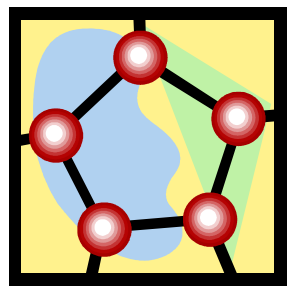
Web-Services

Dr. Herbert Prähofer
Institut für Systemsoftware
Johannes Kepler Universität Linz

Dr. Dietrich Birngruber
Software Architect
TechTalk
www.techtalk.at



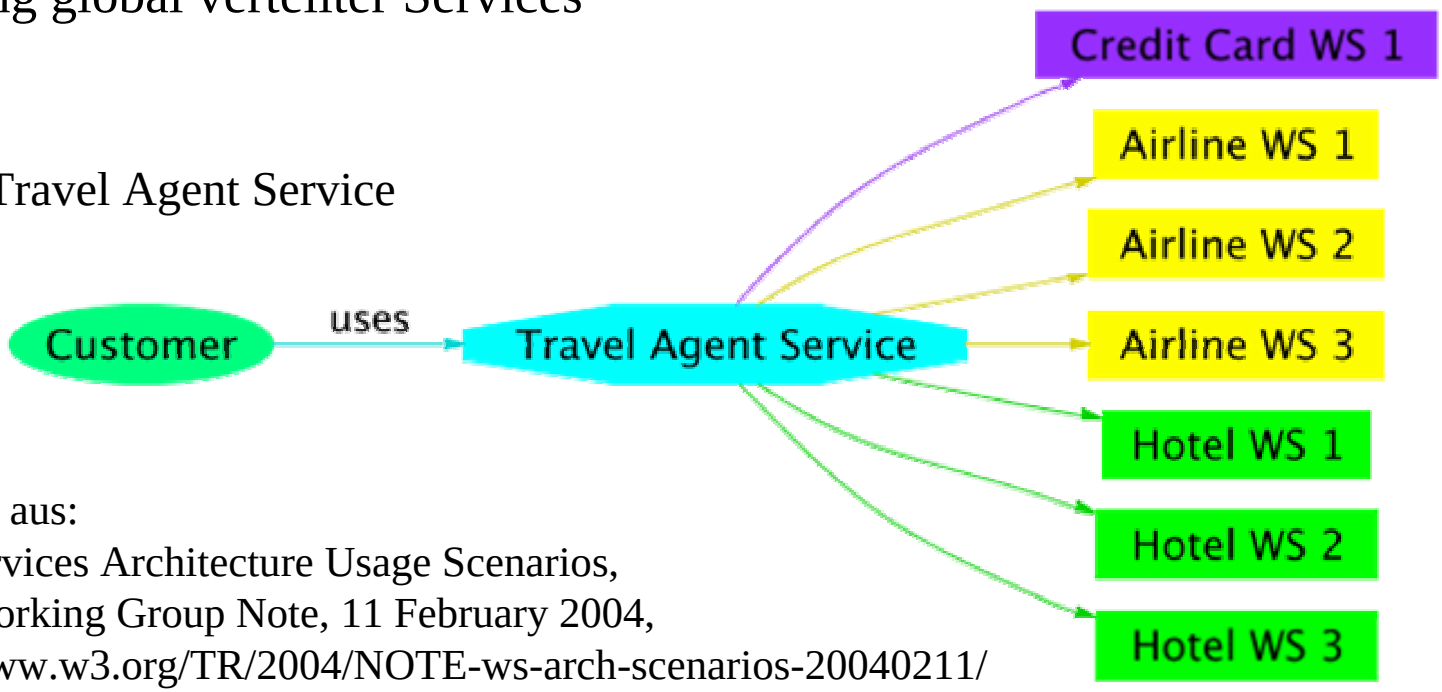
Web-Services

- Einführung
- Web-Services in .NET
- SOAP
- SOAP und .NET
- Dienstbeschreibung mit WSDL
- Web-Service Verzeichnisdienste: UDDI und DISCO
- Ausblick auf Web-Services .NET 2.0
- Zusammenfassung

Motivation

- Integration heterogener, verteilter Systeme
  - global verteilt
  - unterschiedliche Programmiersprachen
  - unterschiedliche APIs
- B2C und B2B Anwendungen
- Nützung global verteilter Services

Beispiel: Travel Agent Service



entnommen aus:

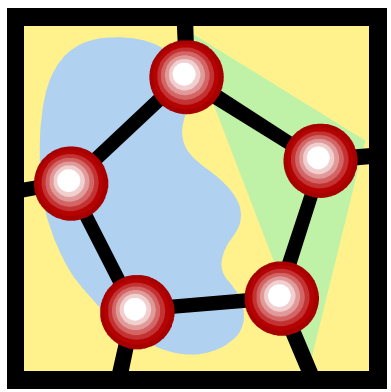
Web Services Architecture Usage Scenarios,
W3C Working Group Note, 11 February 2004,
http://www.w3.org/TR/2004/NOTE-ws-arch-scenarios-20040211/

Was sind Web-Services?

- Middleware für verteilte Anwendungen
- für Remote-Procedure-Calls und Datenaustausch
- offener Standard auf der Basis von XML
- für lose gekoppelte Softwaredienste
- unabhängig von Programmiersprache, Laufzeitumgebung und Betriebssystem
- Verwendung bestehender Internet-Protokolle und Server-Architekturen

Definition Web-Service (nach W3C)

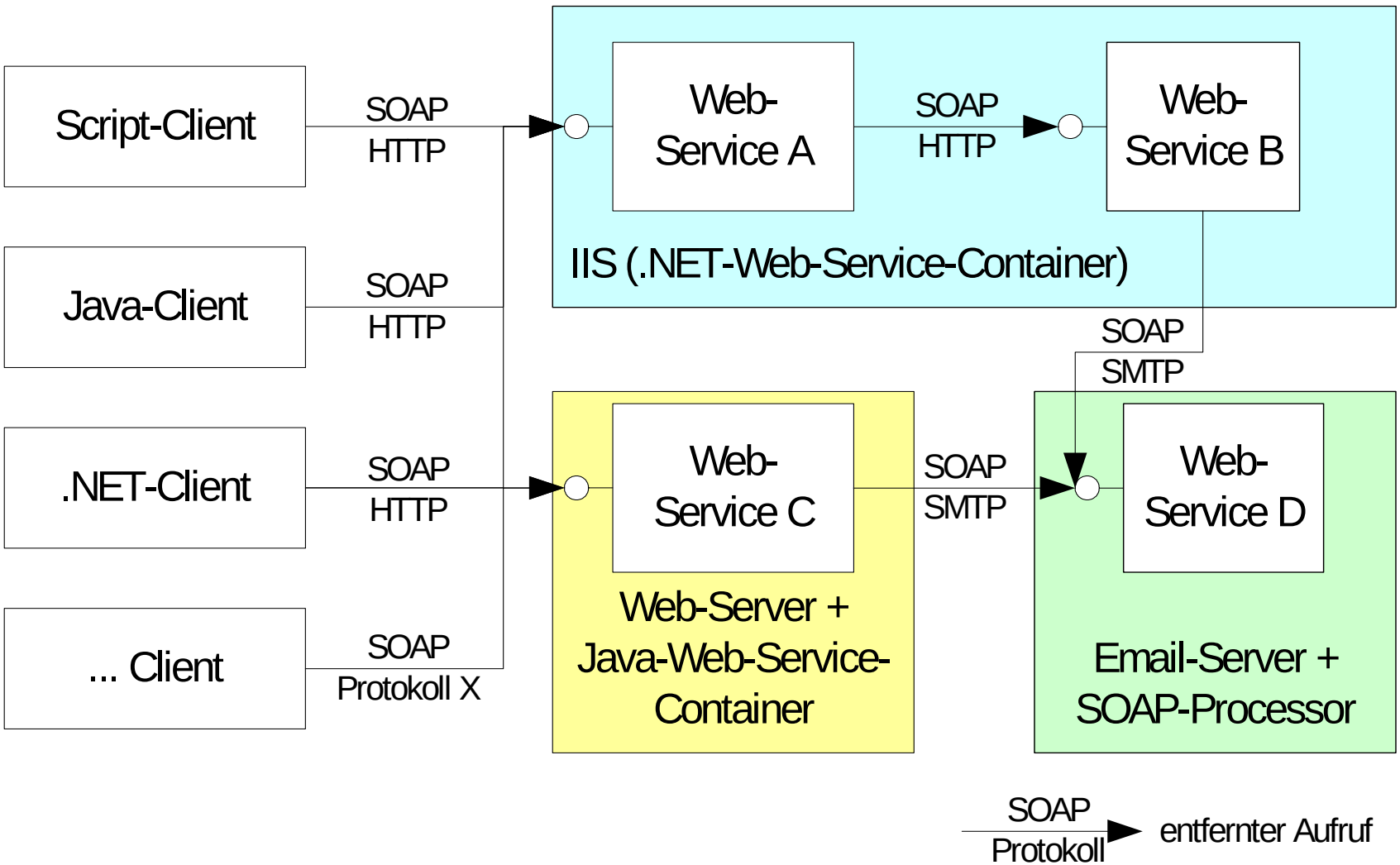
- durch eine URI identifizierbare Softwareanwendung
- mit Schnittstellendefinition in XML
- mit Interaktion auf Basis XML-basierter Nachrichten
- und Nachrichtenaustausch über Internetprotokolle



Unabhängigkeit und Integration durch ...

- SOAP
  - XML-Standard für Nachrichtenaustausch
  - unabhängig von Transportprotokoll
  - unabhängig von Client- und Service-Implementierung: Java, .NET, Python, …
- Web Services Description Language - WSDL (1.1)
  - Interface-Beschreibung in XML
- Kommunikation auf Basis existierenden Protokolle und Serverarchitekturen
  - HTTP und Web-Server
  - SMTP und Mail-Server
  - FTP und FTP-Server
- Standardisierung (W3C)
  - SOAP 1.2, WSDL 1.1 (1.2 und 2.0)
  - Zusätzliche Protokolle basierend auf SOAP and WSDL
  - Bindung an Protokollschicht (HTTP)

Web-Services Szenario



Web-Services im Vergleich

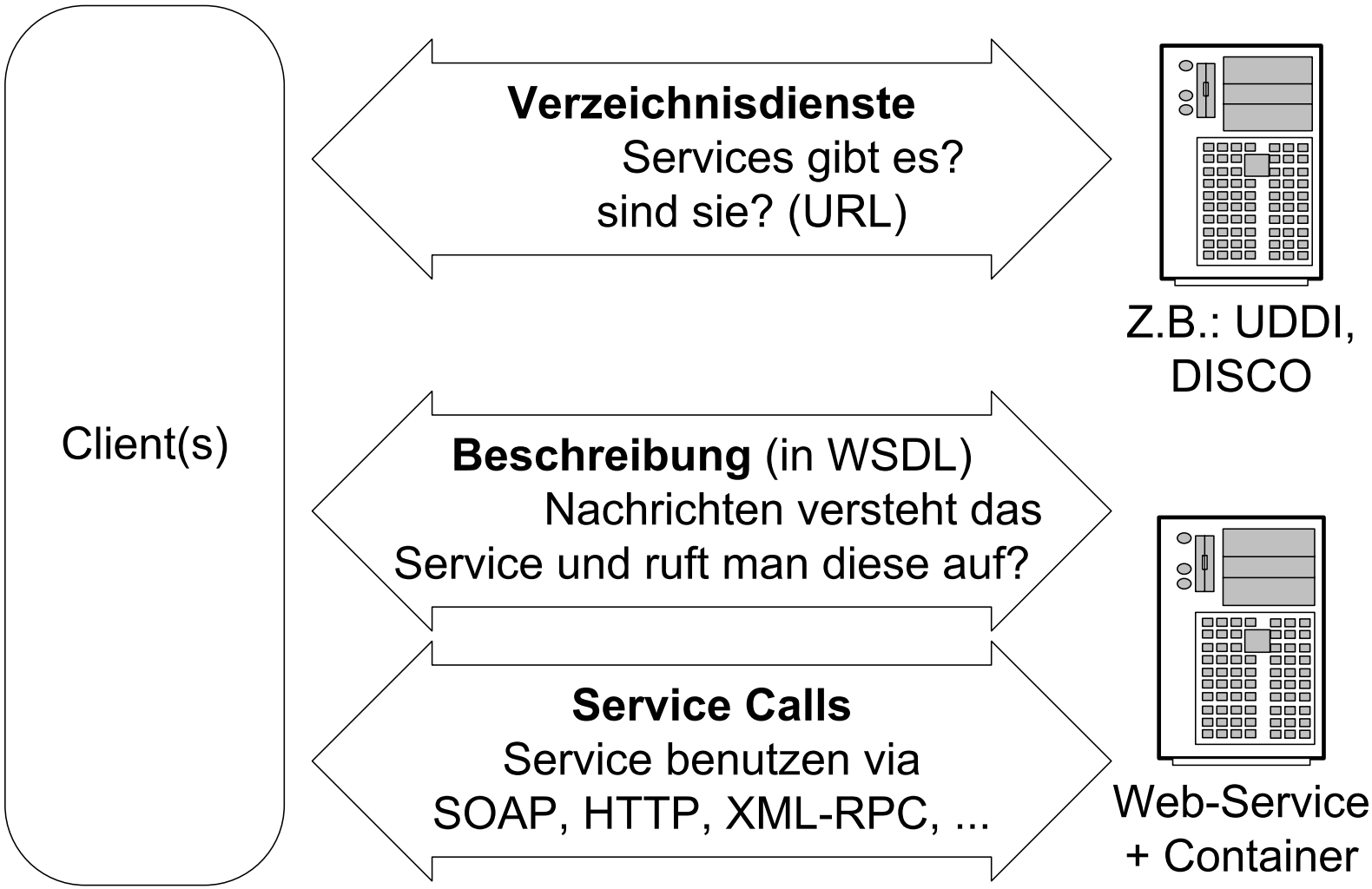
	Java RMI	**.NET Remoting**	**CORBA**	**Web-Services**
Programmier-sprache	Java	.NET Sprachen (C#, VB.NET, ..)	unabhängig	unabhängig
Schnittstellen-definition	Java Interfaces	C# Interfaces	CORBA IDL	WSDL (XML-basiert)
Datenstrukturen	Java Objekte	.NET Objekte	IDL-spezifizierte Objekte	XML-Daten
Übertragungs-protokoll	RMI-IIOP	binär oder SOAP	GIOP/IIOP	HTTP, HTTPS, SMTP, FTP
Verpackung	Java Objekt-Serialisierung	.NET Objekt-Serialisierung	ORB/CDR	SOAP
Infrastruktur	Java RMI-Infrastruktur	.NET Remoting-Infrastruktur	ORBs	Web-, Mail-, FTP-Server

Vor- und Nachteile

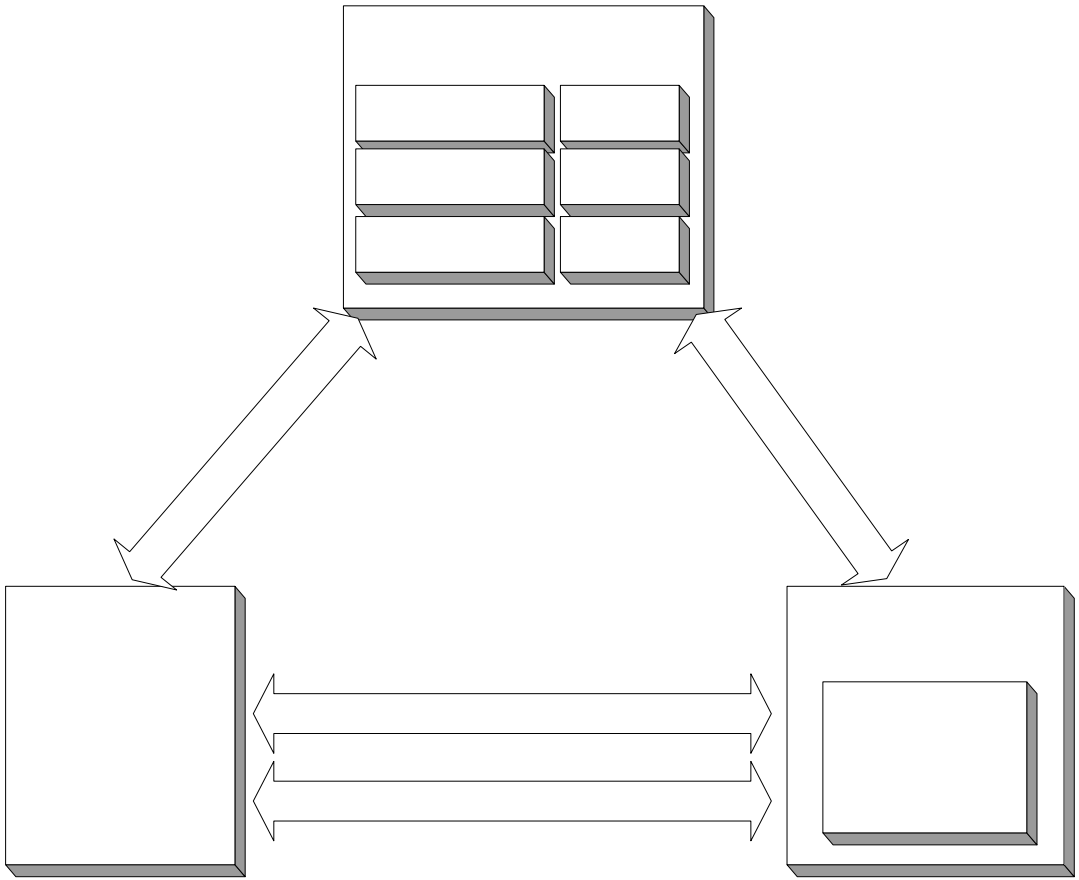
- Vorteile
  - unabhängig von Programmiersprache, Laufzeitumgebung und Betriebssystem
  - baut auf bestehender Internet-Infrastruktur auf
  - standardisiert
  - von wesentlichen Playern vorangetrieben (Microsoft, IBM, SAP, Sun)

- Nachteile
  - Performance (XML)

Web-Service-Infrastruktur



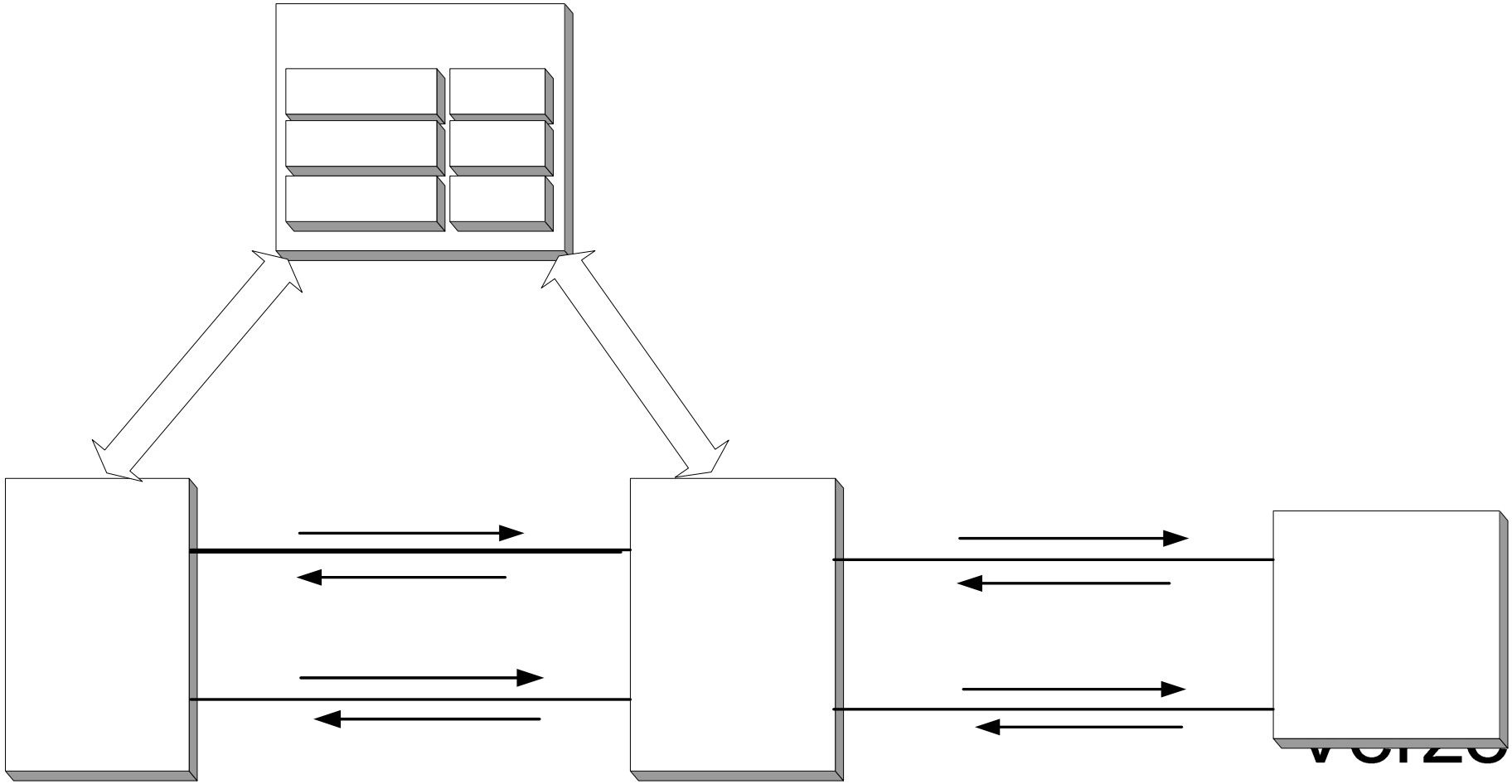
Web-Service Architektur



Verzei

WSD

Web-Service Architektur: Ablauf



Web-Services

- Einführung
- Web-Services in .NET
- SOAP
- SOAP und .NET
- Dienstbeschreibung mit WSDL
- Web-Service Verzeichnisdienste: UDDI und DISCO
- Ausblick auf Web-Services .NET 2.0
- Zusammenfassung

Web-Services in .NET

- IIS und ASP.NET-Infrastruktur unterstützen Web-Services
- .NET Framework bietet eine Reihe von
  - Basisklassen
  - Attributen
  - Protokollen

  für die Realisierung von Web-Services
- Visual Studio.NET bietet mächtige Werkzeuge zur Erstellung von Web-Services
  - Entwicklung
  - Testumgebung
  - Verwaltung des IIS
  - Erzeugung von Proxies (wsdl.exe)

.NET-Namensräume

- System.Web.Services
  - um Web-Services zu erstellen (z.B.: WebService, WebMethod)
- System.Web.Services.Configuration
  - um SOAP zu erweitern
- System.Web.Services.Description
  - um WSDL zu erstellen und zu bearbeiten
- System.Web.Services.Discovery
  - um DISCO zu benutzen
- System.Web.Services.Protocols
  - zur Implementierung von Protokollen (z.B. SOAP-HTTP)
- System.Xml.Serialization
  - zur XML-Serialisierung bei Datentransfer

Implementierung von Web-Services

- in asmx-Datei mit @WebService-Direktive

```
<%@ WebService Language="C#" Class=„MyWebService" %>
```

- ableiten von Basisklasse System.Web.Services.WebService

```
public class MyWebService : WebService {
```

- Kennzeichnung und Einstellungen über .NET-Attribute
  - Kennzeichnung von Web-Service-Methoden
  - Festlegen von Nachrichtenformat und Kodierung
  - zu verwendende Namensräume und XML-Elemente
  - etc.

```
[WebMethod(Description= “comment ")]
[…]
public Returntype MyWebMethod( …) {
    …
```

asmx-Datei in einem virtuellen Verzeichnis auf dem IIS

Beispiel: TimeService

TimeService**.asmx**

gespeichert in virtuellen Verzeichnis
time/TimeService.asmx

```
<%@ WebService Language="C#" Class="TimeService" %>

using System;
using System.Web.Services;

public class TimeService : WebService {

   [WebMethod(Description="Returns the current time")]
   public string GetTime(bool shortForm) {
      if (shortform) return DateTime.Now.ToShortTimeString();
      else return DateTime.Now.ToLongTimeString();
   }
   …
}
```

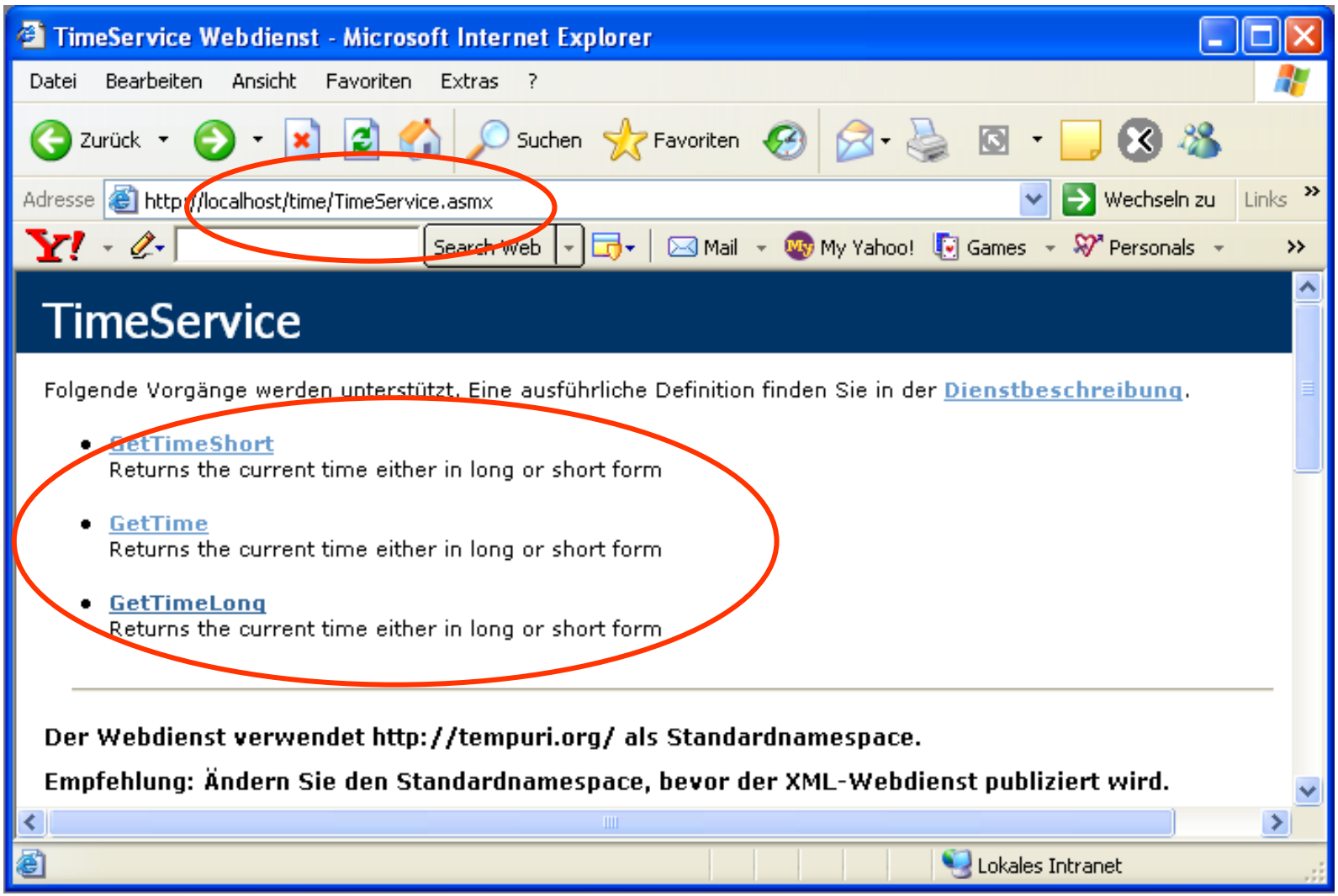
WebService-Direktive

WebService erweitern

Attribut [WebMethod] deklariert Web-Service-Methode

TimeService Webdienst

WebService-Beschreibung im IE



Beispiel: Einfacher .NET-Client

- wsdl.exe erzeugt Proxy für Client (TimeClient.TimeService)

```
> wsdl.exe /namespace:TimeClient  /out:TimeServiceProxy.cs
  http://localhost/netsem-ws/MyFirstService.asmx
```

- Client-Programm erzeugt TimeService-Objekt und ruft GetTime auf

```
using System;
using TimeClient; //Namespace des erzeugten Proxies

public class NetClient {
   public static void Main(string[] args) {
      TimeService service = new TimeService();
      Console.WriteLine("Die Zeit am Server ist: ");
      string time = service.GetTime(true);
      Console.WriteLine(time);
   }
}
```

Beispiel: Einfacher Java-Client

- Verwendung von Werkzeug und Java-Bibliotheken GLUE:
  - wsdl2Java erzeugt Java Interface (ITimeServiceSoap) und Proxy (TimeServiceHelper)

```
import Kapitel7.GLUEProxy.*; // erzeugte GLUE Proxy-Klassen
/** Einfacher XML Web Service client in Java welcher GLUE verwendet. */
public class JavaClientGLUE {
   public static void main(String[] args) {
      try {
         //Service benutzen mittels wsdl2java erzeugten Klasse + Interface
         ITimeServiceSoap service = TimeServiceHelper.bind();
         String time = service.GetTime(true);
         System.out.println("Die Zeit am Server ist: \n" + time);
      } catch (Exception ex) {  ex.printStackTrace(); }
   }
}
```

Web-Services

- Einführung
- Web-Services in .NET
- SOAP
- SOAP und .NET
- Dienstbeschreibung mit WSDL
- Web-Service Verzeichnisdienste: UDDI und DISCO
- Ausblick auf Web-Services .NET 2.0
- Zusammenfassung

SOAP

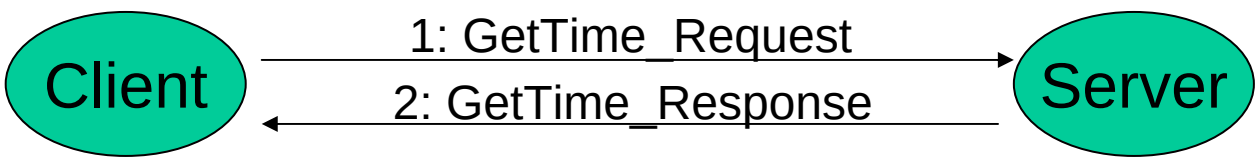
- einfaches Nachrichtenprotokoll in XML
  - für Verpackung beliebiger Anwendungsdaten
  - einzelne Nachrichten („one-way“)
  - asynchron
- Unabhängig vom Transportprotokoll

- SOAP definiert nicht:
  - verteiltes Objektmodell
  - Kommunikationsprotokolle
  - Distributed Garbage Collection
  - verteilte Ereignisse (distributed callbacks)

Verwendung von SOAP

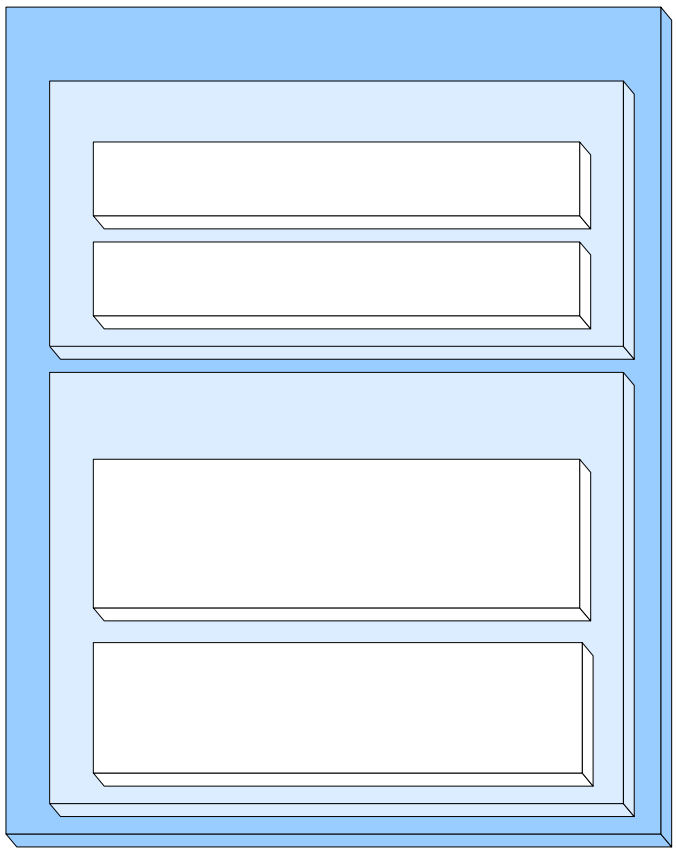
- SOAP ist erweiterbar
  - Methodenaufrufe (RPC)
  - Sicherheit
  - Authentifikation
  - etc.
- Protokolle durch Kombination einzelner Nachrichten (*Message Exchange Patterns*)
  - one-way, request-response, multicast, …

z.B. request-response für RPC durch 2 Nachrichten



SOAP-Nachrichten

SOAP-Nachrichten sind vergleichbar mit einem Brief, mit



- Umschlag (<Envelope>) als Behälter
- Briefkopf (<Header>)
  mit beliebigen Metainformationen
  (*Message Headers*)
- Brief (<Body>)
- mit beliebigen XML-Daten
- Fehlerbeschreibungen

XML-Struktur (vereinfacht, SOAP 1.2)


```
<?xml version="1.0" ?>
<soap:Envelope xmlns:soap="...">
  <soap:Header> <!-- (optional and extendable)  -->
    <m:my xmlns:m="anURI" soap:mustUnderstand="true" soap:role="uri2" />
    ...
  </soap:Header>

  <soap:Body>
    data (depends on format and encoding)
    <soap:Fault>
      <soap:Code>...who is responsible?... </Code>
      <soap:Reason>...textual description...</soap:Reason>
      <soap:Detail>...more error details...</soap:Detail>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

Die Daten im <Body>-Element

- **Nachrichtenformat**:
  - `document` Struktur entspricht einem *XML-Schema*
  - `rpc` Struktur entspricht der *SOAP-Spezifikation für RPC*
- **Kodierung**:
  - `literal` Daten werden nach einem *XML-Schema* kodiert
  - `encoded` Daten werden nach *SOAP-Regeln* kodiert
- Übliche Kombinationen:
  - `document/literal` Standard für .NET
  - `rpc/encoded` oft verwendet bei Java-Server

HTTP-Anbindung

- HTTP-GET, HTTP-POST
  - Aufruf HTTP-kodiert (*URL-encoded*)
  - Response XML-kodiert

  ➔Einschränkung auf einfache Aufrufe (keine Header-Einträge, keine strukturierten Daten)

- SOAP über HTTP-POST
  - Datenteil des POST-Request enthält SOAP-kodierte Anfrage
  - Antwort SOAP-kodiert

  ➔ ohne Einschränkungen

Beispielaufruf via HTTP-GET bzw. -POST

Aufruf von GetTime(bool shortForm) bei Web-Service
http://localhost/time/TimeService.asmx

- Aufruf :

http://localhost/time/TimeService.asmx/GetTime?shortForm=true

- HTTP-Response:

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://tempuri.org/">***string***</string>

Beispielaufruf via SOAP über HTTP

- SOAP über HTTP-POST:

```
POST /time/TimeService.asmx HTTP/1.1
Content-type: text/xml; charset=utf-8
SOAPAction: http://dotnet.jku.at/GetTime
Content-length: 198
User-Agent: Java1.4.0
Host: localhost
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
        <soap:Body>
                <GetTime xmlns="http://tempuri.org/"
                        <shortForm>
                                true
                        </shortForm>
                < /GetTime>
        </soap:Body>
</soap:Envelope>
```

HTTP-Header SOAPAction identifiziert SOAP-Request

Web-Methode

Parameter

Beispielaufruf via SOAP über HTTP (2)

- SOAP-kodierte Antwort:

```
HTTP/1.1 200 OK Content-Type: text/xml; charset=utf-8 Content-Length: length
<?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>
            <GetTimeResponse xmlns="http://tempuri.org/">
                  <GetTimeResult>string</GetTimeResult>
            </GetTimeResponse>
      </soap:Body>
</soap:Envelope>
```

Rückgabe-wert

Web-Services

- Einführung
- Web-Services in .NET
- SOAP
- SOAP und .NET
- Dienstbeschreibung mit WSDL
- Web-Service Verzeichnisdienste: UDDI und DISCO
- Ausblick auf Web-Services .NET 2.0
- Zusammenfassung

SOAP und .NET

.NET bietet bequeme Unterstützung für

- Bestimmen von Nachrichtenformat und Kodierung
- Kodierung von .NET-Datentypen
- Entwickeln von Header-Einträgen
- Handhabung des Lebenszyklus von Web-Services

Nachrichtenformat und Kodierung (1)


```
[SoapRpcMethod(
      Use=SoapBindingUse.Encoded
      Action="http://dotnet.jku.at/Sample/AddAddressRpc",                 // SOAPAction
      RequestNamespace="http://dotnet.jku.at/Sample/Request",
      RequestElementName="AddAddressRpcRequest",                          // SOAP element name
      ResponseNamespace="http://dotnet.jku.at/Sample/Response",
      ResponseElementName="AddAddressRpcResponse")]                       // SOAP element name
[WebMethod(Description="Adds an address DataSet for the specified user")]
public void AddAddressRpc(long userID, Address address) { ... }
```

- Attribute SoapRpcService und SoapRpcMethod für rpc-Format
- mit Parametern
  - Use: Kodierung (SoapBindingUse.Literal oder SoapBindingUse.Encoded)
  - Action: URI für SOAPAction-HTTP-Header
  - RequestNamespace, RequestElementName:
    Namensraum und Name des SOAP-Elements für Request
  - ResponseNamespace und ResponseElementName:
    Namensraum und Name des SOAP-Elements für Response

Nachrichtenformat und Kodierung (2)

```
[SoapDocumentMethod(Use=SoapBindingUse.Literal,
      Action="http://dotnet.jku.at/Sample/AddAddressDocLit",          // SOAPAction
      RequestNamespace="http://dotnet.jku.at/Sample/Request",
      RequestElementName="AddAddressDocLitRequest",                   // SOAP element name
      ResponseNamespace="http://dotnet.jku.at/Sample/Response",
      ResponseElementName="AddAddressDocLitResponse")]                // SOAP element name
[WebMethod(Description="Adds an address DataSet for the specified user")]
public void AddAddressDocLit(long userID, Address address) { ... }
```

```
[SoapDocumentService(Use=SoapBindingUse.Encoded)]
public class TimeService : WebService { ... }
```

- Attribute SoapDocumentService und SoapDocumentMethod für document-Format

SOAP-Kodierung von .NET-Datentypen

- Serialisierung von .NET-Datentypen
  - auf Basis von Kodierungsregeln
    http://schemas.xmlsoap.org/soap/encoding
  - angepasst durch Attribute (Namespace System.Web.Serialization)

SoapAttributeAttribute	Serialisierung des Feldes als XML-Attribut
SoapElementAttribute	Serialisierung des Feldes als XML-Element
SoapIgnoreAttribute	Keine Serialisierung des Feldes
SoapIncludeAttribute	Inkludieren eines abgeleiteten Typs ins Schema
SoapEnumAttribute	Anpassen eines Namens einer Enumeration

Beispiel: Kodierung eines Datentyps (1)

- WebMethod GetTimeDesc verwendet für Rückgabewert Typ TimeDesc

```
[WebMethod(Description="Returns the time description of the server")]
public TimeDesc GetTimeDesc() {
   TimeDesc td = new TimeDesc();
   // ...
   return td;
}
```

- Kodierung von TimeDesc angepasst durch Attribut [SoapAttribute]
  → Felder kodiert als XML-Attribute

```
public struct TimeDesc {
   [SoapAttribute] public string TimeLong;
   [SoapAttribute] public string TimeShort;
   [SoapAttribute (AttributeName = "ZoneID")] public int TimeZone;
}
```

Beispiel: Kodierung eines Datentyps (2)

- SOAP-kodierte Antwort

```
...
<soap:Envelope  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" ...
   <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <types:GetTimeDescResponse>
            <GetTimeDescResult href="#id1" />
        </types:GetTimeDescResponse>
        <types:TimeDesc id="id1" xsi:type="types:TimeDesc"
            types:TimeLong="10:00:25" types:TimeShort="10:00" types:ZoneID="1" />
   </soap:Body>
</soap:Envelope>
```

Inkludieren von abgeleiteten Typen

- SoapIncludeAttribute erlaubt Aufnahme von Typen
  ➔ Wichtig bei Spezialisierungen

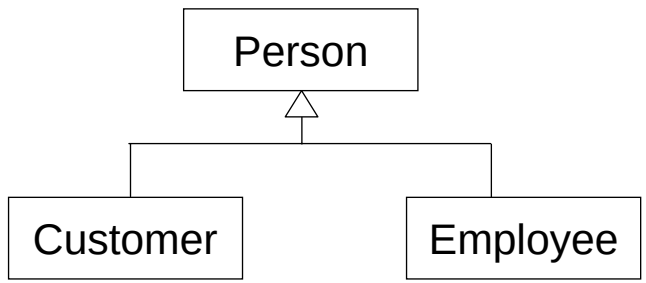
Beispiel: PersonService

Web-Method mit Rückgabewert vom Typ Person

```
public class PersonService : WebService {

    [WebMethod]…
    public Person[ ] GetAll() {…}
}
```

Person hat 2 Spezialisierungen Customer und Employee



➔ Customer und Employee müssen explizit in die Web-Beschreibung aufgenommen werden !

Beispiel: PersonService (1)

- Klassen Person, Customer und Employee

```
public abstract class Person { …}
public class Customer : Person { …}
public class Employee : Person {…}
```

- PersonService definiert Web-Method GetAll mit Rückgabetyp Person[]
- Mit SoapInclude werden Customer und Employee inkludiert

```
<%@ WebService Language="C#" Class="PersonService" %>
using System; … using System.Xml.Serialization;
public class PersonService : WebService {

    [WebMethod] [SoapRpcMethod]
    [SoapInclude(typeof(Customer)), SoapInclude(typeof(Employee))]
    public Person[] GetAll() {
        Person[] data = new Person[2];
        data[0] = new Customer("1“, "Max Customer", "EMP-33");
        data[1] = new Employee("EMP-33", "Tom Employee");
        return data;
    }
}
```

Beispiel: PersonService (2)

- SOAP-kodierte Antwort

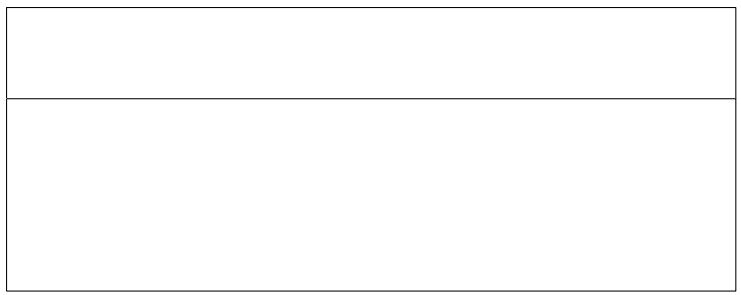
```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" ... >
    <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <tns:GetAllResponse>
            <GetAllResult href="#id1" />
        </tns:GetAllResponse>
        <soapenc:Array id="id1" soapenc:arrayType="types:Person[2]">
            <Item href="#id2" />
            <Item href="#id3" />
        </soapenc:Array>
        <types:Customer id="id2" xsi:type="types:Customer">
            <SSN xsi:type="xsd:string">1</SSN>
            <Name xsi:type="xsd:string">Max Customer</Name>
            <EmpSSN xsi:type="xsd:string">EMP-33</EmpSSN>
        </types:Customer>
        <types:Employee id="id3" xsi:type="types:Employee">
            <SSN xsi:type="xsd:string">EMP-33</SSN>
            <Name xsi:type="xsd:string">Tom Employee</Name>
        </types:Employee>
    </soap:Body>
</soap:Envelope>
```

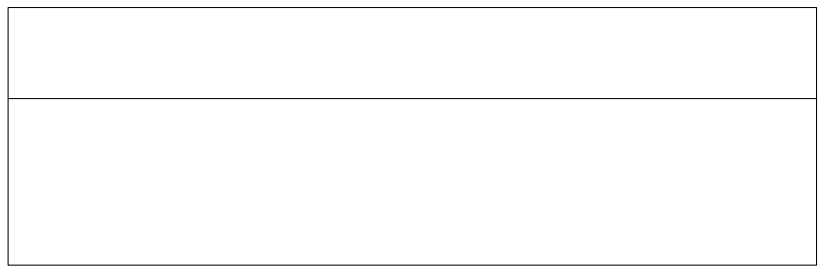
SOAP-Header-Einträge

- SOAP-Headers für Metainformationen zu Nachrichten
- Beliebige Header-Einträge möglich
- Alle Header-Einträge haben Attribute
  - Empfänger (Actor)
  - ob Header vom Empfänger behandelt werden muss (mustUnderstand)

- .NET unterstützt Header-Einträge durch
  - Klasse SoapHeader: Entwicklung von Header-Klassen
  - Attribut SoapHeaderAttribute: Setzen von Headers bei Web-Methoden

Klassen SoapHeader und SoapHeaderAttribute





- Empfänger der Nachricht

```
public string Actor {get; set;}
```

- Header muss bearbeitet werden

```
public bool MustUnderstand  {get; set;}
```

- Header erfolgreich bearbeitet

```
public bool DidUnderstand  {get; set;}
```

- In-, Out-, InOut-Richtung des Headers

```
public SoapHeaderDirection Direction
        {get; set;}
```

- Name des Feldes der Webdienstklasse für diesen Header

```
public string MemberName{get; set;}
```

Beispiel: AuthHeader (1)

Benutzeridentifikation bei TimeService

- Login liefert Identifikation (cookie)
- GetTime muss Identifikation im Header zurückschicken

- Header-Klasse AuthHeader definiert public-Feld cookie

```
public class AuthHeader : SoapHeader { public string cookie; }
```

- Web-Service-Klasse definiert Feld curUser, um AuthHeader zu speichern

```
[WebService(Namespace="http://dotnet.jku.at/time/")]
public class HeaderTimeService : WebService {
    public AuthHeader curUser;
```

- Login mit User und Passwort liefert Indentifikations-String

```
    [WebMethod (Description="Authenticates the user")]
    public string Login(string user, string pwd) { ... create cookie ... }

    bool ValidateCookie(string cookie) { ... validate cookie ... }
```

Beispiel: AuthHeader (2)

- GetTime verlangt AutHeader Header-Eintrag (wird in Feld "curUser" gespeichert)
- Validiert curUser auf Basis der Login-Daten

```
[WebMethod(Description="Returns the current time")]
[SoapHeader("curUser")]
public string GetTime() {
    if (curUser != null && ValidateCookie(curUser.cookie))
        return System.DateTime.Now.ToLongTimeString();
        else throw new SoapHeaderException("Access denied!",
                                           SoapException.ClientFaultCode);
}
```

Beispiel: AuthHeader (3)

- Client
  - erzeugt Service-Proxy und AutHeader-Objekt

```
HeaderTimeService proxy = new HeaderTimeService();
AuthHeader entry = new AuthHeader();
```

  - erhält cookie über Login

```
entry.cookie = proxy.Login(user, pwd);
```

  - setzt AuthorHeader bei Proxy
  - ruft GetTime mit AuthHeader-Eintrag auf

```
entry.cookie = proxy.Login(user, pwd);
proxy.AuthHeaderValue = entry;
Console.WriteLine(proxy.GetTime());
```

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://..." ... >
  <soap:Header>
    <AuthHeader xmlns="http://dotnet.jku.at/time/">
        <cookie>aewt12348cvNNgrt55</cookie>
    </AuthHeader>
  </soap:Header>
  <soap:Body>
    <GetTime xmlns="http://dotnet.jku.at/time/" />
  </soap:Body>
</soap:Envelope>
```

Lebenszyklus

- Web-Service-Objekte sind zustandslos
  - werden bei jedem Methodenaufruf neu erzeugt
- Daten können in Properties von
  - Application-Zustandsobjekt oder

```
public HttpApplicationState Application {get;}
```

  - Sesssion-Zustandsobjekt

```
public HttpApplicationState Session {get;}
```

gespeichert werden

```
public sealed class HttpSessionState : ICollection, IEnumerable {
    public object this[ string name ] {get; set;}
    public object this[ int index ] {get; set;}
    …
}
```

Beispiel: StateDemo (1)

- Web-Service StateDemo demonstriert Speicherung von Daten

```
<%@ WebService Language="C#" Class="StateDemo" %>
using System.Web.Services;
[WebService(Namespace="http://dotnet.jku.at/StateDemo/")]
public class StateDemo : WebService {
```

- Methode IncApplication erhöht Property "Hit" bei Application-Objekt

```
    [WebMethod()]
    public int IncApplication() {
        int hit = (Application["Hit"] == null) ? 0 : (int) Application["Hit"];
        hit++;
        Application["Hit"] = hit;
        return hit;
    }
```

Beispiel: StateDemo (2)

- Parameter EnableSession ermöglicht Sessions
- IncSeesion erhöht Property "Hit" bei Session-Zustandsobjekt

```
    [WebMethod(EnableSession=true)]
    public int IncSession() {
        int hit = (Session["Hit"] == null) ? 0 : (int) Session["Hit"];
        hit++;
        Session["Hit"] = hit;
        return hit;
    }
}
```

Web-Services

- Einführung
- Web-Services in .NET
- SOAP
- SOAP und .NET
- Dienstbeschreibung mit WSDL
- Web-Service Verzeichnisdienste: UDDI und DISCO
- Ausblick auf Web-Services .NET 2.0
- Zusammenfassung

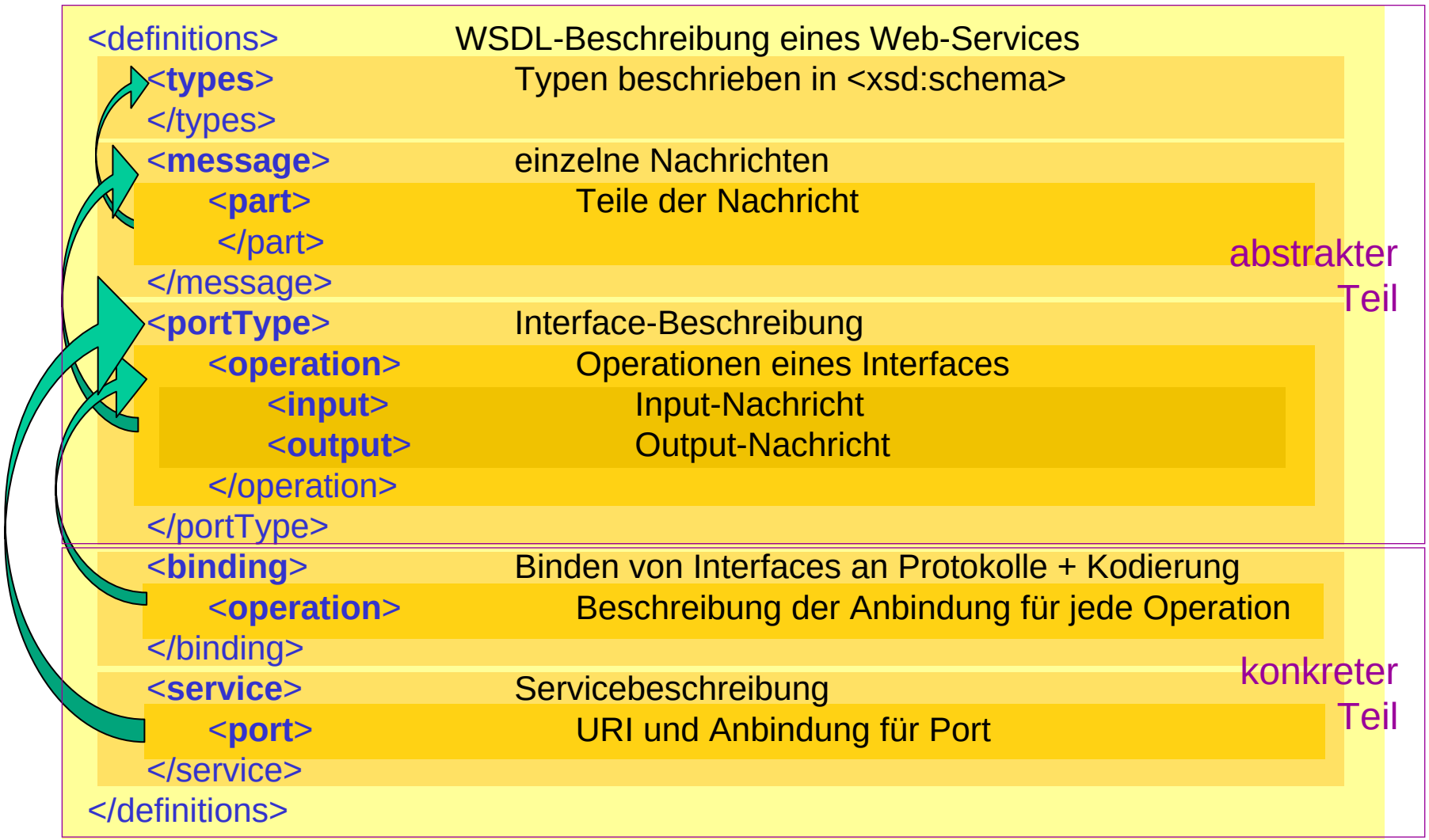
Web Service Description Language (WSDL)

- WSDL ist XML-basierte IDL für Web-Services
- in WSDL wird beschrieben:
  - Verwendete Datentypen
  - Aufbau einzelner Nachrichten
  - Operationen (Methoden)
  - Protokolle um Operationen aufzurufen
  - Adressen für Web-Service

aktuelle Version in .NET: WSDL 1.1 (http://schemas.xmlsoap.org/wsdl/)

Working Draft: WSDL 2.0 (4. 10. 2004)

Grundstruktur von WSDL 1.1



Beispiel: WSDL für TimeService (1)

- WSDL-Beschreibungen werden von Web-Container erzeugt

http://localhost/simple/TimeService.asmx**?WSDL**

```
<?xml version="1.0" encoding="utf-8"?>
<definitions xmlns:soap=http://schemas.xmlsoap.org/wsdl/soap/
        xmlns:tns="http://dotnet.jku.at/time/"
        xmlns:s="http://www.w3.org/2001/XMLSchema“
        xmlns:http="http://schemas.xmlsoap.org/wsdl/http/“
        xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/“
        xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/“
        targetNamespace="http://dotnet.jku.at/time/"
        xmlns="http://schemas.xmlsoap.org/wsdl/">
    <types />
    <message name="GetTimeSoapIn" />
    <message name="GetTimeSoapOut">
        <part name="GetTimeResult" type="s:string" />
    </message>
    <portType name="TimeServiceSoap">
        <operation name="GetTime">
            <input message="tns:GetTimeSoapIn" />
            <output message="tns:GetTimeSoapOut" />
        </operation>
    </portType>
    …
```

abstrakter Teil

Beispiel: WSDL für TimeService (2)

```
    …
    <binding name="TimeServiceSoap" type="tns:TimeServiceSoap">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc" />
        <operation name="GetTime">
            <soap:operation soapAction="http://dotnet.jku.at/time/GetTime" style="rpc" />
            <input>
                <soap:body use="encoded" namespace="http://dotnet.jku.at/time/“
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
            </input>
            <output>
                <soap:body use="encoded" namespace="http://dotnet.jku.at/time/“
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
            </output>
        </operation>
    </binding>
    <service name="TimeService">
        <documentation>Simple web service for querying the time</documentation>
        <port name="TimeServiceSoap" binding="tns:TimeServiceSoap">
            <soap:address location="http://localhost/time/TimeService.asmx" />
        </port>
    </service>
</definitions>
```

konkreter Teil

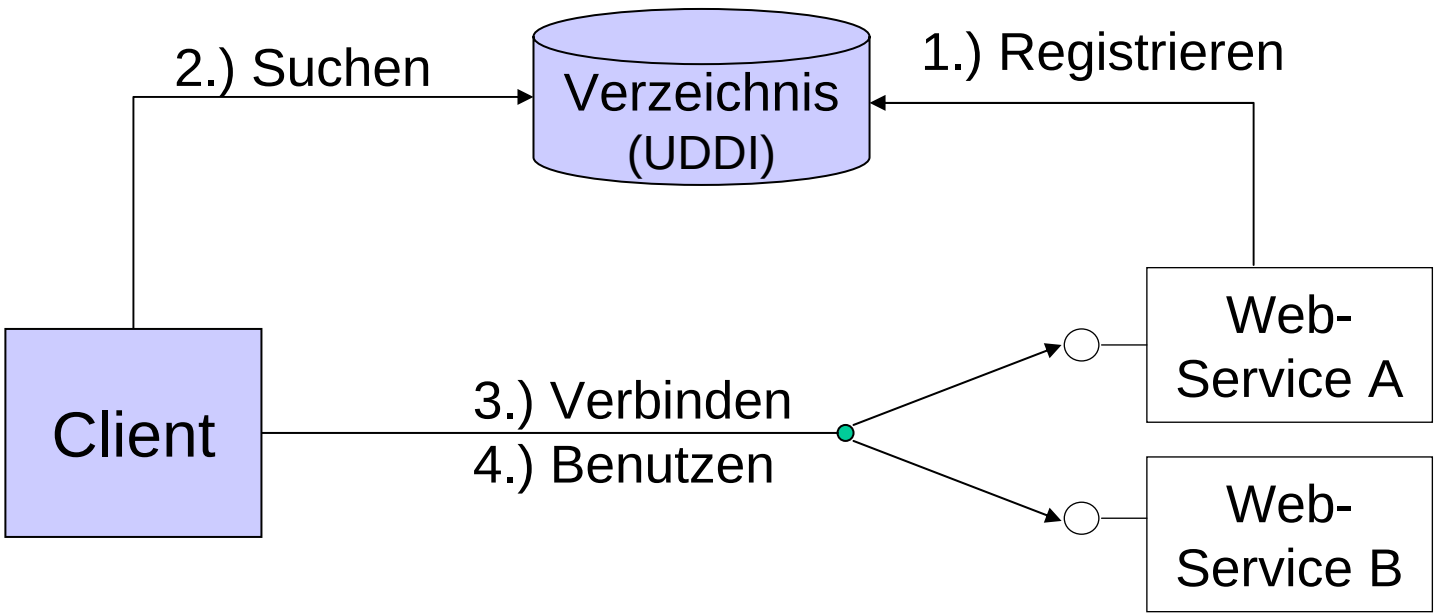
Web-Services

- Einführung
- Web-Services in .NET
- SOAP
- SOAP und .NET
- Dienstbeschreibung mit WSDL
- Web-Service Verzeichnisdienste: UDDI und DISCO
- Ausblick auf Web-Services .NET 2.0
- Zusammenfassung

Universal, Description, Discovery and Integration (UDDI)

http://www.uddi.org

- einheitliches Protokoll, um Web-Services zur Laufzeit zu suchen und zu benutzen
- Verwendung über Web-Services-Schnittstelle



DISCO

- Microsoft-Technik für dynamische Nutzung von Web-Services
- DISCO-Datei
  - enthält XML-Dokument mit URIs zu Web-Service und WSDL
  - kann Ergebnis einer UDDI-Abfrage sein
- .NET-Unterstützung in Namespace
  System.Web.Services.Discovery

DISCO-Beschreibungen

- Erzeugung von DISCO-Beschreibungen:
  - durch Werkzeug disco.exe

```
> disco.exe  /out:WebProject1  WebProject1/TimeService.asmx
```

  - direkt durch IIS

http://localhost/WebProject1/TimeService.asmx**?DISCO**

http://localhost/simple/TimeService.asmx?DISCO

```
<?xml version="1.0" encoding="utf-8" ?>
<discovery xmlns:xsd="http://www.w3.org/2001/XMLSchema„
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance„
    xmlns="http://schemas.xmlsoap.org/disco/">
    <contractRef ref="http://localhost/WebProject1/TimeService.asmx?wsdl„
        docRef="http://localhost/WebProject1/TimeService.asmx"
        xmlns="http://schemas.xmlsoap.org/disco/scl/" />
    <soap address="http://localhost/WebProject1/TimeService.asmx"
        xmlns:q1="http://dotnet.jku.at/time/"
        binding="q1:TimeServiceSoap"
        xmlns="http://schemas.xmlsoap.org/disco/soap/" />
</discovery>
```

Zugriff auf WSDL

Aufruf

Beispiel: TimeService Discovery (1)

- 2 Variationen von TimeService
  - TimeService1

```
<%@ WebService Language="C#" Class="TimeService1" %>
using System.Web.Services;
[WebService(Namespace="http://dotnet.jku.at/time/", Name="TimeService")]
public class TimeService1 : WebService {
      [WebMethod(Description="Returns the current server time")]
      public string GetTime() { return System.DateTime.Now.ToLongTimeString(); }
}
```

  - TimeService2

```
<%@ WebService Language="C#" Class="TimeService2" %>
using System.Web.Services;
[WebService(Namespace="http://dotnet.jku.at/time/", Name="TimeService")]
public class TimeService2 : WebService {
     [WebMethod]
     public string GetTime() { return "I don't know the time!"; }
}
```

Beispiel: TimeService Discovery (2)

- Disco-Client mit Erforschung einer DISCO-Datei

```
using System; using System.Web.Services.Discovery; using System.Collections;
public class DiscoSample {
    public static void Main(string[] args) {
```

  - Laden der DISCO-Dateien

```
        DiscoveryClientProtocol discoClient = new DiscoveryClientProtocol();
        foreach (string uri in args) {
            discoClient.Discover(uri);
        }
```

  - alle DISCO-Beschreibungen (DiscoveryDocument) durchgehen

```
        discoClient.ResolveAll();
        TimeService proxy = new TimeService();
        foreach (object obj in discoClient.Documents.Values) {
            DiscoveryDocument dDoc = obj as DiscoveryDocument;
```

Beispiel: TimeService Discovery (3)

– alle <contactRef>-Elemente durchgehen und URL auslesen

```
ContractReference contr = null;
IEnumerator it = dDoc.References.GetEnumerator();
while (contr == null && it.MoveNext())
    contr = it.Current as ContractReference;
```

– über URL zu Web-Service verbinden und Web-Methode aufrufen

```
            if (contr != null) {
                proxy.Url = contr.DocRef;
                Print("Connecting proxy to " + proxy.Url);
                proxy.Discover();
                Print("Result of GetTime: " + proxy.GetTime());
            }
        }
    }

    static void Print(string msg) { System.Console.WriteLine(msg); }
}
```

Web-Services

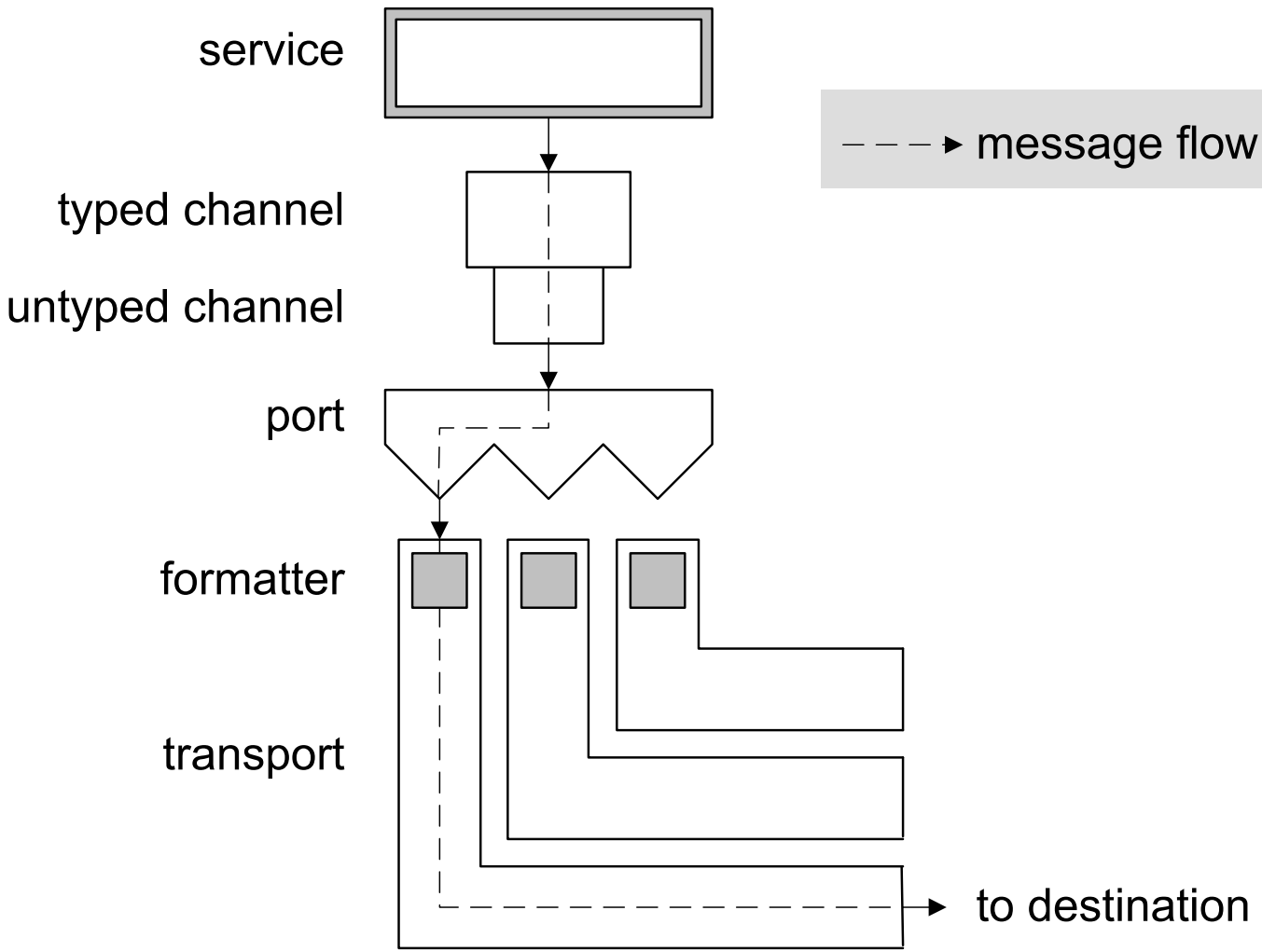
- Einführung
- Web-Services in .NET
- SOAP
- SOAP und .NET
- Dienstbeschreibung mit WSDL
- Web-Service Verzeichnisdienste: UDDI und DISCO
- Ausblick auf Web-Services .NET 2.0
- Zusammenfassung

Indigo

- Web-Services in .NET 2.0 sind integriert in *Indigo*
- Indigo vereint
  - .NET Remoting
  - Web Services
  - .NET Enterprise Services

  in einem einheitlichem Programmiermodell
- Indigo bietet
  - Transaktionen
  - Zuverlässige Kommunikation mit Messaging
  - Sichere Kommunikation und Authentifizierung
  - Unabhängigkeit von Protokollen
  - Unabhängigkeit von der Verwendung bestimmter Hosts
  - Message-basierte Server-Aktivierung

Indigo-Architektur



Indigo Web-Service Beispiel (1)

- Implementierung des Web-Service TimeService

```
using System; using System.MessageBus.Services;
[DatagramPortType(Name="TimeService", Namespace="http://dotnet.jku.at/WS")]
public class TimeService {
      [ServiceMethod]
      public DateTime GetTime() {
            DateTime now = DateTime.Now;
            Console.WriteLine ("Time request at {0}", now); // output to monitor server
            return now;
      }
}
```

- Kompilieren und Assembly erzeugen

```
csc  /target:library  /reference:System.MessageBus.dll  TimeService.cs
```

- erzeugen der WSDL-Beschreibungen

```
wsdlgen  TimeService.dll
```

Indigo Web-Service Beispiel (2)

- Implementierung einer Server-Anwendung

```
using System; using System.MessageBus.Services; using System; using System.MessageBus;
class Server {
	static void Main () {
		ServiceEnvironment se = null;
		try {
			se = ServiceEnvironment.Load();
			se.Open();
			Console.WriteLine("Press enter to stop the service ...");
			Console.ReadLine();
		} finally { if (se != null) se.Close(); }
	}
}
```

Indigo Web-Service Beispiel (3)

- Konfiguration des Servers in Datei Server.exe.config

```
<configuration>
    <system.messagebus>
        <serviceEnvironments>
            <serviceEnvironment name="main">
                <port>
                    <identityRole>
                        soap.tcp://localhost:12345/TimeService/
                    </identityRole>
                </port>
                <remove name="securityManager"/> <!-- Security disabled!!! -->
                <policyManager>
                    <areUntrustedPolicyAttachmentsAccepted> true
                    </areUntrustedPolicyAttachmentsAccepted>
                    <isPolicyReturned> true </isPolicyReturned>
                </policyManager>
                <serviceManager>
                    <activatableServices>
                        <add type="TimeService, TimeService" />
                    </activatableServices>
                </serviceManager>
            </serviceEnvironment>
        </serviceEnvironments>
    </system.messagebus>
</configuration>
```

Indigo Web-Service Beispiel (4)

- Kompilieren der Server-Anwendung

```
csc /reference:System.MessageBus.dll Server.cs
```

- Erzeugen der Proxy für Client

```
wsdlgen dotnet_jku_at.WS.wsdl dotnet_jku_at.WS.xsd
```

- Kompilieren des Proxy-Codes

```
csc /target:library /reference:System.MessageBus.dll dotnet_jku_at.WS.cs
```

Indigo Web-Service Beispiel (5)

- Implementierung der Client-Anwendung

```
using System; using System.MessageBus.Services;
public class Client {
      public static void Main () {
            ServiceEnvironment se = null;
            try {
                  se = ServiceEnvironment.Load();
                  ServiceManager sm = se[typeof(ServiceManager)] as ServiceManager;
                  if (sm == null) throw new Exception("ServiceManager is not available.");
                  se.Open();
                  Uri uri = new Uri("soap.tcp://localhost:12345/TimeService/");
                  // get a channel to the web service from the default service manager
                  ITimeServiceChannel channel = (ITimeServiceChannel)
                        sm.CreateChannel(typeof(ITimeServiceChannel), uri);
                  Console.WriteLine(channel.GetTime());// invoke web service method
            } catch (Exception e) { Console.WriteLine(e);
            } finally { if (se != null) se.Close(); }
      }
}
```

- Konfiguration des Client (analog zu Server) und Kompilieren

```
csc  /reference:System.MessageBus.dll,dotnet_jku_at.WS.dll  Client.cs
```

Indigo Web-Service Beispiel (6)

- Starten von Server und Client

```
//----- Server
> Host.exe
Press enter to stop the service ...

Time request at 1/29/2004 3:35:51 PM
>
```

```
//----- Client


> Client.exe
Time request at 1/29/2004 3:35:51 PM
>
```

Web-Services

- Einführung
- Web-Services in .NET
- SOAP
- SOAP und .NET
- Dienstbeschreibung mit WSDL
- Web-Service Verzeichnisdienste: UDDI und DISCO
- Ausblick auf Web-Services .NET 2.0
- Zusammenfassung

Zusammenfassung

- Web-Services sind eine Middleware-Technologie
- auf der Basis von XML und Internet-Protokollen
- unabhängig von Programmiersprache und Laufzeitumgebung
- für die Integration heterogener, verteilter Systeme

- .NET unterstützt Web-Service-Technologie
  - Entwicklung von Web-Services
  - Entwicklung von Web-Service-Clients
  - Ausforschung, dynamische Verbindung und Aufruf
- In .NET 2.0 vereint Indigo die unterschiedlichen Remoting-Technologien

Ressourcen (außer dotnet.jku.at)

- UDDI & Co

  www.uddi.org: Homepage der UDDI-Initiative

  www-3.ibm.com/services/uddi: Verzeichnisdienst von IBM

  uddi.microsoft.com: Verzeichnisdienst von Microsoft

  www.xmethods.com: Verzeichnis mit UDDI- und DISCO-Einträgen

- Für Entwickler

  www.w3.org: Spezifikationen

  www.webservices.org: Artikel und Tutorials über Web-Services

  www.gotdotnet.com: Seite für .NET-Technologie inkl. Web-Services

  groups.yahoo.com/group/soapbuilders: Diskussionsgruppe über SOAP

- Java-Implementierungen

  xml.apache.org/axis/: Web-Service Server in Java basierend auf Apache

  www.themindelectric.com: GLUE: Web-Service Server in Java