Web Services

Praktikum aus Softwareentwicklung 2

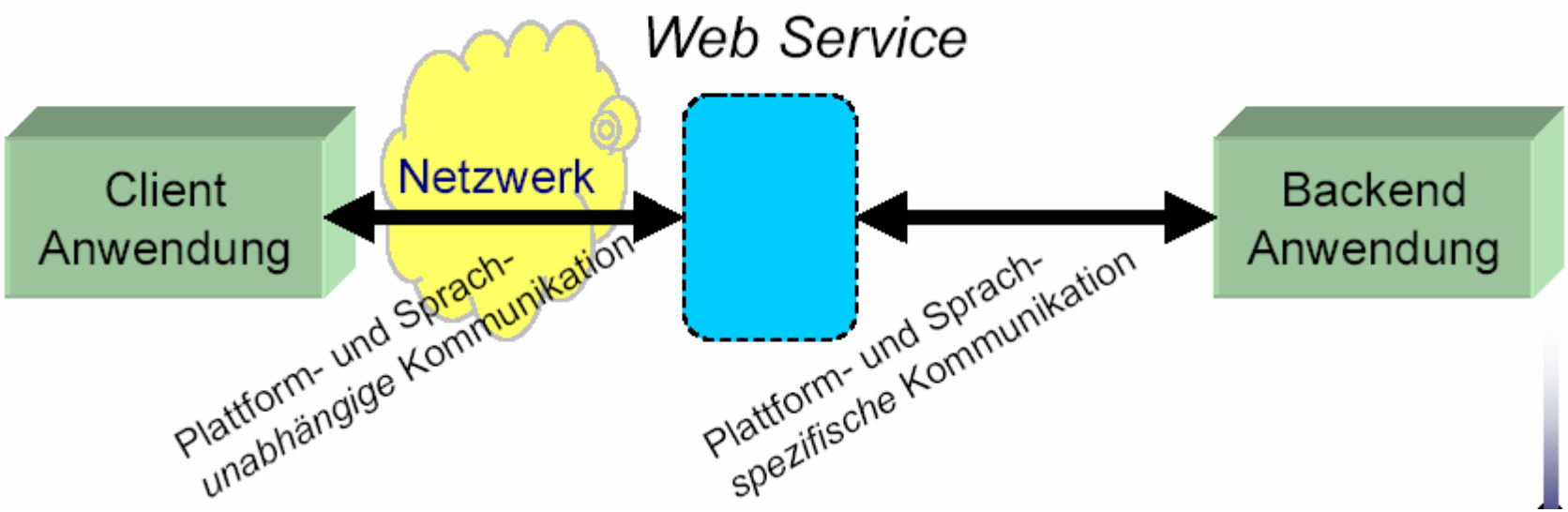
Web Services

- Einführung
  - Definition, Eigenschaften, Anwendungen....
- Architektur
  - Kommunikation, Basis Technologien....
- JAX
  - Überblick, Architektur....
- JAX-RPC
  - Übersicht, Architektur, Beispiel....

Web Services Einführung

Definition

- „...eine über das Netzwerk zugängliche Schnittstelle zu Anwendungsfunktionalität, basierend auf standardisierten Internet-Technologien„
- „Ermöglicht Clients Prozeduren auf entfernten Systemen über eine Netzwerkverbindung auszuführen.“



Web Services Einführung
Eigenschaften

- Verteilte Software Komponenten
- Basiert auf XML Standards
  - WSDL – Service Beschreibung (Web Service Description Lang.)
  - SOAP – Datenaustausch (Simple Object Access Protocol)
  - UDDI – Service Registrierung (Universal Description, Discovery and Integration)
- Internet Protokoll HTTP
- Lose Koppelung

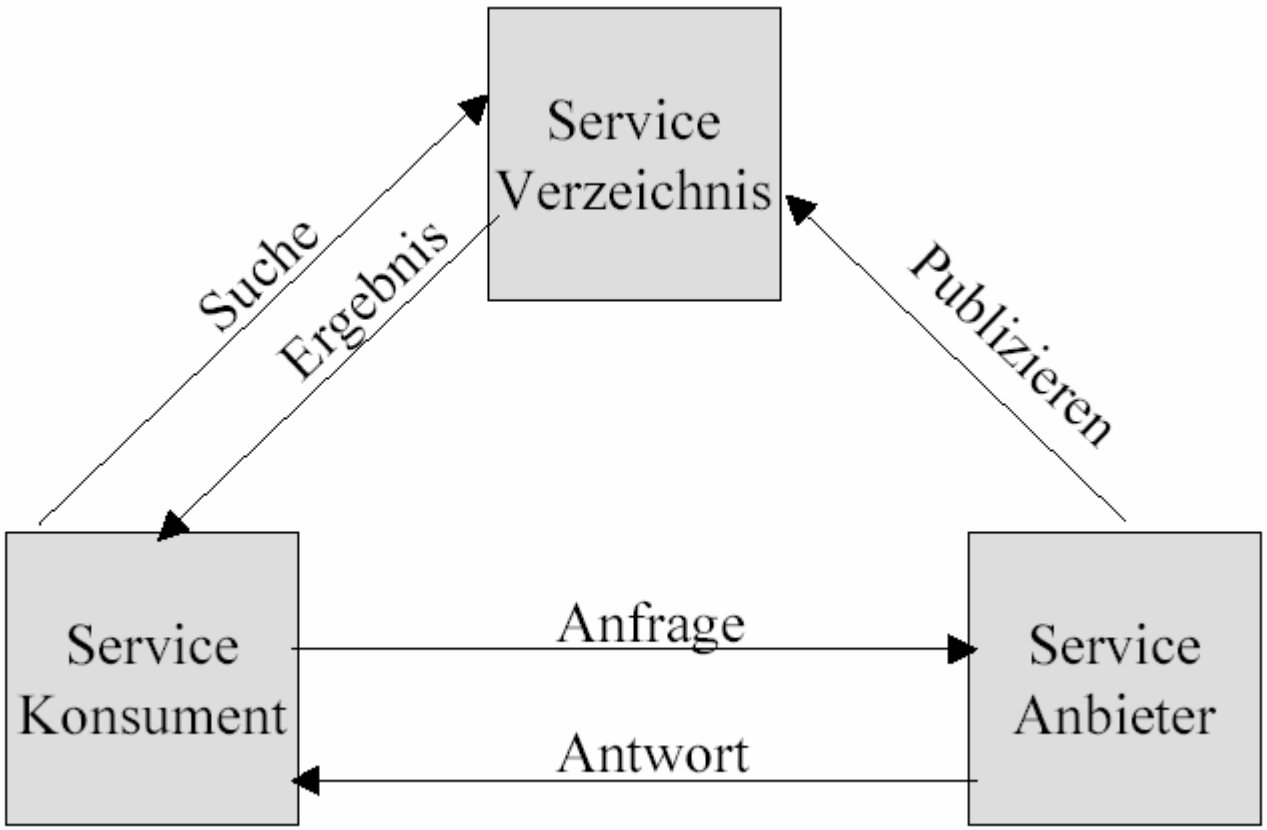
Web Services Einführung

HTML versus XML

- Web-Anwendungen
  - sind auch Web Services (HTML-basierte)
  - sind auf menschliche Benutzer (bzw. Web-Browser) ausgerichtet
  - können nur umständlich und fehleranfällig in andere Programme integriert werden
- XML-basierte Web Services (XWS)
  - sind genauso leicht zugänglich wie Web-Anwendungen
  - sind auf die Verwendung durch Programme ausgerichtet (strukturierte Daten statt Präsentation)
  - vereinfachen die Integration von Informationen und von Workflows

Web Services Einführung

Service Orientierte Architektur (SOA)



Web Services Einführung
Anwender Perspektiven

- Geschäftsanwender / Organisationen
  - Einfache Zusammenarbeit mit Partnern (Transaktionskosten)
  - *Business Web*
- Privatanwender
  - Automatische Preisvergleiche (*Shopping Agents*)
  - Integrierte Auskunftsdienste (berücksichtigen Standort, Fahrpläne, Wetter, Verkehrsstaus,...), etc.
  - Intelligente, personalisierte Dienste (*Persönliche Agenten*)
- IT-Manager
  - Enterprise Application Integration (EAI)
  - Business to Business Integration (B2Bi)
- Entwickler
  - Web Services veröffentlichen (Schnittstellen zu existierenden Systemen entwickeln), externe Web Services einbinden

Web Services Einführung

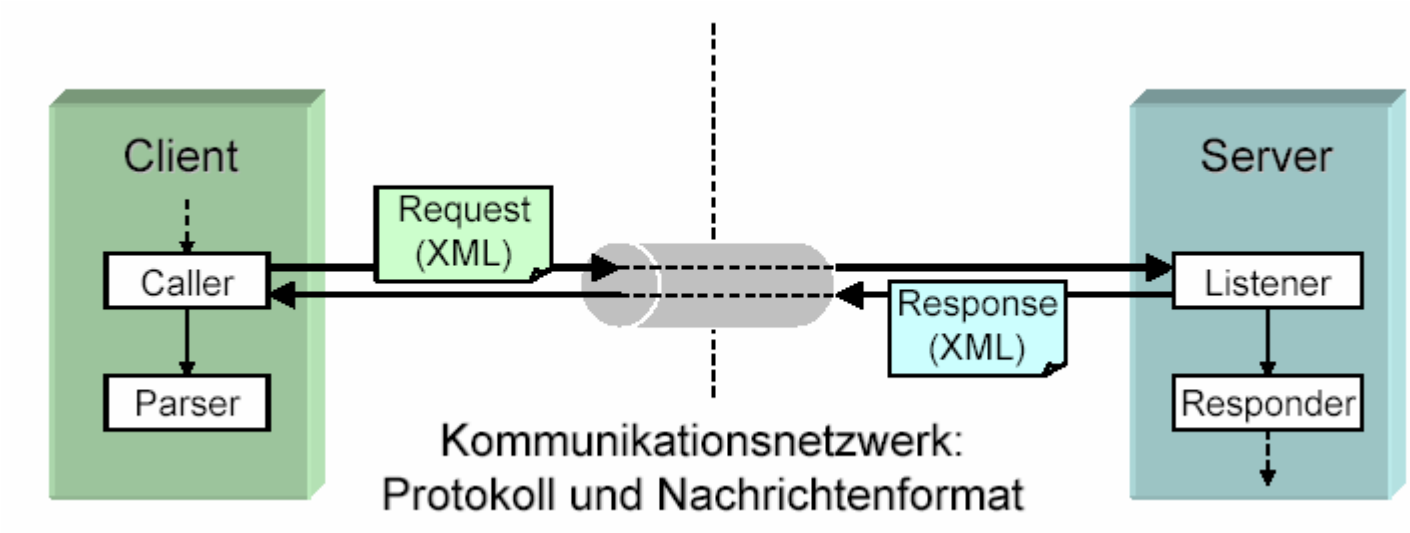
Links

- IBM
  - http://www-106.ibm.com/developerworks/webservices
- Microsoft
  - http://msdn.microsoft.com/webservices
- Sun Microsystems
  - http://java.sun.com/webservices
- Apache
  - http://ws.apache.org/wsif
  - http://ws.apache.org/axis/java/index.html
- W3C
  - http://www.w3.org/2002/ws

Web Services

- Einführung
  - Definition, Eigenschaften, Anwendungen....
- Architektur
  - Kommunikation, Basis Technologien....
- JAX
  - Überblick, Architektur....
- JAX-RPC
  - Übersicht, Architektur, Beispiel....

Architektur
Kommunikation



Lose Kopplung:

- zustandsloses Kommunikationsprotokoll (HTTP)
- "neutrales" Nachrichtenformat

Architektur

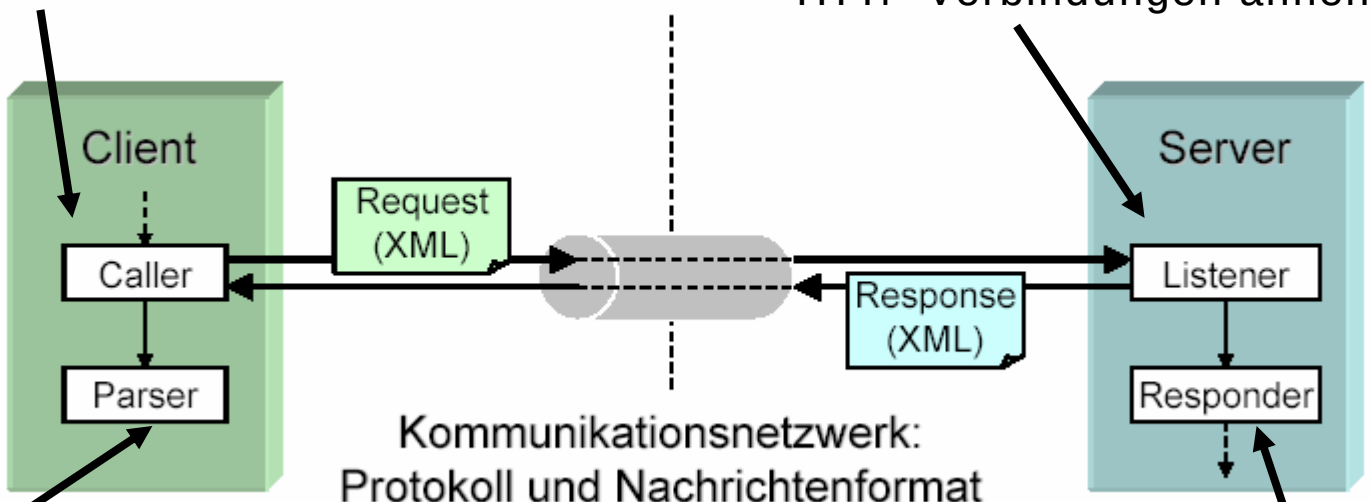
Basistechnologien

Java, VB, .NET

Eine Komponente, die eine HTTP--Verbindung herstellen kann

ASP, JSP, Servlet, .NET

Eine Komponente, die HTTP-Verbindungen annehmen kann



DOM, SAX

Http,
TCP/IP, UDP

XML,
Soap, WSDL

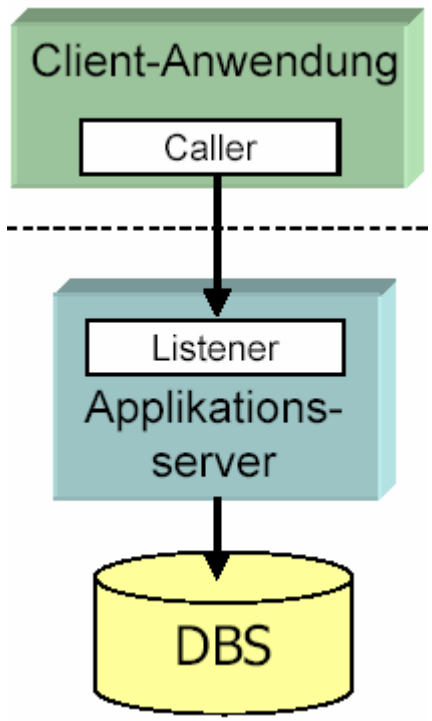
Eine Komponente, die Anwendungsfunktionalität bereit stellt

Architektur
Client to Service

Benutzerschnittstelle und *Prozesslogik*

Prozesslogik

Datenhaltung



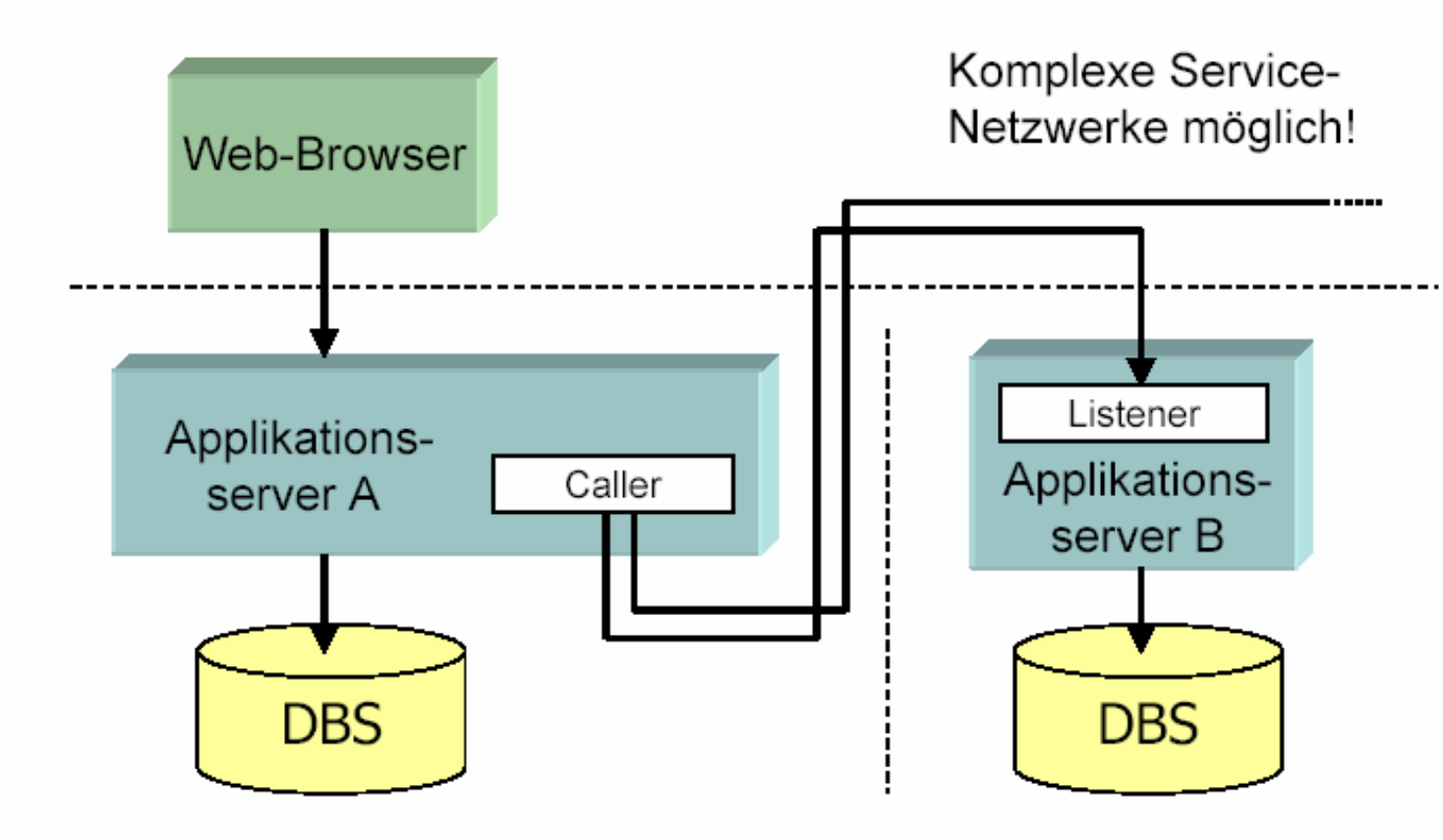
Web Service Caller
Java, C++, C#, VB, PHP

Webservice *Listener,*
Anwendungslogik,
Datenbankzugriff

Daten, Schema, Stored Procedures

Architektur

Service to Service



tiscover
THE TRAVEL NETWORK

Architektur

Technologie Stack

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

UDDI – Universal Description, Discovery and Integration

WSDL – Webservice Description Language

SOAP – Simple Object Access Protocoll

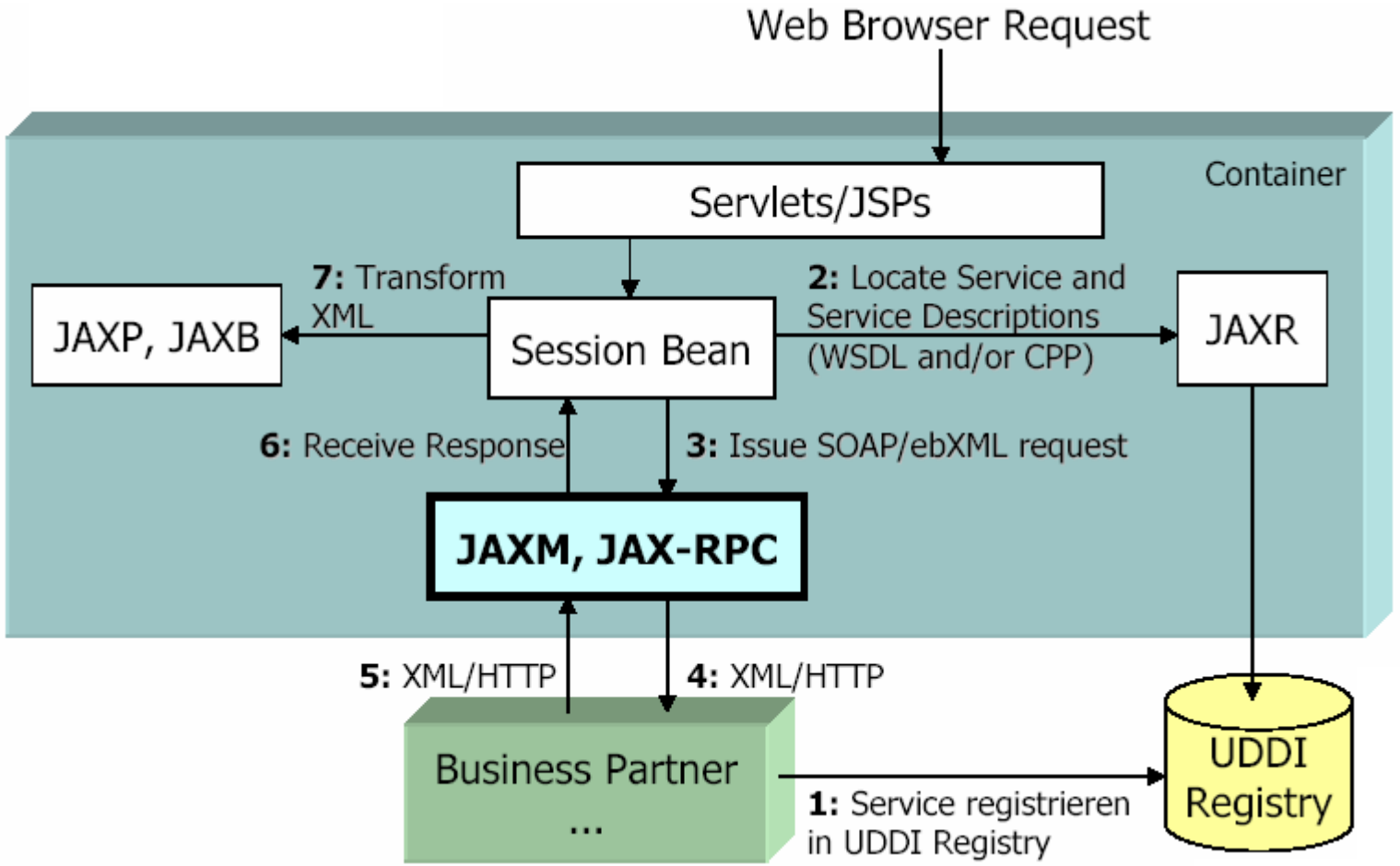
Web Services

- Einführung
  - Definition, Eigenschaften, Anwendungen....
- Architektur
  - Kommunikation, Basis Technologien....
- JAX
  - Überblick, Architektur....
- JAX-RPC
  - Übersicht, Architektur, Beispiel....

JAX
Überblick

- JAX – Java API‘s for XML
- Einteilung in 2 Bereiche: API‘s die direkt mit Dokumenten arbeiten und API‘s die mit Prozeduren arbeiten
  - Dokumenten-orientiert
    - JAXP – Java API für XML Processing
    - JAXB – Java Architecture for XML Binding
    - SAAJ – SOAP with Attachments API for Java (SAAJ ist eine Implementation von JAXM – Java API for XML Messaging
  - Prozedur-orientiert
    - JAX-RPC – Java API for XML-based RPC
    - JAXR – Java API for XML Registries

JAX
Architektur



JAX
JAXP

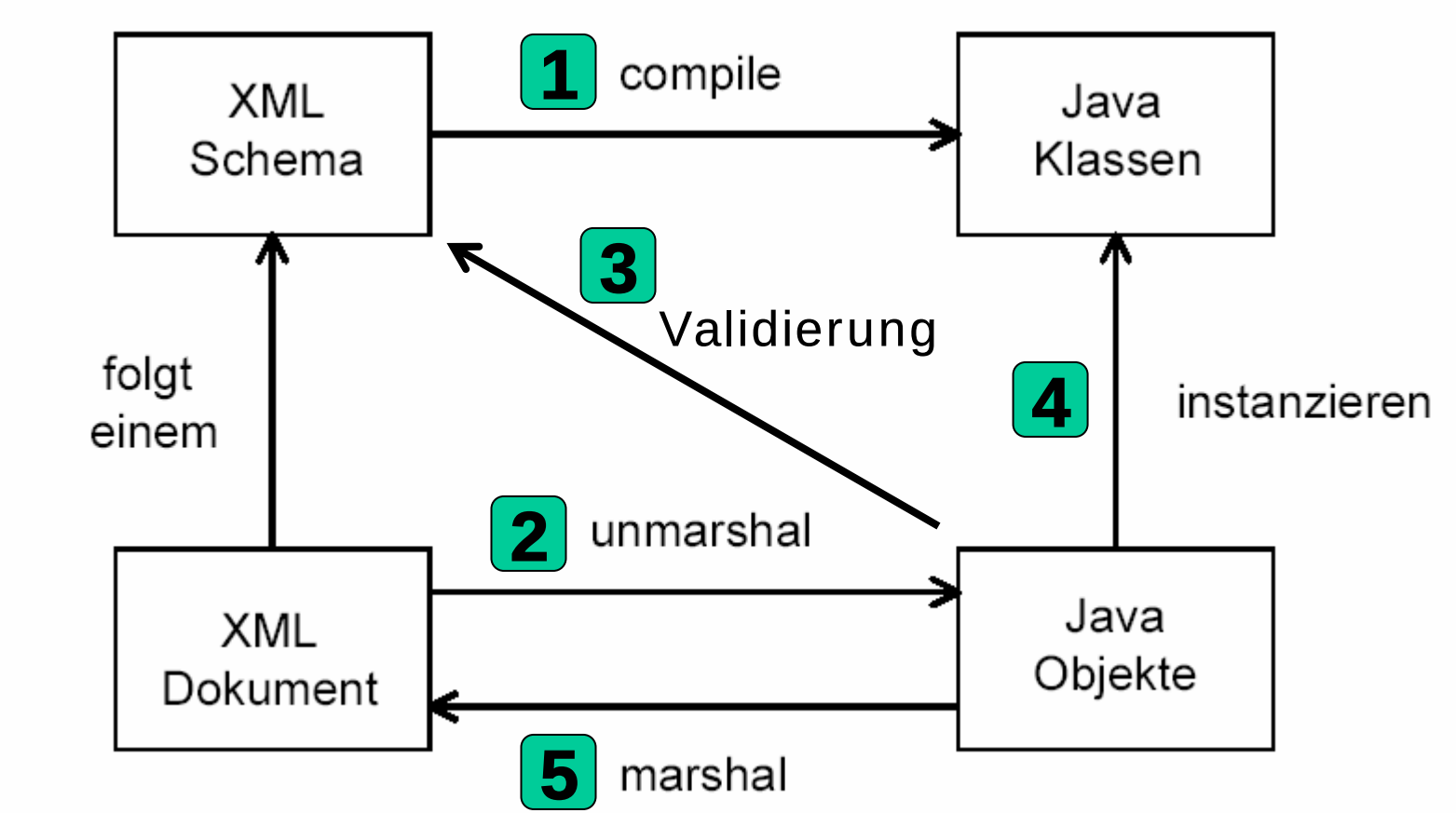
- JAXP – Java API für XML Processing
  - SAX API– Simple API for XML
  - DOM API – Document Object Model
  - XSLT API – XML Stylesheet Language for Transformations
    - XML Stylesheet Language (XSL) bestimmt wie die XML Daten angezeigt werden
    - XSLT benutzt die Formatangaben von XSL für die Transformation

JAX
JAXB - Überblick

- Java Architecture for XML Binding
  - Ein Konzept um XML Dokumente im Speicher zu repräsentieren
  - Automatisches Erzeugen von Java Klassen aus vorgegebenem XML Schema
  - Spezielle Klassen für XML-Elemente / Attribute
  - Methoden zum Zugriff auf Attribute und Baumstruktur (getypt)
  - Java Objekte instanzieren diese Klassen
  - Abbildungsprozess erfolgt über Marshalling Framework
    - „marshal“ Java Objekte →XML Elemente
    - „unmarshal“ XML Elemente → Java Objekte

JAX

JAXB – Bind Process



JAX
JAXB – compile

```
<?xml version="1.0"?>
<XmlEmployees>
  <XmlEmployee>
    <EmpNo>2</EmpNo>
    <FirstName>Robert</FirstName>
    <LastName>Nelson</LastName>
    <PhoneExt>250</PhoneExt>
    <HireDate>1988-12-28</HireDate>
    <DeptNo>600</DeptNo>
    <JobCode>VP</JobCode>
    <JobGrade>2</JobGrade>
    <JobCountry>USA</JobCountry>
    <Salary>105900.000000</Salary>
    <FullName>Nelson,
              Robert</FullName>
  </XmlEmployee>
</XmlEmployees>
```

```
public interface XmlEmployeeType {
  java.lang.String getJobCode();
  void setJobCode(java.lang.String value);
  java.lang.String getDeptNo();
  void setDeptNo(java.lang.String value);
  java.lang.String getPhoneExt();
  void setPhoneExt(java.lang.String
value);
....
}
```

JAX
JAXB – Komplettes Beispiel

- Auf der Website des Praktikums befindet sich ein komplettes Beispiel
  - `Employees.xml` → Beispiel XML File (Daten)
  - `Employees.xsd` → Beispiel Schema (Struktur)
  - `Jaxb.java` → Demo Programm
  - Batch Dateien in dieser Reihenfolge ausführen
    - `jaxb-generate.bat` → Generiert die Klassen in das package: com.exercise.eight
    - `jaxb-compile.bat` → Kompiliert alle Klassen auch Jaxb.java
    - `jaxb-run.bat` → Startet das Beispiel Programm
    - Es wird ein JDK >= 1.4.1 benötigt!

JAX

SAAJ

- SOAP with Attachments API for Java ist ein Standard für die Versendung von XML Dokumenten über das Internet
- Basiert auf SOAP 1.1 und der SOAP with Attachments Spezifikation
- Low-level SOAP Schnittstellen
- JAX-RPC basiert auf SAAJ
- Synchroner Request/Response Mechanismus

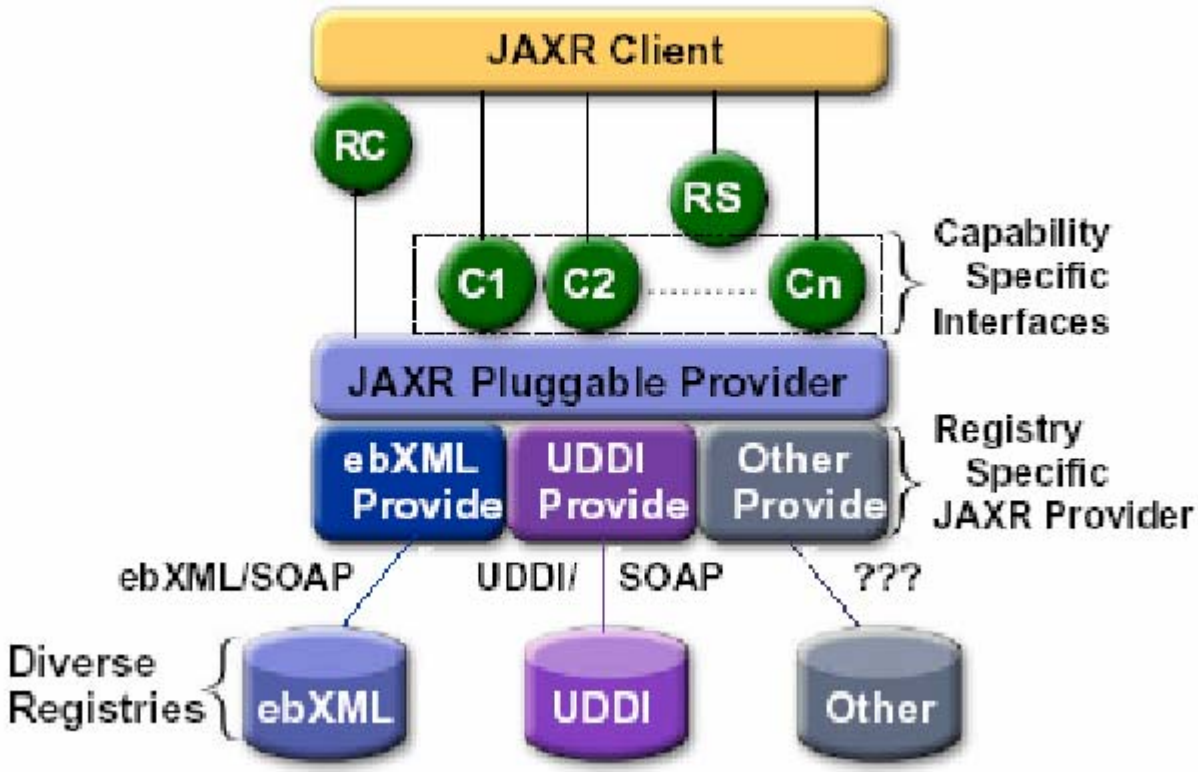
JAX

JAXR - Registries

- Java API for XML Registries
- Registries enthalten wichtige Informationen zur Kommunikation und Kollaboration von Unternehmen
- **Generische** API zum Abfragen/Bearbeiten von Registries, die JAXR Provider bereitstellen
- Unterstützt **Vereinigungsmenge** der Konzepte komplementärer Ansätze (z.B. UDDI, ebXML)
- Package: `javax.xml.registry`

JAX

JAXR - Architketur



JAX

SOAP – (1/3)

- Simple Object Access Protocol
- Lightweight Message Protokoll
- Getypter Datenaustausch zwischen Applikationen
- Nachrichtenstruktur wird in XML definiert
- Besonders geeignet für RPC Funktionalität
- Lose Kopplung
- Transport Binding über darunterliegendes Transport Protokoll - HTTP

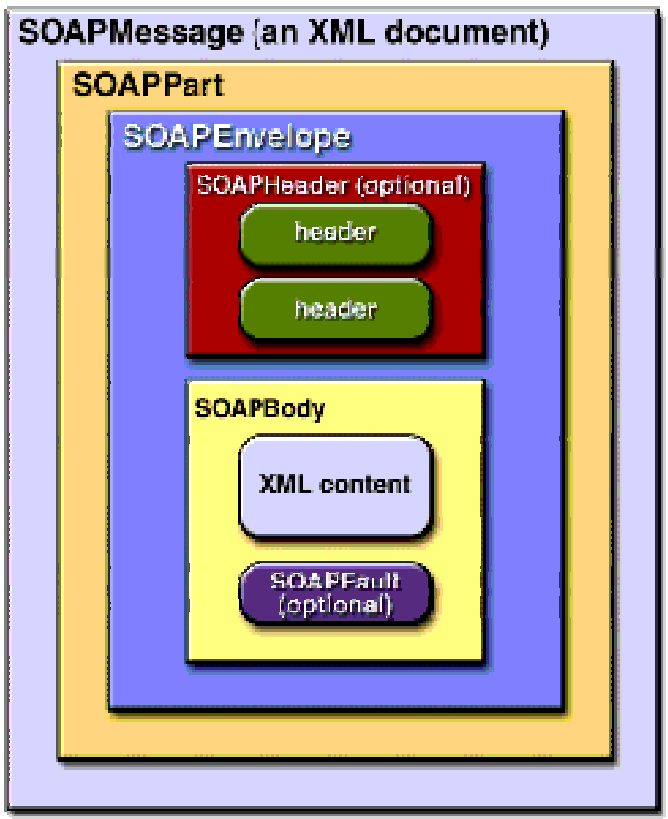
JAX

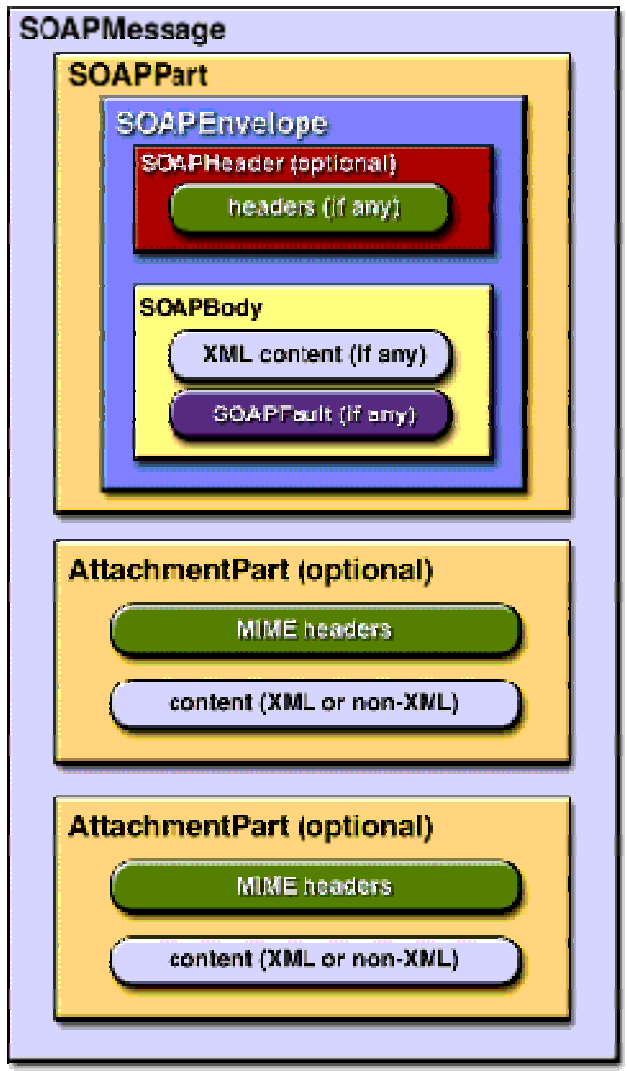
SOAP – (2/3)

- Spezifikation SOAP v. 1.1
- W3C Spezifikation: http://www.w3.org
- Umfasst:
  - SOAP envelope – Struktur eine SOAP Nachricht
  - SOAP encoding – deserialisierungs Regeln
  - SOAP binding Framework – Binding an ein bestimmtes Transport Protokoll
  - SOAP RPC

JAX

SOAP – (3/3)


SOAPMessage (an XML document)
SOAPPart
SOAPEnvelope
SOAPHeader (optional)
header
header
SOAPBody
XML content
SOAPFault (optional)


SOAPMessage
SOAPPart
SOAPEnvelope
SOAPHeader (optional)
headers (if any)
SOAPBody
XML content (if any)
SOAPFault (if any)
AttachmentPart (optional)
MIME headers
content (XML or non-XML)
AttachmentPart (optional)
MIME headers
content (XML or non-XML)

Web Services

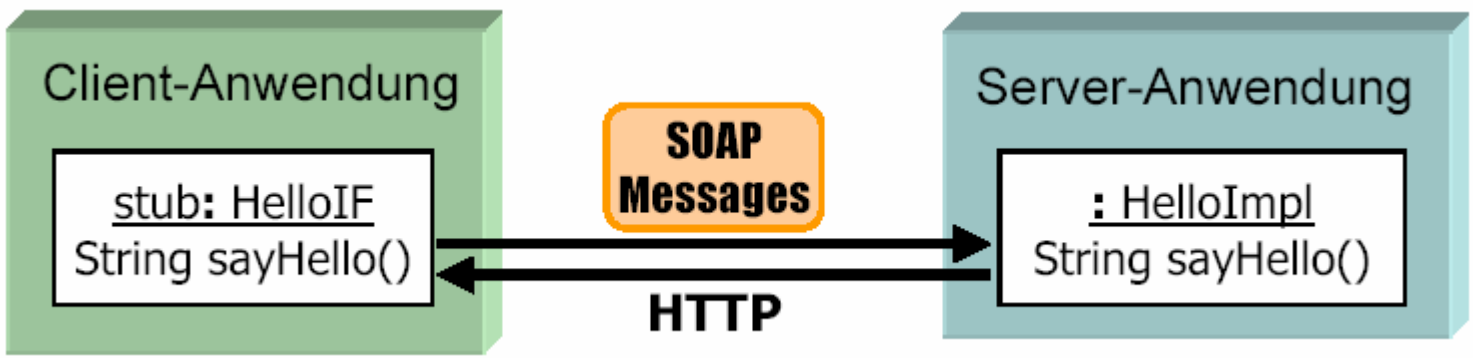
- Einführung
  - Definition, Eigenschaften, Anwendungen....
- Architektur
  - Kommunikation, Basis Technologien....
- JAX
  - Überblick, Architektur....
- JAX-RPC
  - Übersicht, Architektur, Beispiel....

JAX-RPC
Übersicht

- „Java API for XML based RPC“
- RPC = Remote Procedure Call
- Programmierschnittstelle
  - Für Entwicklung von Web Services
  - Für Entwicklung von Web Services Clients
- Referenz Implementierung und Developer Pack von Sun Microsystems: JWSDP 2.0
  - http://java.sun.com/webservices/downloads/webservicespack.html

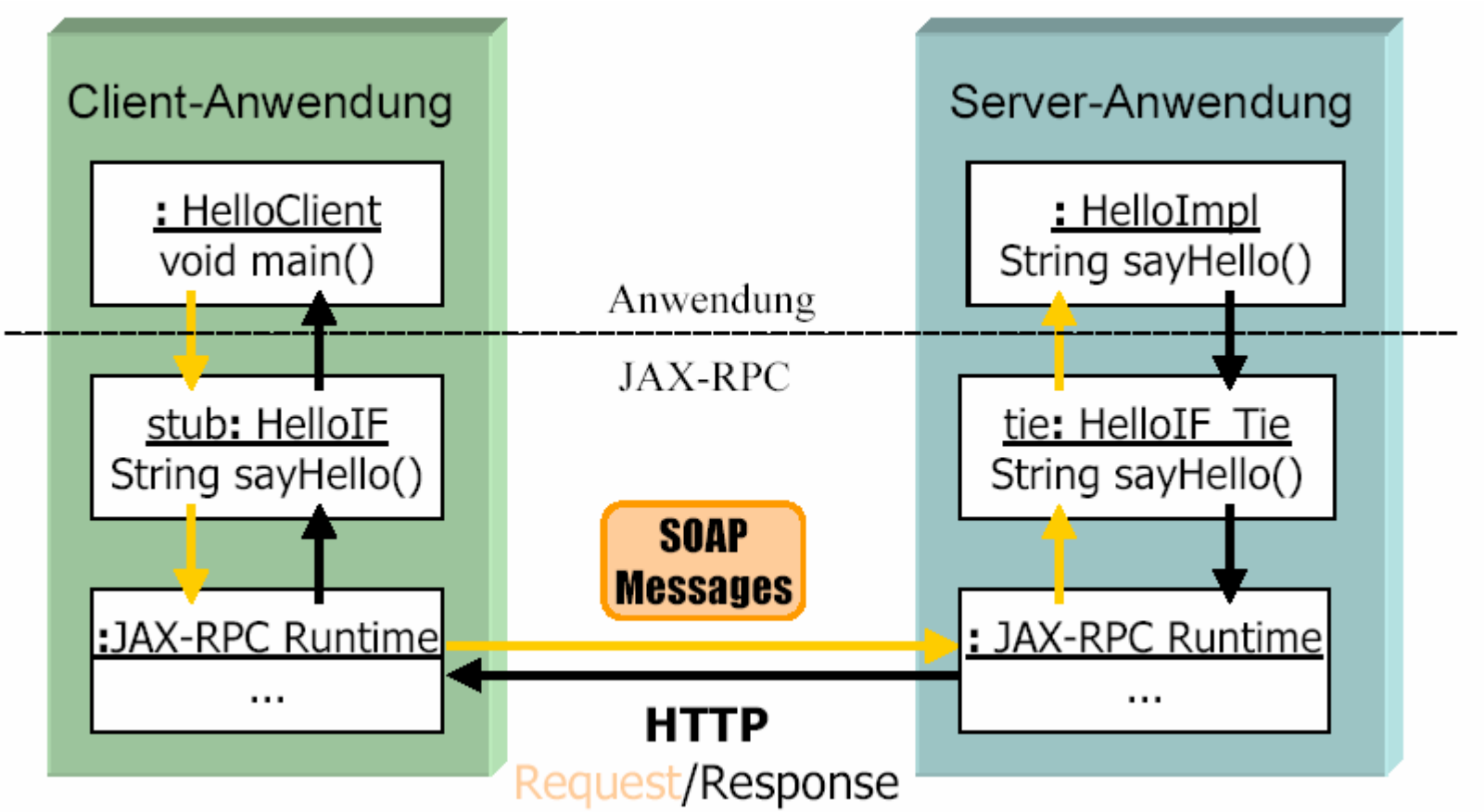
JAX-RPC
Client/Server Schnittstelle

- Protokoll basiert auf SOAP
- API “versteckt” hohe Komplexität
- Ein Service besteht nur aus 2 Klassen: Ein Interface welches die Remote Prozeduren des Services beschreibt und eine Klasse die diese implementiert
- Client verwendet *Stub* Objekt zum Aufrufen von Methoden des Service Objekts *Tie*



JAX-RPC

Client/Server Schichten



Web Services Developer Pack 2.0 (1/2)

- WS Dev Packet 2.0 und Tomcat 5.0 für WSDP 1.5 downloaden:
- http://java.sun.com/webservices/downloads/webservicespack.html
- http://java.sun.com/webservices/containers/
- Tomcat entpacken nach: D:\tomcat50-jwsdp → D:\wsdp-2.0
- Start Tomcat WSDP → http://localhost:8080
- Systemproperty bei JDK1.5: java.endorsed.dirs=D:\wsdp-2.0\jaxp\lib;D:\wsdp-2.0\jaxp\lib\endorsed

Web Services Developer Pack 2.0 (2/2)

- Inhalt:
  - JAX-WS Version 2.0 EA (Java API for XML Web Services)
  - JAX-WSA Version 1.0 EA (Java API for XML Web Services Addressing)
  - Fast Infoset Version 1.0.1
  - Sun Java Streaming XML Parser Version 1.0 EA
  - XML Digital Signature Version 1.0.1
  - XML and Web Services Security Version 2.0 EA2
  - JAXB Version 2.0 EA (Java Architecture for XML Binding)
  - JAXP Version 1.3.1_01 (Java API for XML Processing)
  - JAXR Version 1.0.8_01 EA (Java API for XML Registries)
  - JAX-RPC Version 1.1.3_01 EA (Java API for XML-based RPC)
  - SAAJ Version 1.3 EA (SOAP with Attachments API for Java)

JAX-RPC
Entwicklungschritte

- Entwicklung des „Hello“ Webservices mit dem Jwsdp 2.0
  1. **Coding**: Codierung des „service endpoint interface“ und der Implementationsklasse
  2. **Build**: Compilierung, Bau, Generierung und Verpackung der Dateien welche für den Service benötigt werden
  3. **Deploy**: Einspielen des WAR Files mit dem Service

JAX-RPC
Coding (1/4)

- Ein „service endpoint interface“ muss diese Regeln befolgen:
  - Es ist vom „java.rmi.Remote interface“ abgeleitet
  - Es darf keine Konstanten deklarieren, wie public final static
  - Die Methoden müssen die „ java.rmi.RemoteException“ oder eine ihrer abgeleiteten Klassen werfen
  - Methoden Parameter und Rückgabewerte müssen unterstütze Typen sein. Es muss ein Mapping des Java-Types auf einen XML/WSDL Typen möglich sein

JAX-RPC
Coding (2/4)

J2SE SDK Klassen	Primitives	Java Collections	Complexe Typen
java.lang.Boolean java.lang.Byte java.lang.Double java.lang.Float java.lang.Integer java.lang.Long java.lang.Short java.lang.String java.math.BigDecimal java.math.BigInteger java.net.URI java.util.Calendar java.util.Date	boolean byte double float int long short	**List** ArrayList LinkedList Stack Vector **Map** HashMap Hashtable Properties TreeMap **Set** HashSet TreeSet	Arrays JavaBeans Value Types

JAX-RPC
Coding (3/4)

- Value Types (Value Objects)
  - Public default Constructor
  - Es darf nicht das `java.rmi.Remote` Interface implementieren, weder direkt noch indirekt
  - Die Felder müssen JAX-RPC konforme Typen sein
  - Felder können private, public oder protected sein
  - Ein public Feld darf nicht final oder transient sein
  - Nicht public Felder müssen Setter und Getter Methoden haben

JAX-RPC
Coding (4/4)

- Hello Service Schnittstelle definieren *(service endpoint interface)*

```
package test;
import java.rmi.Remote;
import java.rmi.RemoteException;

interface HelloIF extends Remote {
  public String sayHello(String s)throws RemoteException;
}
```

- Hello Service Schnittstelle **implementieren**

```
class HelloImpl implements HelloIF {
  public String sayHello(String s)throws RemoteException {
    return "You said <" + s + "> and I say good bye. ";
  }
}
```

JAX-RPC
Build (1/4)

- Verzeichnisstruktur:
- 2 Projekte ws-server und ws-client
  - ws-server/src/ → Java Source Files für den Server
  - ws-server/classes → generierte Klassen
  - web

  - src/client → Java Source Files für den Client

JAX-RPC

Build (2/4)

- Generierung und Deployment
  1. Compilieren der beiden Ausgangsklassen → `hello_compile.bat`
  2. Generieren des WSDL-Files und definieren des Services → `hello_wscompile.bat`
     Es wird ein „Model“ erzeugt: `server/model.gz` und das WSDL-File `server /HelloService.wsdl`
     Das minimale Konfigurationsfile für den Service:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration   xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <service name="HelloService„
            targetNamespace="urn:Foo"
            typeNamespace="urn:Foo„
            packageName="test">
    <interface name="test.HelloIF"/>
  </service>
</configuration>
```

  3. Verpackung in ein portables WAR-File. Dieses WAR-File ist noch nicht bereit für das Deployment, da noch die Tie Klassen fehlen, aber diese sind Plattform und Tool abhängig.

JAX-RPC

Build (3/4)

3. Das portables WAR-File besteht aus folgenden Dateien:

   ```
   WEB-INF/classes/test/HelloIF.class
   WEB-INF/classes/test/HelloImpl.class
   WEB-INF/jaxrpc-ri.xml
   WEB-INF/model.gz
   WEB-INF/web.xml
   ```

   Die Datei `jaxrpc-ri.xml` ist Implementationsspezifisch!

   Mit der Batch Datei `hello_war-portable.bat` werden die benötigten Dateien in das web Vz. kopiert und die Datei: `server/hello-portable.war` wird erstellt.

4. Nun kann die WAR-Datei für das Deployment mit den `Tie` Klassen erstellt werden. Diese Datei wird mit dem `wsdeploy`-Tool erstellt. Dabei werden die `Tie` Klassen erstellt, ein WSDL File und alle Dateien werden in ein WAR File gepackt, welches bereit ist für den Deployment Vorgang.

JAX-RPC

Build (4/4)

4. Mit der Batch-Datei `hello_wsdeploy.bat` wird die WAR-Datei `hello-jaxrpc.war` erstellt.

   Dieses `hello-jaxrpc.war` ist unsere fertige Webapplikation. Es sind weiter Klassen dazugekommen und auch das `web.xml` wurde geändert.

   Auch das ursprüngliche `jaxrpc-ri.xml` wurde geändert und die original Dateien sind ebenfalls mit den Namen `*-before.xml` im WAR File.

JAX-RPC

Deployment

- Deployment Jwsdp 2.0
  - Im Jwsdp 2.0 ist kein Tomcat Webcontainer integriert, aber es gibt speziellen einen speziellen 5.0 Tomcat
  - Start und Stop-Scripts befinden sich im `tomcat50-jwsdp\bin` Vz.
  - Das Webservice WAR File `hello-jaxrpc.war` wird einfach in das Vz. `\tomcat50-jwsdp\webapps` kopiert oder über `http://localhost:8080/manager/html`
  - Testaufruf mit: `http://localhost:8080/hello-jaxrpc/`

JAX-RPC

Server

- Web Services Engine verwaltet Services
  - Kommunikation basiert auf HTTP
  - Implementierung basiert auf Servlets
  - Realisiert SOAP Protokoll
- Notwendige Komponenten
  - HTTP Server
  - Servlet Engine
  - Services Engine

JAX-RPC
Arten von Clients

- 3 Arten von Webservice Clients
  - Static stub → die Stub-Klasse wird vor der Laufzeit erzeugt. Die Stub Klasse wird als **statischer Proxy** verwendet
  - Dynamic proxy → die Stub-Klasse wird zur Laufzeit erzeugt und kann auch als **dynamischer Proxy** bezeichnet werden
  - Dynamic invocation interface (DII) → mit diesen Interface können Remote Procedures auch dann aufgerufen werden, wenn die Signatur oder der Service bis zur Laufzeit nicht bekannt sind. Diese Clients sind kompliziert zu entwickeln und man benötigt genau Kenntnisse von WSDL - Dokumenten

JAX-RPC

Static Stub Client (1/5)

- *HelloClientStatic* ist ein *Standalone* Programm, welches die sayHello Remote Procedure unseres HelloService aufruft
- 3 Schritte sind notwendig:
  - Coding: Implementierung des Clients
  - Build: Generierung, Bauen und Verpacken des Clients
  - Running: Aufrufen des Clients

JAX-RPC

Static Stub Client (2/5)

1. Erzeugen eines Stub Objektes:

   ```
   (Stub)(new HelloService_Impl().getHelloIFPort())
   ```

   Dieser Code ist abhängig von der Implementierung! Das Objekt `HelloService_Impl` wird erst im Schritt 2 (Build) generiert. Der Name der Klasse entspricht dem Namen des Webservice plus „_Impl“
2. Setzen der Endpunkt-Adresse um auf den Service zuzugreifen:

   ```
   stub._setProperty
   (javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY, args[0]);
   ```

   Diesen Endpunkt übergeben wird als Parameter beim Starten
3. Casten des Stub Objektes auf das Service Endpunkt Interface, `HelloIF`:

   ```
   HelloIF hello = (HelloIF)stub;
   ```

JAX-RPC

Static Stub Client (3/5)

```
import staticstub.*;
import javax.xml.rpc.Stub;
public class HelloClientStatic {
public static void main(String[] args) {
  try {
    Stub stub = createProxy();
    stub._setProperty(javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY, args[0]);
    HelloIF hello = (HelloIF)stub;
    System.out.println(hello.sayHello("hi Mr Service!"));
  } catch (Exception ex) { ex.printStackTrace(); }
  }

  private static Stub createProxy() {
    // Note: MyHelloService_Impl is implementation-specific.
    return (Stub)(new HelloService_Impl().getHelloIFPort());
  }
}
```

JAX-RPC

Static Stub Client (4/5)

- Generierung: `hello_client_wscompile.bat` → `wscompile`
  - Ruft das WSDL File über: http://localhost:8080/hello-jaxrpc/test?WSDL auf und generiert aus diesem die Stub Klasse und weitere Laufzeit Klassen (Serializer, Value Types..)
  - Die Konfigurationsdatei für `wscompile` (`config-wsdl.xml`):

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
   xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <wsdl location="http://localhost:8080/hello-jaxrpc/test?WSDL"
                          packageName="staticstub"/>
</configuration>
```

- Kompilieren und Verpacken mit den Scripts: `hello_client_compile.bat` und `hello_client_jar.bat`

JAX-RPC

Static Stub Client (5/5)

- Aufrufen des Clients: `hello_client_run.bat`
  - Setzt einen Request auf: http://localhost:8080/hello-jaxrpc/test
- Ausgabe:
  - `You said <hi Mr Service!> and I say good bye.`
- Logfiles am Server:
  - `tomcat50-jwsdp/logs/access*.log`
  - `tomcat50-jwsdp/logs/launcher.server.log`

Ende der 8. Übung