braun
Kaiserallee:5
Karlsruhe, D-76133

XML

Das selbe Beispiel in XML:

```
<address>
<name>
<title>Herr</titel>
<first-name>Arnold</first-name>
<last-name>Braun </last-name>
</name>
<street> Kaiserallee: 5 </street>
<city>Karlsruhe </city>
<state>D- </state>
<zipcode>76133 </zipcode>
</address>
```

- Mit XML, kann man die Bedeutung der Tags als Leser verstehen.
- Noch wichtiger- der Computer kann die Bedeutung verstehen bzw. interpretieren.

- XML wurde entwickelt, um Daten zu beschreiben.
  → (XSL, DTD)
- HTML dagegen wurde entwickelt, um Daten darzustellen.

Vor- und Nachteile

- **Vorteil von XML:**

1. liegt in der Fähigkeit zum Datenaustausch: Wichtig beim Business-to-Business, da verschiedene Firmen nicht mit gleichen Anwendungen arbeiten und so die gegenseitige Kommunikation erschweren.
2. Methodenaufrufe werden von verschiedene Applikationen (Anwendungen und Programme) über dasselbe Protokoll (HTTP) mit Hilfe von SOAP übertragen.

- **Nachteil von XML**

1. Es ist noch nicht problemlos von allen Browsern zu lesen.

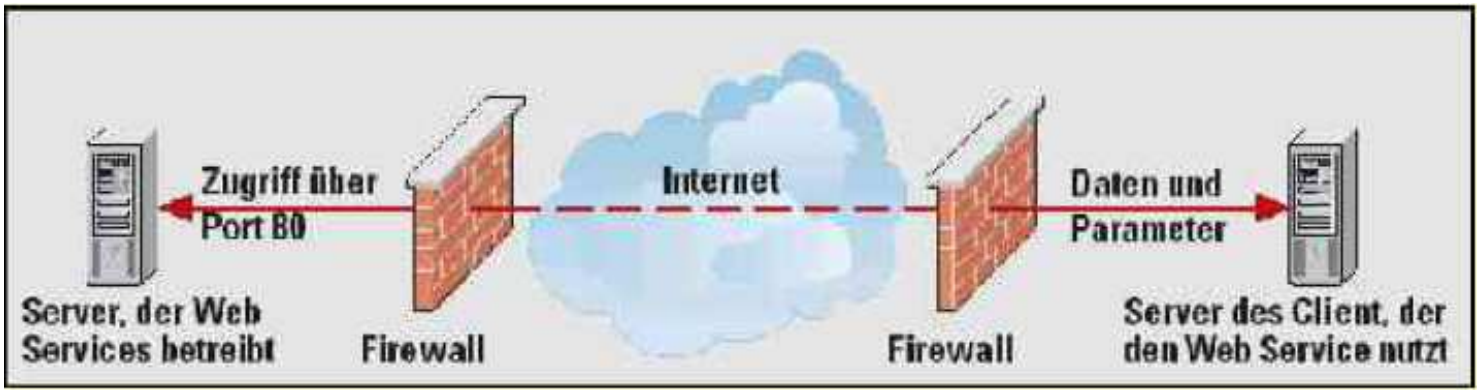
HTTP (Hypertext Transfer Protocol)

- Internet hauptsächlich aus HTTP Verbindungen bestehend
  HTTP- Übertragungsprotokolle einfach aufgebaut.

- Einfache Struktur bzw. Aufbau: Es werden einfache Stringwerte übertragen.

- Die meisten HTTP-Port (Port 80 Standard Port) frei geschaltet.

- SOAP kann mit verschiedenen Transportprotokollen verwendet werden, meistens wird HTTP gewählt. Die Standard-Internetprotokolle erleichtern den Betrieb auch über Firewalls hinweg, was mit CORBA, RMI und DCOM schwieriger zu realisieren.



Kommunikation mit Web Services erfolgt durch die Firewalls hindurch

SOAP (Simple Object Access Protocol)

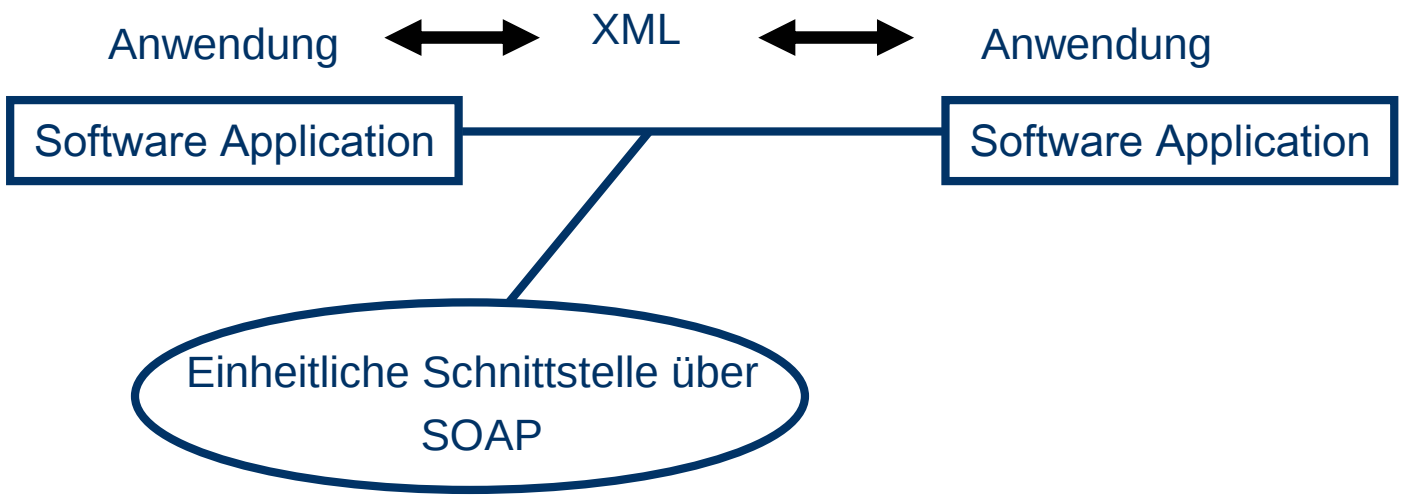
- Schlankes auf XML basierendes Protokoll zum Informationsaustausch in einer dezentralen Umgebung.
- Es ist ein plattformunabhängiges Protokoll, welches dazu dient, Anwendungen über das Web oder Hypertext Transfer Protocol (HTTP) kommunizieren zu lassen.
- SOAP dient also zum Austausch von komplexen Datenobjekten zwischen verschiedenen Programmen.
- SOAP Nachrichten werden durch Standards codiert und gesendet.

Das Besondere an SOAP Web Services ist die allgemeine Akzeptanz, Kommunikation ist sogar zwischen den beiden konträren Plattformen Sun / Java und Microsoft / .NET möglich.

Nachrichten über SOAP

- Die wichtigsten Vorteile von SOAP sind: allgemein akzeptierte Standardisierung, Plattformunabhängigkeit, Offenheit, Robustheit und Skalierbarkeit.

- Der gravierendste Nachteil ist: mehr Overhead und dadurch etwas geringere Performance wegen des verwendeten Datendarstellungsprotokolls XML.



Wenn wir Daten also zwischen heterogenen Systemen austauschen, muss man eine gemeinsame Darstellung vereinbaren!!!

Problem der Darstellung(1)

Beispiel: Viele Darstellungen einer Telefonnummer

```
<phonenNumber>(123) 456-7890 </phonenNumber>
```

```
<phonenNumber>
        <areaCode>123</areaCode>
        <exchange>456</exchange>
        <number>7890 </number>
</phonenNumber>
```

```
<phonenNumber area = “123“ exchange= “ 456 “ number= “ 7890 “/>
```

Problem der Darstellung(2)

- **Folgerung:**
  Es reicht nicht zu sagen, dass Server und Client XML für den Nachrichtenaustausch verwenden. Folgendes muss definiert sein:

•Die Informationstypen, die wir austauschen.

•Wie diese Informationen in XML ausgedrückt werden sollen.

•Wie diese Informationen tatsächlich verwendet werden.

SOAP stellt diese Konventionen zur Verfügung!!!

Bestandteile von SOAP

Bestandteile:

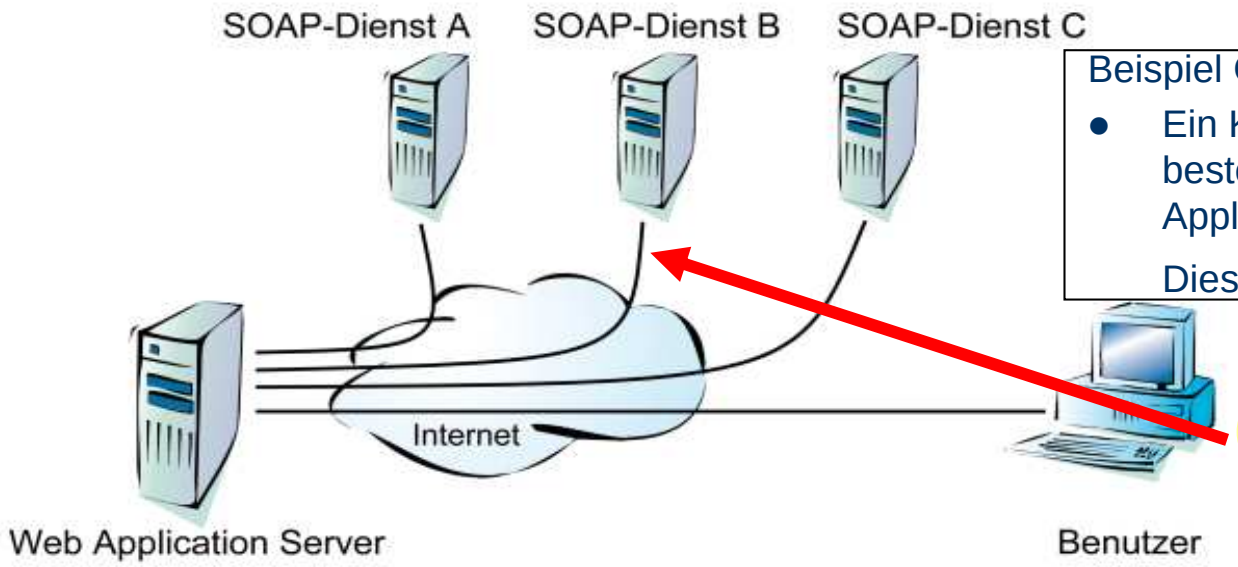
Envelope- umschließt Header und Body

Header- enthält zusätzliche Informationen (optional)

Body-Prozeduraufrufe und deren Rückgaben

- SOAP dient zur Basiskommunikation mit Web Services kann aber **keine Informationen bzgl. Der Nachrichten geben.**

Beispiel: SOAP Nachrichtenaustausch



Beispiel Onlineshop:

- Ein Kunde möchte in einem Onlineshop Artikel bestellen. Der Onlineshop ist in einen Applikationsserver realisiert.

  Dieser Applikationsserver:

Funktionsparameter werden codiert und an Webservice geschickt!

(1) überprüft über den SOAP Web Service des Servers A, ob die angegebene Adresse gültig ist,

(2)ermittelt über den SOAP-Dienst C die für das jeweilige Land zu berechnenden Steuern (Umsatzsteuer, Luxussteuer, ...),

(3)verifiziert über den SOAP-Dienst die Kreditkartennummer,

(4)erfragt beim SOAP-Dienst D tagesaktuelle Währungsumrechnungskurse, um den Endpreis korrekt berechnen zu können und…

(5)zeigt dem Benutzer alle Ergebnisse gesammelt im Webbrowser an.

WSDL (Web Services Description Language)

…für eine erfolgreiche Interaktion mit Web Services…

WSDL (Web Services Description Language, http://www.w3.org/TR/wsdl) ist ein XML-Derivat zur Beschreibung der Schnittstellen von Web Services. Nachrichtenstrom-Formate und Funktionsaufrufe werden definiert.

Programmiersprachen-unabhängige XML- Version. Nimmt Befehle in Form von Funktionsparametern als SOAP- Dateien entgegen und kann diese weiterleiten oder eine entsprechende Antwort zurücksenden.

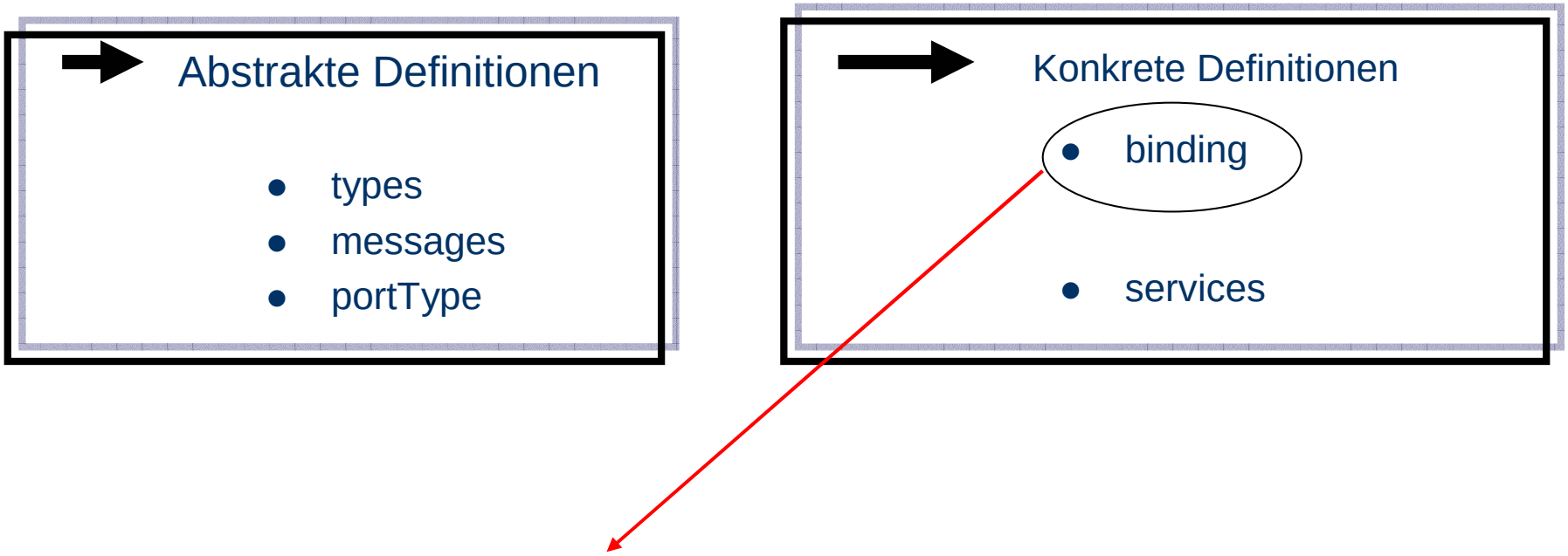
Beschreibt die Operationen, welche Webservices ausführen

WSDL erlaubt:

Programmiersprachen-spezifischen Code zu erzeugen, der die Funktionen, welche eine WSDL- Datei beschreibt, in SOAP- basierte Aufrufe übersetzt.

WSDL- Dokumentenstruktur

- Ein Dokument besteht aus mehreren Definitionen
- Definition können unterteilt werden in:



- Gibt bekannt, wie eine Interaktion über ein spezifisches Protokoll mit SOAP durchgeführt werden soll.

UDDI (Universal Description, Discovery, and Integration)

- UDDI ist ein webbasiertes Informationssystem für Web Services. Es bietet ein Verzeichnis von Adress- und Produktdaten sowie von Anwendungsprogrammierschnittstellen (WSDL- Dateien).
- Entwickelung u.a. von Ariba, IBM und Microsoft.
- Zu finden unter http://www.uddi.org.
- Ist selbst ein Web Service, der als Verzeichnis zum Veröffentlichen und Auffinden anderer Web Services fungiert.
- Datenbank auf XML basierend.

UDDI bei IBM

- Stark vereinfacht!!!

„Add a new Business“

- Name
- Adresse

„Add a new Service“

- Service for…? (Description)

„Add a new technical Model“

- URL der WSDL- Datei eintragen

Description

Gliederung bei UDDI

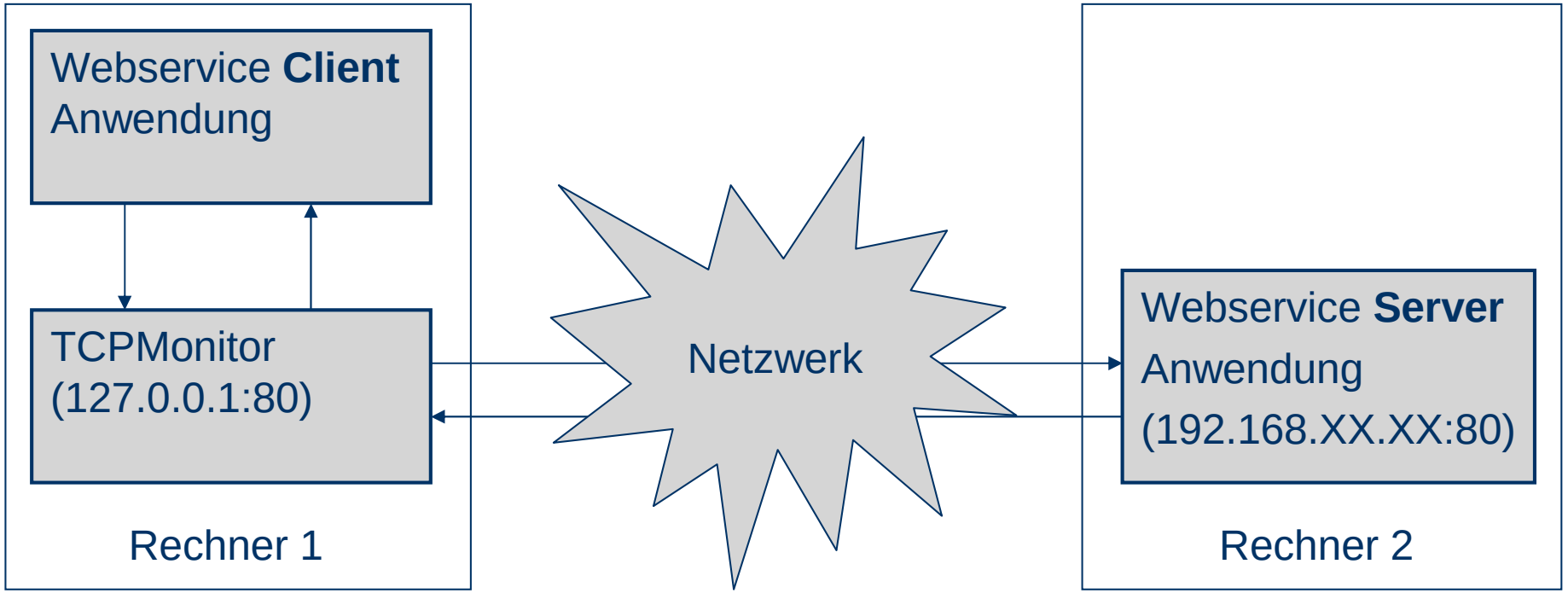
Datenbank wurde eingeteilt in:

- Business: Wenn man nach Firmennamen sucht
- Services: Wenn man nach dem Namen eines Dienstes sucht.
- Technical Model: technische Beschreibung des Webservice (speziell wenn ein Webservice über eine WSDL Datei definiert ist.)

Beispiel

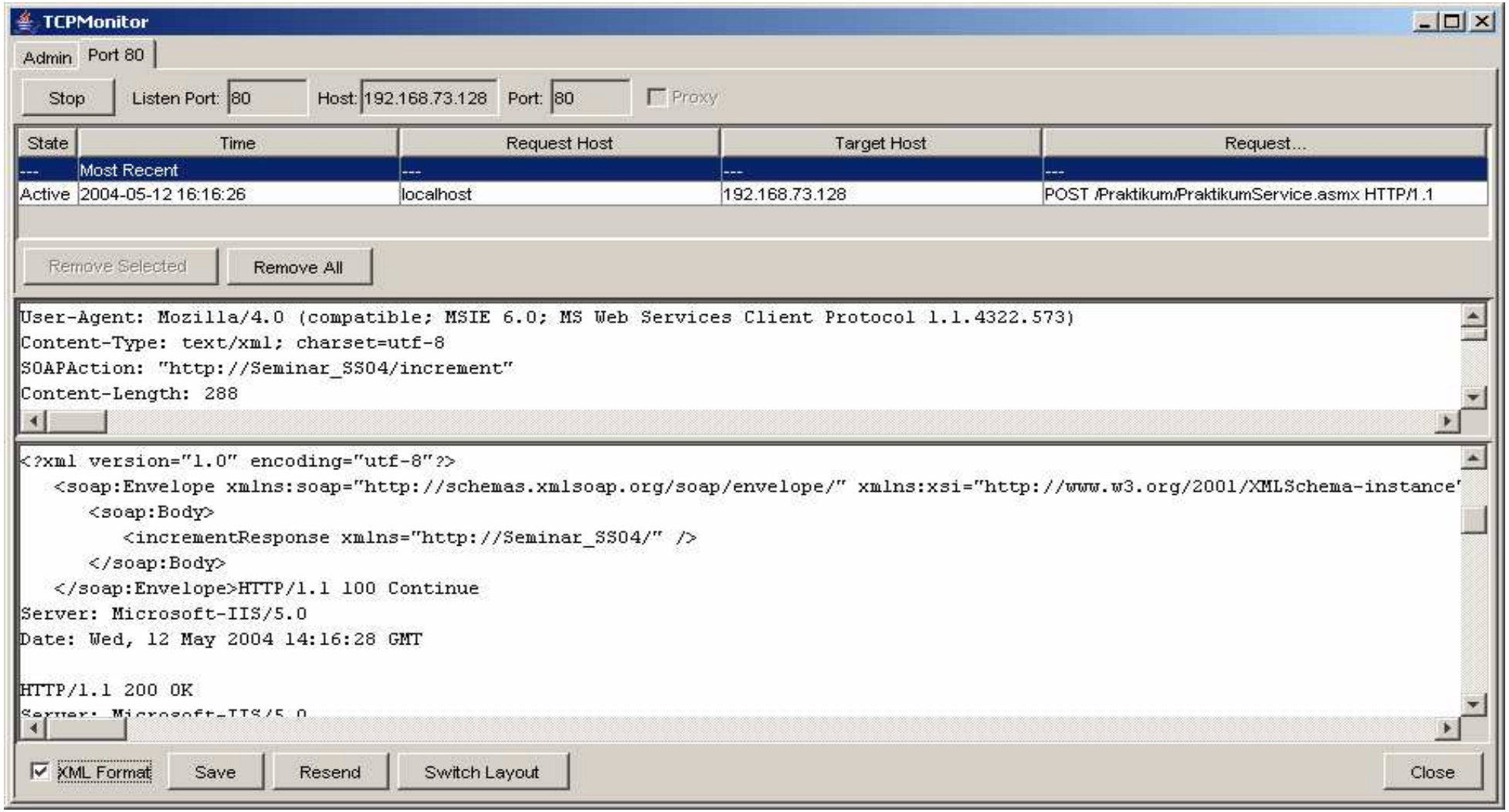
- ***Beispiel für die Suche eines Webservices per UDDI, die Untersuchung der Schnittstelle per WSDL und die Benutzung per SOAP.***

Webservice Beispiel: Aufbau



- gesamte Kommunikation wird als Text übertragen.
- TCPMonitor wird dazwischen geschalten und loggt die Übertragung (ist Teil des Axis Projekts)

Webservice Beispiel: TCPMonitor



Webservice Beispiel: Server (1)

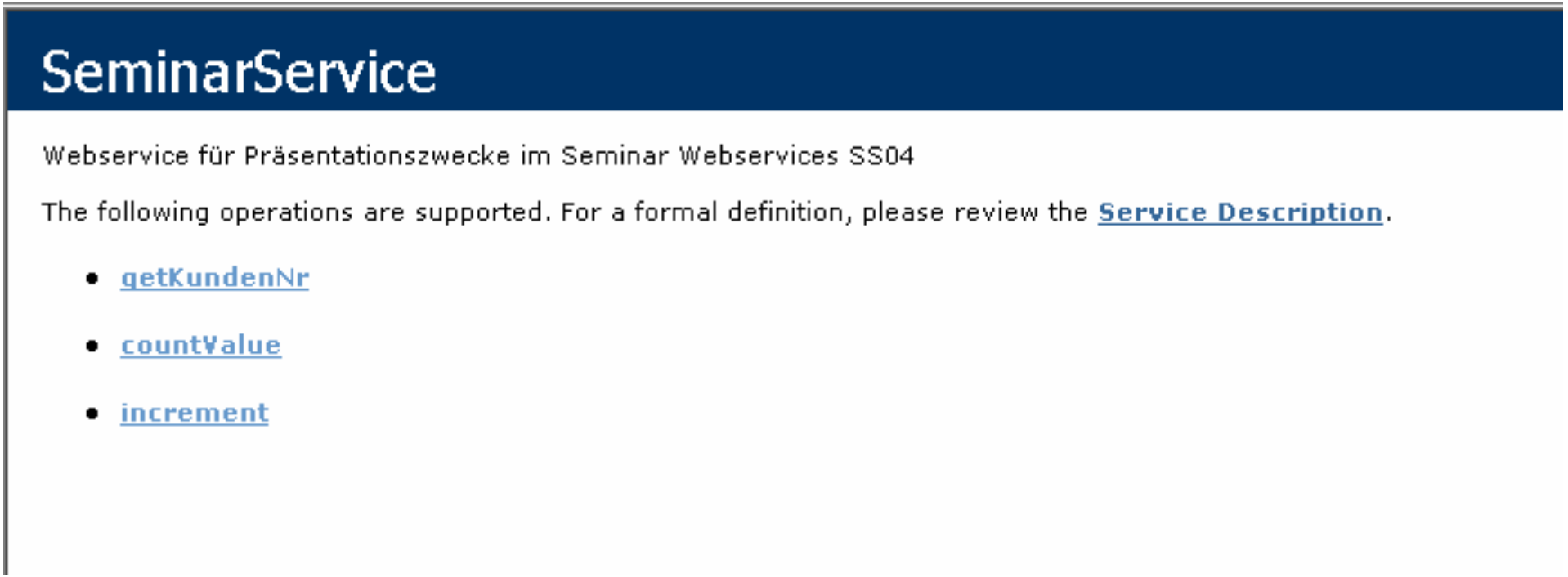
```
using System;
...

namespace Praktikum
{
  [WebService(Namespace="http://Seminar_SS04/",Description="Webservice
                  für Präsentationszwecke im Seminar Webservices SS04")]
  public class SeminarService : System.Web.Services.WebService
  {
    private int count = 0;
    ...
    [WebMethod]
    public void increment()
    { count++; }

    [WebMethod]
    public int countValue()
    { return count; }
  }
}
```

Webservice Beispiel: Server (2) Webpage

- Von Visual Studio .NET erstellte Webpage



- Bietet eine Beschreibung des Webservices und seiner Funktionen

Webservice Beispiel: Server (3) Webpage

- Von Visual Studio .NET erstellte Webpage für eine Methode mit Testfunktion (per HTTP Post)

SeminarService

Click here for a complete list of operations.

```
<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="http://Seminar_SS04/">MüllerK50001</string>
```

getKundenNr

Test

To test the operation using the HTTP POST protocol, click the 'Invoke' button.

Parameter	Value
kundenName:	

Invoke

SOAP

The following is a sample SOAP request and response. The **placeholders** shown need to be replaced with actual values.

```
POST /Praktikum/PraktikumService.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://Seminar_SS04/getKundenNr"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMI
```

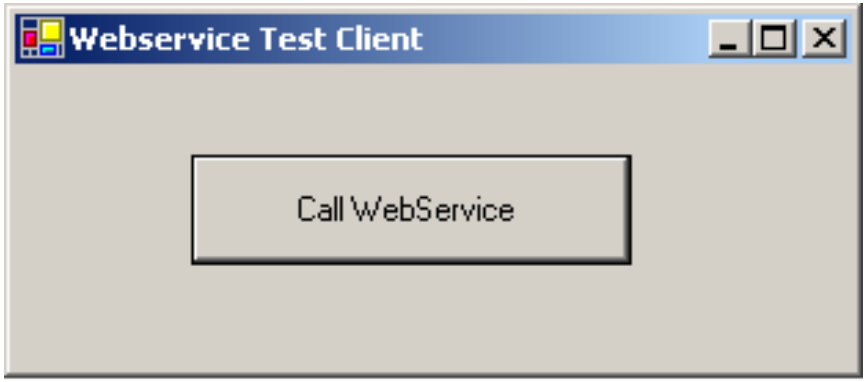
Webservice Beispiel: Client (1)

```
using System;
...

namespace PraktikumWinApp
{
   public class Form1 : System.Windows.Forms.Form
   {
     private System.Windows.Forms.Button button1;
     private WebReferenceNeu.SeminarService sWebS =
                                 new WebReferenceNeu.SeminarService();
     ...

     private void button1_Click(object sender, System.EventArgs e)
     {
       sWebS.increment();
       MessageBox.Show("count=" + sWebS.countValue().ToString(),"Info");
       sWebS.increment();
       MessageBox.Show("count=" + sWebS.countValue().ToString(),"Info");
     }
   }
}
```

Webservice Beispiel: Client (2)



Nach 1 x sWebS.increment();

Nach 2 x sWebS.increment();

- Webservice Funktionen werden oft als Klassen in die Programmiersprache eingebunden, stellen aber kein eigentliches Objekt dar (kein Zustand, ohne spezielle Implementierungen)

Webservice Beispiel: Kommunikation SOAP Anfrage (1)

```
POST /Praktikum/PraktikumService.asmx HTTP/1.1
VsDebuggerCausalityData: AwAAAMaH7L4RZ0FLgZQo1hqStKAAAAAAAAAA …
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; ...)
Content-Type: text/xml; charset=utf-8
SOAPAction: "http://Seminar_SS04/getKundenNr"
Content-Length: 334
Expect: 100-continue
Host: 192.168.73.128
```

HTTP

```
<?xml version="1.0" encoding="utf-8"?>
   <soap:Envelope
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <soap:Body>
         <getKundenNr xmlns="http://Seminar_SS04/">
            <kundenName>MÃ¼ller</kundenName>
         </getKundenNr>
      </soap:Body>
   </soap:Envelope
```

SOAP

Webservice Beispiel: Kommunikation SOAP Antwort (2)

```
HTTP/1.1 100 Continue
Server: Microsoft-IIS/5.0
Date: Wed, 12 May 2004 16:45:13 GMT

HTTP/1.1 200 OK / Server: Microsoft-IIS/5.0 /Date: Wed, 12 May...
X-AspNet-Version: 1.1.4322 / VsDebuggerCausalityData:...
Cache-Control: private, max-age=0
Content-Type: text/xml; charset=utf-8
Content-Length: 370
```

```
<?xml version="1.0" encoding="utf-8"?>
   <soap:Envelope
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <soap:Body>
         <getKundenNrResponse xmlns="http://Seminar_SS04/">
            <getKundenNrResult>MÃ¼llerK50001</getKundenNrResult>
         </getKundenNrResponse>
      </soap:Body>
   </soap:Envelope
```

Webservice Beispiel: Kommunikation HTTP Post

Anfrage:

```
POST /Praktikum/PraktikumService.asmx/getKundenNr HTTP/1.1
Host: 127.0.0.1
Content-Type: application/x-www-form-urlencoded
Content-Length: length

kundenName=string
```

Antwort:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<string xmlns="http://Seminar_SS04/">string</string>
```

Webservice Beispiel: WSDL (1) Namespace

```
<?xml version="1.0" encoding="utf-8"?>
<definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"

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

   xmlns:s="http://www.w3.org/2001/XMLSchema"

   xmlns:s0="http://Seminar_SS04/"

   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"

   xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"

   xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"

   targetNamespace="http://Seminar_SS04/„

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

Webservice Beispiel: WSDL (2) Types (XML Schema)

```
<types>
   <s:schema elementFormDefault="qualified"
                                  targetNamespace="http://Seminar_SS04/">
     <s:element name="getKundenNr">
       <s:complexType>
         <s:sequence>
           <s:element minOccurs="0" maxOccurs="1" name="kundenName"
                                                  type="s:string" />
         </s:sequence>
       </s:complexType>
     </s:element>
     <s:element name="getKundenNrResponse">
       <s:complexType>
         <s:sequence>
           <s:element minOccurs="0" maxOccurs="1"
                          name="getKundenNrResult" type="s:string" />
         </s:sequence>
       </s:complexType>
     </s:element>
   </s:schema>
 </types>
```

Parameter

Rückgabewert

xmlns:s="http://www.w3.org/2001/XMLSchema"

Webservice Beispiel: WSDL (3) Message + PortType

Messages:

Nachrichteninhalt

```
<message name="getKundenNrSoapIn">
  <part name="parameters" element="s0:getKundenNr" />
</message>
<message name="getKundenNrSoapOut">
  <part name="parameters" element="s0:getKundenNrResponse" />
</message>
```

„Vorgänge“ (hier Request−Response) :

„Vorgangsinhalt“

```
<portType name="SeminarServiceSoap">
  <operation name="getKundenNr">
    <input message="s0:getKundenNrSoapIn" />
    <output message="s0:getKundenNrSoapOut" />
  </operation>
</portType>
```

xmlns:s0="http://Seminar_SS04/"

Webservice Beispiel: WSDL (4) binding (SOAP binding)

```
<binding name="SeminarServiceSoap"
                                    type="s0:SeminarServiceSoap">
  <soap:binding
      transport="http://schemas.xmlsoap.org/soap/http"
                                    style="document" />
  <operation name="getKundenNr">
    <soap:operation
        soapAction="http://Seminar_SS04/getKundenNr"
                                    style="document„ />
    <input>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
  </operation>
</binding>
```

PortType

Transportart

Methode

Vorgaben für die SOAP Nachrichten

xmlns:s0="http://Seminar_SS04/"

Webservice Beispiel: WSDL (5) service

```
  <service name="SeminarService">
    <documentation>Seminar Webservices SS04</documentation>

    <port name="SeminarServiceSoap"
                                  binding="s0:SeminarServiceSoap">
      <soap:address
                location="http://127.0.0.1/...Service.asmx" />

    </port>
  </service>
</definitions>
```

Webservice wird das Binding zugewiesen

Schlussbetrachtung

- **Vorteile:**
  - Flexibel
  - Automatisierbar (WSDL, UDDI…)
  - Benutzte Standards
    - haben sich bewährt (z.B. HTTP)
    - werden kontinuierlich weiterentwickelt (bei XML: XPath, XQuery, XLink)
    - bieten viele Features (Typensystem von XML Schema, HTTPS…)
    - sind verbreitet und schon implementiert (XML Parser)
  - Kann eine Brücke zwischen .NET und Java sein
  - Firewall freundlich

- **Nachteile:**
  - Parsen von XML Dokumenten ist relativ langsam
  - Viel Overhead
  - Firewall freundlich
  - Verbreitung öffentlicher und kostenloser Webservices