Verteilte Systeme - Zusammenfassung

Jan Fässler & Chregi Glatthard

4. Semester (FS 2013)

Inhaltsverzeichnis

1 Networking

1.1 InetAddress

Static factory methods

- getByName(String name)
- getByAddress (4/16 bytes)
- getAllByName(String host)
- getLocalHost()

Instance methods

- byte[] getAddress()
- String getHostAddress()
- String getHostName()
- String getCanonicalHostName()
- boolean isReachable(int timeout)
- boolean isMulticastAddress()

1.2 Network Interfaces

Listing 1: Network Interfaces and its addresses

```
public static void main(String[] args) throws SocketException {
  Enumeration<NetworkInterface> interfaces = NetworkInterface.getNetworkInterfaces();
  while(interfaces.hasMoreElements()){
    NetworkInterface intf = interfaces.nextElement();
    Syst  em.out.print(intf.getName());
    System.out.println(" ["+intf.getDisplayName()+"]");
    Enumeration<InetAddress> adr = intf.getInetAddresses();
    while(adr.hasMoreElements()){
      System.out.println("\t" + adr.nextElement());
    }
    byte[] hardwareAddress = intf.getHardwareAddress();
  }
}
```

1.3 Sockets

Abstraction through which an application may send and receive data through the network. A Socket is identified by Hostname/IP and port number.

Stream Sockets

- Use TCP as end-to-end protocol
- Provide a reliable byte-stream
- Connection oriented: Socket represents one end of a TCP connection

Datagaram Sockets

- Use UDP as protocol
- Not connection oriented, not reliable

1.3.1 Controlling Socket Behaviors

Blocking & Timeouts

ServerSocket.accept / InputStream.read
read or accept call will not block for more than a fixed number of msec otherwise, InterruptedIOException is thrown (get/setSoTimeout(int timeout))

Socket constructor
Uses a system-defined timeout, cannot be changed by Java API (Solution: use connect)

OutputStream.write
Cannot be interrupted / caused to time-out by Java API

Keep-Alive

- TCP provides a keep-alive mechanism
- Probe messages are sent after a certain time
- Application only sees keep-alive working if the probes fail!
- Per default keep-alive is disabled
- Default timeout: 2h (7200 secs)

Send / Receive Buffer Size

- When a Socket is created, the OS must allocate buffers to hold incoming & outgoing data
- Receive buffer size may also be specified on server socket (for accepted sockets which immediately receive data)

No Delay

- TCP tries to avoid sending small packets
- Buffers data until it has more to send, combines small packets with larger ones
- Necessary if application has to be efficient
- Default: false

1.3.2 Closing Connections

close()

- Once an endpoint (client or server) closes the socket, it can no longer send or receive data
- Close can only be used to signal the other end that the caller is completely finished communicating

shutdownOutput()

- Closes output-stream, no more data can be may be written (IOException)
- All data written before shutdownOutput can be read by receiver

shutdownInput()

- Closes the input stream
- Any undelivered data is (silently) discarded, read operations will return -1

s.close() / s.shutdownOutput()

- Data may still be waiting to be delivered to the other side
- By default, socket tries to deliver remaining data, but if socket crashes, data may be lost without notification to sender (as close returns immediately)

1.3.3 User Datagram Protocol

- UDP allows to address applications over ports
- UDP adds another layer of addressing (ports) to that of IP
- UDP detects some form of data corruption that may occur in transit and discards corrupted messages
- UDP retains message boundaries

2 Internet

2.1 Protocol

GET

- Access of content from the server
- Idempotent, i.e. the side effects of N¿0 identical requests is the same as for a single request (f(f(x)) = f(x))

POST
Comparable to GET but Method must not necessarily be idempotent and Request data is transferred in the body of the request

HEAD

- Identical to GET, except that the server must not return the body
- Can be used to request meta information (headers) about the resource

OPTIONS (1.1)
Returns information about the communication options available on the specified resource (or on the server in general if request URI=*)

PUT (1.1)
Stores a web page on the server (rarely implemented)

DELETE (1.1)
Removes a web resource from the servver (rarely implemented)

TRACE (1.1)
Returns the request as it was accepted by server (⇒ debugging)

CONNECT (1.1)
Implemented by Proxy Server capable to provide an SSL tunnel

2.1.1 Response Codes

200-299: Success

200 OK

201 Created

202 Accepted

300-399: Redirections

300 Multiple Choices

301 Moved Permanently

302 Found

303 See Other (e.g. after POST)

304 Not Modified

305 Use Proxy

307 Temporary Redirect

400-499: Client Error

400 Bad Request

401 Unauthorized

402 Payment Required

403 Forbidden

404 Not Found

405 Method Not Allowed

407 Proxy Authentication Required

408 Request Time-out

411 Length Required

413 Request Entity Too Large

414 Request-URI Too Large

415 Unsupported Media Type

500-599: Server Error

500 Internal Server Error

501 Not Implemented

503 Service Unavailable

505 HTTP Version not supported

2.2 Request Headers

Host server host

Referer host from which the request is initiated

Accept data types supported by the client

Accept-Language language supported by client

Accept-Encoding encodings supported by client, e.g. gzip or deflate

User-Agent browser details, supplies server with information about the type of browser making the request

Connection: Keep-Alive browser is requesting the use of persistent TCP connections

2.3 Response Headers

Content-Type MIME-Type of content

Content-Length size of body (in bytes)

Content-Encoding compression algorithms

Location used by redirections

Date timestamp when the response was created

Last-Modified modification date of resource (assumed by server)

Expires date after which the result is considered stale

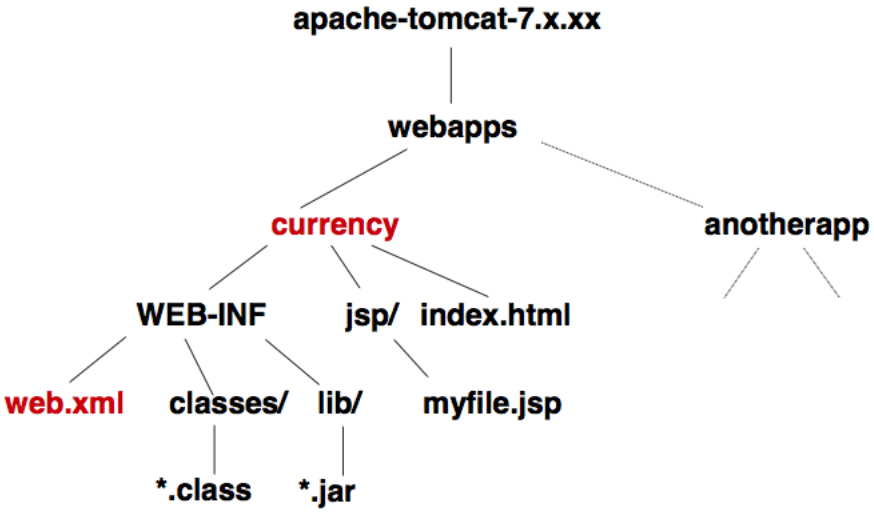
Server information about the server

Transfer-Encoding specifies type of transformation

Cache-Control information about cache handling (e.g. no-cache disables caching)

WWW-Authenticate information about authentication method

2.4 Servlet



Listing 2: Servlet Example

```
public class Converter extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse response) throws
      IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String amount = request.getParameter("amt");
    String from = request.getParameter("from");
    String to = request.getParameter("to");
    String res = computeResult(amount, from, to);
    out.println("<html>\n<body bgcolor=\"white\">");
    out.println("<h1>Currency Converter</h1>");
    out.println(amount + " " + from + " = " + res);
    out.println("</body>\n</html>");
  }
  String computeResult(String amount, String from, String to){...}
}
```

Listing 3: web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/
    XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.
    sun.com/xml/ns/javaee/web-app_3_0.xsd" version="3.0">
    <servlet>
        <servlet-name>CurrencyConverter</servlet-name>
        <servlet-class>ch.fhnw.ds.Converter</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>CurrencyConverter</servlet-name>
    <url-pattern>/convert</url-pattern>
  </servlet-mapping>
</web-app>
```

3 Webservices

3.1 XML-RPC

simples RPC Protokoll über HTTP, benötigt keine lange Einarbeitungszeit

3.1.1 Primitive Datentypen

- int, i4 — signed 32bit Integer
- string — ASCII string (no latin1)
- boolean — either 0 or 1
- double — double-precision floating point number
- dateTime.iso8601 — z.B. 20050717T14:08:14
- base64 — raw binary data, base64 encoded

Listing 4: Beispiele

```
<i4>13</i4>
<boolean>0</boolean>
```

3.1.2 Structs

- Struct enthält Members mit Name und Wert.
- können rekursiv sein (Structs die Structs enthalten)

Listing 5: Struct Beispiel

```
<i4>13</i4>
<struct>
  <member>
    <name>from</name>
    <value><i4>-5</i4></value>
  </member>
  <member>
    <name>to</name>
    <value><i4>5</i4></value>
  </member>
</struct>
```

3.1.3 Arrays

- Element-Typen können gemischt werden

Listing 6: Array

```
<array>
 <data>
  <value><i4>-5</i4></value>
  <value><string>44</string></value>
  <value><boolean>1</boolean></value>
 </data>
</array>
```

3.1.4 XML-RPC Request

Listing 7: Method Call

```
<?xml version="1.0" encoding="UTF-8"?>
<methodCall>
  <methodName>Echo.getEcho</methodName>
    <params>
      <param>
        <value>World</value>
      </param>
    </params>
</methodCall>
```

3.1.5 XML-RPC Response

Listing 8: Single Result

```
<?xml version="1.0" encoding="UTF-8"?>
<methodResponse>
  <params>
    <param>
      <value>Hello World, welcome to XML-RPC</value>
    </param>
  </params>
</methodResponse>
```

Als Resultat kann nur ein Wert zurückkommen, dieser kann jedoch auch ein Struct oder ein Array sein.

Listing 9: Fault Result

```
<?xml version="1.0" encoding="UTF-8"?>
<methodResponse>
  <fault>
    <value>
      <struct>
        <member>
          <name>faultCode</name>
          <value><i4>0</i4></value>
        </member>
        <member>
          <name>faultString</name>
          <value>No such handler: Echo.foo</value>
        </member>
      </struct>
    </value>
  </fault>
</methodResponse>
```

3.1.6 Apache XML-RPC Sample Server

Listing 10: Sample Server

```
import org.apache.xmlrpc.server.*;
import org.apache.xmlrpc.webserver.WebServer;

public class HelloServer {
  public static void main (String [] args) throws Exception {
    PropertyHandlerMapping phm = new PropertyHandlerMapping();
    phm.addHandler("Echo", ch.fhnw.ds.xmlrcp.echo.EchoImpl.class);
    WebServer server = new WebServer(80);
    XmlRpcServer xmlRpcServer = server.getXmlRpcServer();
    xmlRpcServer.setHandlerMapping(phm);
    server.start();
    System.out.println("Server started at port 80");
```

```
13    }
    }
```

Listing 11: Handler Class Server

```
1 public class EchoImpl {
  public String getEcho(String name) {
  return "[XML-RPC] Hello "+name+", welcome to XML-RPC";
  }
  }
```

Nur Instanzmethoden der Handlerklasse sind zugreifbar. Keine void Methoden. Public Default Constructor zwingend.

3.1.7 Apache XML-RPC Client

Listing 12: Handler Class Server

```
  import java.util.*;
  import org.apache.xmlrpc.*;

  public class HelloClient {
5   public static void main (String [] args) throws Exception {
      XmlRpcClientConfigImpl config = new XmlRpcClientConfigImpl();
      config.setServerURL(new URL("http://localhost/xmlrpc"));
      XmlRpcClient client = new XmlRpcClient();
      client.setConfig(config);
10    List params = new ArrayList();
      params.add(args[0]);
      Object result = client.execute("Echo.getEcho", params );
      System.out.println("The result is: "+result.toString());
    }
15 }
```

3.2 SOAP

3.2.1 WSDL (Web Services Description Language)

(früher Web Services Definition Language)
WSDL ist eine XML basierte Interface Beschreibungssprache die genutzt wird um die Funktionalität von Webservices zu beschreiben. Eine WSDL Beschreibung eines Webservices enthält:

- Wie der Service aufgerufen werden kann
- Welche Parameter er erwartet
- Welche Datenstruktur er zurückgibt

3.2.2 JAX-WS (Java API for XML Web Services

JAX-WS ist eine Java API um Webservices zu erstellen. Es ist Teil der Java EE (Enterprise Edition) Plattform von Sun Microsystems. Wie auch andere Java EE APIs nutzt JAX-WS Annotationen (@Webservice, @WebMethod usw). Basiert auf SOAP. Nur WSDL 1.1 unterstützt.

3.2.3 Anleitung JAX-WS

1. Interface erstellen

Listing 13: Interface erstellen

```
package ch.fhnw.ds.jaxws.server;
import javax.jws.WebService;
@WebService
public interface HelloService {
  String sayHello(@WebParam(name = "name") String name);
}
```

2. Interface implementieren

Listing 14: Interface implementieren

```
package ch.fhnw.ds.jaxws.server;
import java.util.Date;
@WebService
public class HelloServiceImpl implements HelloService {
  @Override
  public String sayHello(@WebParam(name = "name") String name){
    return "Hello " + name + " from SOAP at " + new Date();
  }
}
```

3. Java Objekte für XML Requests & Responses generieren % wsgen -cp bin -keep -s src -d bin ch.fhnw.ds.jaxws.server.HelloSer

- cp ¡path¿ classpath
- keep keep generated files
- s ¡path¿ path where to place generated source files
- d ¡path¿ path where to place generated output files
- ¡SEI¿ specify a SIB (service implementation bean)

Listing 15: SayHello

```
package ch.fhnw.ds.jaxws.server.jaxws;
import ...;
@XmlRootElement(name = "sayHello",
    namespace = "http://server.jaxws.ds.fhnw.ch/")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "sayHello",
    namespace = "http://server.jaxws.ds.fhnw.ch/")
public class SayHello {

  @XmlElement(name = "name", namespace = "")
  private String name;
  public String getName() { return this.name; }
  public void setName(String name) { this.name = name; }
}
```

Listing 16: SayHelloResponse

```
package ch.fhnw.ds.jaxws.server.jaxws;
import ...;
@XmlRootElement(name = "sayHelloResponse",
    namespace = "http://server.jaxws.ds.fhnw.ch/")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "sayHelloResponse",
    namespace = "http://server.jaxws.ds.fhnw.ch/")
public class SayHelloResponse {

  @XmlElement(name = "return", namespace = "")
  private String _return;
  public String getReturn() { return this._return; }
  public void setReturn(String _return) {
    this._return = _return;
```

```
15    }
  }
```

4. Service publishen

Listing 17: HelloServicePublisher

```
  package ch.fhnw.ds.jaxws.server;
  import javax.xml.ws.Endpoint;

4 public class HelloServicePublisher {
    public static void main(String[] args){
      Endpoint.publish(
        "http://127.0.0.1:9876/hs", // publication URI
        new HelloServiceImpl()); // SIB instance
9     System.out.println("service published");
    }
  }
```

5. Generierte Webservice Definition anschauen http://localhost:9876/hs?wsdl

Listing 18: WSDL

```
  <?xml version="1.0" encoding="UTF-8"?> <definitions   xmlns:soap="http://schemas.xmlsoap.
      org/wsdl/soap/"  xmlns:tns="http://server.jaxws.ds.fhnw.ch/"   xmlns:xsd="http://www.w3
      .org/2001/XMLSchema"  xmlns="http://schemas.xmlsoap.org/wsdl/"  targetNamespace="http
      ://server.jaxws.ds.fhnw.ch/"   name="HelloServiceImplService">
  <types>
    <xsd:schema>
4     <xsd:import namespace="http://server.jaxws.ds.fhnw.ch/" schemaLocation="http://
          localhost:9876/hs?xsd=1">
      </xsd:import>
    </xsd:schema>
  </types>
  <message name="sayHello">
9 <part name="parameters"
  element="tns:sayHello">
  </part>
  </message>
  <message name="sayHelloResponse">
14 <part name="parameters"
  element="tns:sayHelloResponse">
  </part>
  </message>
  <portType name="HelloServiceImpl">
19 <operation name="sayHello">
  <input message="tns:sayHello"></input>
  <output message="tns:sayHelloResponse"></output>
  </operation>
  </portType>
24 <binding name="HelloServiceImplPortBinding"
  type="tns:HelloServiceImpl">
  <soap:binding
  transport="http://schemas.xmlsoap.org/soap/http"
  style="document">
29 </soap:binding>
  <operation name="sayHello">
  <soap:operation soapAction=""></soap:operation>
  <input>
  <soap:body use="literal"></soap:body>
34 </input>
  <output>
  <soap:body use="literal"></soap:body>
  </output>
  </operation>
39 </binding>
  <service name="HelloServiceImplService">
  <port name="HelloServiceImplPort"
  binding="tns:HelloServiceImplPortBinding">
```

```
  <soap:address location="http://localhost:9876/hs">
44 </soap:address>
  </port>
  </service>
  </definitions>

49 Referenzierte Schema Definition:

  <?xml version="1.0" encoding="UTF-8"?>
  <xs:schema xmlns:tns="http://server.jaxws.ds.fhnw.ch/" xmlns:xs="http://www.w3.org/2001/
      XMLSchema"
  version="1.0"
54 targetNamespace="http://server.jaxws.ds.fhnw.ch/">
  <xs:element name="sayHello" type="tns:sayHello"></xs:element> <xs:element name="
      sayHelloResponse" type="tns:sayHelloResponse"></xs:element>
  <xs:complexType name="sayHello">
    <xs:sequence>
      <xs:element name="name" type="xs:string" minOccurs="0"></xs:element>
59  </xs:sequence>
  </xs:complexType>
  <xs:complexType name="sayHelloResponse">
    <xs:sequence>
      <xs:element name="return" type="xs:string" minOccurs="0"></xs:element>
64  </xs:sequence>
   </xs:complexType>
  </xs:schema>
```

6. Client Proxy generieren % wsimport -keep -p ch.fhnw.ds.jaxws.client.jaxws -d bin -s src

- -keep keep generated files
- -p ¡package¿ specify (overwrite) target package
- -s ¡path¿ path where to place generated source files
- -d ¡path¿ path where to place generated output files
- ¡WSDL¿ Web Service Definition
- =¿ HelloServiceImpl generated interface
- =¿ HelloServiceImplService factory class

7. Client Applikation schreiben

Listing 19: Client

```
  package ch.fhnw.imvs.client;
  import ch.fhnw.imvs.client.jaxws.HelloServiceImpl;
3 import ch.fhnw.imvs.client.jaxws.HelloServiceImplService;

  public class Client {
    public static void main(String[] args) {
      HelloServiceImplService service =
8       new HelloServiceImplService();
      HelloServiceImpl port =
        service.getHelloServiceImplPort();
      String result = port.sayHello("Dominik");
      System.out.println(result);
13  }
  }
```

8. JAX-WS HTTP Request

Listing 20: ...

```
1 POST /hs HTTP/1.1           HTTP Request
  Accept: text/xml, multipart/related
  User-Agent: JAX-WS RI 2.2.4-b01
  Host: 127.0.0.1:9877
```

```
  Connection: keep-alive
6 Content-Length: 209
  Content-Type: text/xml; charset=utf-8   SOAP HTTP Binding
  SOAPAction: "http://server.jaxws.ds.fhnw.ch/HelloServiceImpl/sayHelloRequest"

  <?xml version="1.0" ?>          SOAP Payload
11 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
      <ns2:sayHello xmlns:ns2="http://server.jaxws.ds.fhnw.ch/">
        <name>Dominik</name>
      </ns2:sayHello>
16  </S:Body>
  </S:Envelope>
```

9. JAX-WS HTTP Response

Listing 21: ...

```
1 HTTP/1.1 200 OK           HTTP Response
  Transfer-encoding: chunked
  Content-type: text/xml; charset=utf-8
  Date: Mon, 18 Mar 2013 00:06:44 GMT
  Content-Length: 277
6
  <?xml version="1.0" ?>          SOAP Payload
  <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
  <ns2:sayHelloResponse
11 xmlns:ns2="http://server.jaxws.ds.fhnw.ch/">
  <return>
  Hello Dominik from SOAP at Mon Mar 18 01:06:44 CET 2013
  </return>
  </ns2:sayHelloResponse>
16 </S:Body>
  </S:Envelope>
```

3.3 Vergleich SOAP und XML-RPC

Feature	XML-RPC	SOAP
Structs	yes	yes
Arrays	yes	yes
Named structs & arrays	no	yes
Short learning curve	yes	no
Developer specified character set	no	yes
Developer defined data types	no	yes
Can specify recipient	no	yes
Require client understanding	no	yes
Message specific processing instructions	no	yes

4 Representation State Transfer (REST)

4.1 Prinzipien

- Addressability - Give everything an ID
- Uniform, Constrained Interface
- Representation-oriented
- Link things together - Use Resource references
- Stateless communications - Resources hold state
- Use standard HTTP methods

4.1.1 Addressability

- Resources = key abstractions in REST
- Each resource is addressable via a URI

Listing 22: REST example

```
http://www.example.com/customers/1234 http://www.example.com/customers?lastName=Meier http
    ://www.example.com/orders/2011/03/445245 http://www.example.com/products/ http://www.
    example.com/products/4711
```

4.1.2 CRUD

READ

- HTML: **GET, HEAD**
- Retrieve information in a particular representation
- No side effects, possibly cached
- May contain query parameters

CREATE

- HTML: **POST**
- Create a new sub-resource (without known ID)

CREATE & UPDATE

- HTML: **PUT**
- Update an existing resource
- Create a new resource with a known ID

DELETE

- HTML: **DELETE**
- Remove resources

OPTIONS

Returns allowed operations

4.1.3 Representation-oriented

Allow multiple representations of a resource:

- text/html
- text/plain
- application/json
- application/xml

4.1.4 Link things together

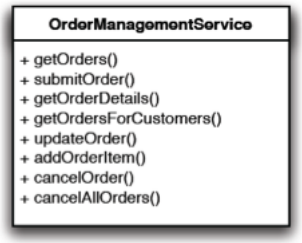
References to other resources may be used in representations

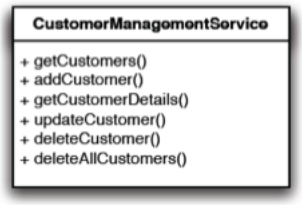
Listing 23: example

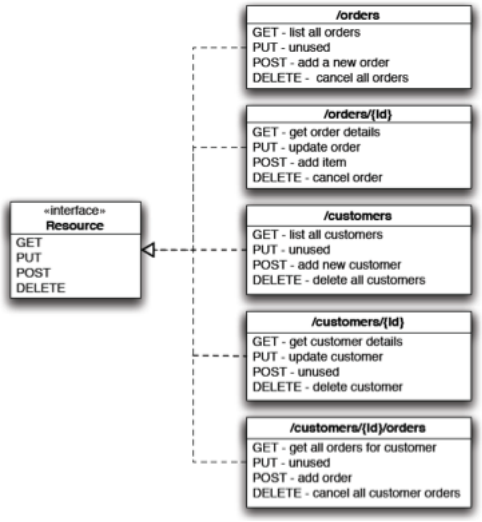
```
<order>
   <date>16.03.2013</date>
   <amount>23</amount>
   <product ref="http://example.com/products/4711" />
   <customer ref="http://example.com/customers/1234" />
</order>
```

4.2 SOAP vs REST

REST	SOAP
Resource Oriented	Service oriented
Messages represented in different formats	Messages represented in XML
HTTP used as protocol	Can be bound to different protocols
HTTP verbs are used for access and manipulation	Access and Manipulation is service specific
HTTP error codes are used as error messages	Fault Elements in SOAP body describes errors
No formal interface description language	WSDL is used as interface description language
GET requests can be cached in a proxy	All requests are POST requests, no caching!







4.3 JAX-RS

4.3.1 HTTP Methods

- Annotate resource class methods with HTTP method annotations @GET, @POST, @PUT, @DELETE, @HEAD
- Java method name has no significance
- Return value is mapped to the response (void = no response)

Listing 24: JAX-RS Service

```
@Singleton
@Path("/polls")
public class DoodlePoolResource {
  @GET
  @Path("{id}")
  public String getPoll(@PathParam("id") String key){ ... }

  @PUT
  @Path("{id}")
  public String putPoll(@PathParam("id") String key){ ... }

  @DELETE
  @Path("{id}")
  public void deletePoll(@PathParam("id") String key){ ... }
}
```

4.3.2 Injection

- Automatic Type Conversion from Strings to
  - Primitive types (int, short, float, double, byte, char, boolean)
  - Classes T which have a constructor with a single String parameter
  - Classes T which contain a static method T valueOf(String arg)
- Default values may be defined with @DefaultValue for the case that the parameter is not passed with the request

PathParam
Allows to extract values from URI template parameters

MatrixParam
Allows to extract matrix parameters (/images/cars;color=blue/2010/)

QueryParam
Allows to extract query parameters added to a URI

FormParam
Allows to extract values from posted form data

HeaderParam
Allows to extract request headers

CookieParam
Allows to extract values from HTTP cookies

4.3.3 Content Negotiation

@Produces declares type of result, default: all types are supported
@Produces({"text/plain", "text/html"})

@Consumes declares type which is accepted (PUT / POST)
@Consumes(”application/x-www-form-urlencoded”)

4.4 Data Binding

Listing 25: XStream Provider

```
@Provider
@Consumes("application/xstream")
@Produces("application/xstream")
public class XStreamProvider implements MessageBodyReader<Object>, MessageBodyWriter<Object
    >{
  private XStream xstream = new XStream(new DomDriver());

  public boolean isReadable(Class<?> type, Type genericType, Annotation[] annotations,
      MediaType mimeType) {
    return true;
  }
  public Object readFrom(Class<Object> type, Type genericType, Annotation[] annotations,
      MediaType mimeType, MultivaluedMap<String, String> httpHeaders, InputStream
      entityStream) {
    return xstream.fromXML(entityStream);
  }
  public boolean isWriteable(Class<?> type, Type genericType, Annotation[] ann, MediaType
      mimeType) {
    return true;
  }

  public long getSize(Object object, Class<?> type, Type genericType, Annotation[] ann,
      MediaType mimeType) {
    return -1;// size not yet known
  }

  public void writeTo(Object object, Class<?> type, Type genericType, Annotation[] ann,
      MediaType mimeType, MultivaluedMap<String, Object> httpHeaders, OutputStream
      entityStream) {
    xstream.toXML(object, entityStream);
  }
}
```

Listing 26: XStream Example

```
public class Client {
  public static void main(String[] args) {
  ClientConfig config = new DefaultClientConfig();
  config.getClasses().add(XStreamProvider.class);
  Client c = Client.create(config);
  WebResource r = c.resource("http://localhost:9998/msg");
  Msg msg = new Msg("Hello from XClient");
  r.type("application/xstream").put(msg);
  Msg res = r.accept("application/xstream").get(Msg.class);
  System.out.println(res);
  System.out.println(res.getText());
  System.out.println(res.getDate());
  }
}
```

4.5 JAXB

Java Architecture for XML Binding, kurz JAXB, ist eine Programmschnittstelle in Java, die es ermöglicht, Daten aus einer XML-Schema-Instanz heraus automatisch an Java-Klassen zu binden, und diese Java-Klassen aus einem XML-Schema heraus zu generieren. Diesen Vorgang nennt man XML-Datenbindung.

Listing 27: Unmarshalling

```
1
  JAXBContext jc = JAXBContext.newInstance("com.acme.foo:com.acme.bar");
  Unmarshaller u = jc.createUnmarshaller();
  FooObject fooObj = (FooObject) u.unmarshal(new File("foo.xml"));
  BarObject barObj = (BarObject) u.unmarshal(new File("bar.xml"));
```

Listing 28: Marshalling

```
  Marshaller m = jc.createMarshaller();
  m.marshal(fooObj, System.out);
```

5 Code Samples

5.1 Socket

Listing 29: Client

```
public class SocketBankDriver implements RemoteDriver {
  private RemoteBank bank = null;
  private Socket socket;
  private PrintWriter out;
  private BufferedReader in;
  public void connect(String[] args) throws IOException {
    socket = new Socket(args[0], Integer.valueOf(args[1]));
    bank = new RemoteBank(this);
    out = new PrintWriter(socket.getOutputStream());
    in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
  }
  public void disconnect() throws IOException{
    sendCommand("disconnect");
    socket.close(); bank = null;
  }
  public bank.Bank getBank(){ return bank; }
  public String sendCommand(String command) throws IOException {
    return sendCommand(command, "");
  }
  public String  sendCommand(String command, String param) throws IOException {
    out.println(command + ":" + param);
    out.flush();
    String input = in.readLine();
    String result = "";
    if (input != null) {
      String[] temp = input.split(":");
      result = (temp.length > 1 ? temp[1] : null );
    }
    return result;
  }
}
```

Listing 30: Server

```
public class SocketBankServer {
  private final int port;
  private MyBank bank;

  public SocketBankServer(int p) {
    port = p; bank = new MyBank();
  }
  public void start() {
    try (ServerSocket server = new ServerSocket(port)) {
      while (true) {
        Socket s = server.accept();
        Thread t = new Thread(new SocketHandler(s, bank));
        t.start();
      }
    } catch (IOException e) { System.err.println(e.getMessage()); }
  }
  public static void main (String args[]) {
    SocketBankServer server = new SocketBankServer(Integer.valueOf(args[0]));
    server.start();
  }
}
public class SocketHandler implements RequestHandler, Runnable {
  private final Socket socket;
  private CommandHandler cHandler;
  private boolean running = false;
  public SocketHandler(Socket s, MyBank b) {
    socket = s;
    cHandler = new CommandHandler(b, this);
  }
```

```
    public void run() {
      running = true;
      System.out.println("handle connection from " + socket);
      try {
        while (running) {
          Request req = receiveResult();
          String result = cHandler.handleCommand(req.getCommand(), req.getParam());
          if (!req.getCommand().equals("disconnect")) sendResponse(req.getCommand(), result);
        }
      } catch (IOException e) { }
          finally {
        try { socket.close();
        } catch (IOException e) { }
      }
    }
    public void stop() throws IOException {
      socket.close(); running = false;
    }
    public void sendResponse(String command, String param) throws IOException {
      PrintWriter out = new PrintWriter(socket.getOutputStream());
      System.out.println("Server send: '"+command+":"+param+"'");
      out.println(command + ":" + param);
      out.flush();
    }
    public Request receiveResult() throws IOException {
      BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
      String input = in.readLine();
      System.out.println("Server receive: '"+input+"'");
      String[] temp = input.split(":");
      return new Request(temp[0],temp.length > 1 ? temp[1] : "");
    }
  }
```

5.2 Internet

Listing 31: Client

```
  public class HttpBankDriver implements RemoteDriver {
    private String address;
    private RemoteBank bank = null;
    private URL url = null;
    public void connect(String[] args) throws IOException {
      bank = new RemoteBank(this);
      address = args[0];
      System.out.println(address);
    }
    public void disconnect() throws IOException {
      sendCommand("disconnect");
    }
    public Bank getBank() { return bank; }
    public String sendCommand(String command) throws IOException {
      return sendCommand(command, "");
    }
    public String sendCommand(String command, String param) throws IOException {
      url = new URL(address);
      HttpURLConnection c = (HttpURLConnection) url.openConnection();
      c.setRequestProperty("User-Agent", "SocketBank/HTTPBankDriver");
      c.setRequestMethod("GET");
      c.setUseCaches (false);
        c.setDoInput(true);
        c.setDoOutput(true);
        //Send request
        DataOutputStream wr = new DataOutputStream(c.getOutputStream ());
        wr.writeBytes("cmd="+command+"&param="+param);
        wr.flush ();
        wr.close ();
        BufferedReader r = new BufferedReader(new InputStreamReader(c.getInputStream()));
        String line;
      StringBuffer response = new StringBuffer();
        while((line = r.readLine()) != null) { response.append(line); }
```

```
34        r.close();
          String input = response.toString();
        String[] temp = input.split(":");
        return (temp.length > 1 ? temp[1] : null );
      }
39  }
```

Listing 32: Server

```
1   public class HttpBankServer {
        private final int port;
      private MyBank bank;
      public HttpBankServer(int p) {
        port = p;
6       bank = new MyBank();
      }
      public void start() throws IOException {
        HttpServer server = HttpServer.create(new InetSocketAddress(port), 0);
        server.createContext("/bank", new HttpRequestHandler(bank)).getFilters().add(new
            ParameterParser());
11      server.start();
      }
        public static void main(String[] args) throws IOException {
        HttpBankServer server = new HttpBankServer(Integer.valueOf(args[0]));
        server.start();
16    }
    }
    public class HttpRequestHandler implements HttpHandler, RequestHandler {
      private CommandHandler cHandler;
      private HttpExchange exchange;
21    public HttpRequestHandler(MyBank b) {
        cHandler = new CommandHandler(b, this);
      }
      public void handle(HttpExchange httpExchange) throws IOException {
        exchange = httpExchange;
26      exchange.getResponseHeaders().add("Content-type", "text/html");

        Map<String, Object> params = (Map<String, Object>) httpExchange.getAttribute("
            parameters");
        String cmd = (String) params.get("cmd");
        String param = (String) params.get("param");
31      String result = cHandler.handleCommand(cmd, param);
        String response =  cmd + ":" + result;
        exchange.sendResponseHeaders(200, response.length());
        OutputStream os = exchange.getResponseBody();
        os.write(response.getBytes());
36      os.close();
      }
    }
```

5.3 XmlRpc

Listing 33: Client

```
1   public class XmlRpcBankDriver implements RemoteDriver {
      private RemoteBank bank = null;
      private XmlRpcClient client = null;
      public void connect(String[] args) throws IOException {
        bank = new RemoteBank(this);
6       String address = args[0];
        XmlRpcClientConfigImpl config = new XmlRpcClientConfigImpl();
        config.setServerURL(new URL(address));
        client = new XmlRpcClient();
        client.setConfig(config);
11    }
      public void disconnect() throws IOException {
        sendCommand("disconnect");
      }
```

```
    public Bank getBank() { return bank; }
16  public String sendCommand(String command) throws IOException {
      return sendCommand(command, "");
    }
    public String sendCommand(String command, String param) throws IOException {
      System.out.println("Client send: '"+command+":"+param+"'");
21    List<Object> params = new ArrayList<Object>();
      params.add(command);
      params.add(param);
      String result;
      try {
26      result = (String) client.execute("Bank.handle", params );
        System.out.println("Client receive: '"+result+"'");
      } catch (XmlRpcException e) { }
      return (result.length() > 0 ? result : null );
    }
31
  }
```

Listing 34: Server

```
  public class XmlRpcBankServer {
    private final int port;
3   private static MyBank bank;
    public XmlRpcBankServer(int p) {
      port = p;
      bank = new MyBank();
    }
8   public static MyBank getBank() { return bank; }
    public void start() throws XmlRpcException, IOException {
      WebServer server = new WebServer(port);
      XmlRpcServer xmlRpcServer = server.getXmlRpcServer();
      PropertyHandlerMapping phm = new PropertyHandlerMapping();
13    phm.addHandler("Bank", ch.fhnw.jfmk.bank.server.handler.XmlRpcHandler.class);
      xmlRpcServer.setHandlerMapping(phm);
          server.start();
    }
    public static void main(String[] args) throws XmlRpcException, IOException {
18    XmlRpcBankServer server = new XmlRpcBankServer(Integer.valueOf(args[0]));
      server.start();
    }
  }
  public class XmlRpcHandler implements RequestHandler{
23  private CommandHandler cHandler;
    public XmlRpcHandler () {
      cHandler = new CommandHandler(XmlRpcBankServer.getBank(), this);
    }
    public String handle(String command, String param) throws IOException {
28    return cHandler.handleCommand(command, param);;
    }
  }
```

5.4 REST

Listing 35: Client

```
  public class RestBankDriver implements RemoteDriver {
    private RemoteBank bank;
    private Client client;
    private String address;
5   private WebResource res;
    public void connect(String[] args) throws IOException {
      bank = new RemoteBank(this);
      client = Client.create();
      address = args[0];
10    if (!address.endsWith("/")) address += "/";
    }
    public void disconnect() throws IOException {
```

```
    if (client != null) client.destroy();
  }
  public Bank getBank() { return bank; }
  public String sendCommand(String command) throws IOException {
    return sendCommand(command, "");
  }
  public String sendCommand(String command, String param) throws IOException {
    System.out.println("Client send: '" + command + ":" + param + "'");
    String mime = "application/plain", result = "";
    MultivaluedMap<String, String> formData = new MultivaluedMapImpl();
    String[] params = param.split(";");
    switch (command) {
    case "createAccount":
      res = client.resource(address + "accounts/create");
      formData.add("owner", param);
      result = res.accept(mime).post(String.class, formData);
      break;
    case "closeAccount":
      res = client.resource(address + "accounts/close/" + param);
      result = res.accept(mime).delete(String.class);
      break;
    case "getOwner":
      res = client.resource(address + "accounts/owner/" + param);
      result = res.accept(mime).get(String.class);
      break;
    case "getAccountNumbers":
      res = client.resource(address + "accounts");
      result = res.accept(mime).get(String.class);
      break;
    case "isActive":
      res = client.resource(address + "accounts/status/" + param);
      result = res.accept(mime).get(String.class);
      break;
    case "deposit":
      res = client.resource(address + "accounts/deposit/" + params[0]);
      formData.add("value", params[1]);
      result = res.accept(mime).put(String.class, formData);
      break;
    case "withdraw":
      res = client.resource(address + "accounts/withdraw/" + params[0]);
      formData.add("value", params[1]);
      result = res.accept(mime).put(String.class, formData);
      break;
    case "getBalance":
      res = client.resource(address + "accounts/balance/" + param);
      result = res.accept(mime).get(String.class);
      break;
    case "disconnect":
      bank = null;
      break;
    default:
      result = "unknown command";
      break;
    }
    return (result.length() > 0) ? result : null;
  }
}
```

Listing 36: Server

```
public class RestBankServer {
  private final int port;
  private HttpServer server;
  public RestBankServer(int p) throws IllegalArgumentException, NullPointerException,
      IOException {
    port = p;
    ResourceConfig rc = new ApplicationAdapter(new BankApplication(new MyBank()));
    server = GrizzlyServerFactory.createHttpServer("http://localhost:"+port, rc);
  }
  public void start() throws IOException {
```

```
      server.start();
      System.in.read();
      server.stop();
    }
    public static void main(String[] args) throws IOException {
      RestBankServer server = new RestBankServer(Integer.valueOf(args[0]));
      server.start();
    }
    public class BankApplication extends Application {
      private Set<Object> singletons = new HashSet<Object>();
      private Set<Class<?>> classes = new HashSet<Class<?>>();
      public BankApplication(MyBank b) {
        singletons.add(new RestHandler(b));
      }
      public Set<Class<?>> getClasses() { return classes; }
      public Set<Object> getSingletons() { return singletons; }
    }
  }

  @Singleton @Path("/bank")
  public class RestHandler implements RequestHandler {
    private CommandHandler cHandler;
    public RestHandler(MyBank b) {
      cHandler = new CommandHandler(b, this);
    }
      @POST @Path("/accounts/create") @Produces("application/plain")
    public String postCreateAccount(@FormParam("owner") String owner) throws IOException {
      return cHandler.handleCommand("createAccount", owner);
    }

    @DELETE @Path("/accounts/close/{id}") @Produces("application/plain")
    public String deleteCloseAccount(@PathParam("id") String id) throws IOException {
      return cHandler.handleCommand("closeAccount", id);
    }
    @GET @Path("/accounts/owner/{id}") @Produces("application/plain")
    public String getOwner(@PathParam("id") String id) throws IOException {
      return cHandler.handleCommand("getOwner", id);
    }
    @GET @Path("/accounts") @Produces("application/plain")
    public String getAccountNumbers() throws IOException {
      return cHandler.handleCommand("getAccountNumbers", "");
    }
    @GET @Path("/accounts/status/") @Produces("application/plain")
    public String getStatus() throws IOException {
      return "null";
    }
    @GET @Path("/accounts/status/{id}") @Produces("application/plain")
    public String getStatus(@PathParam("id") String id) throws IOException {
      return cHandler.handleCommand("isActive", id);
    }
    @PUT @Path("accounts/deposit/{id}") @Produces("application/plain")
    public String putDeposit(@PathParam("id") String id, @FormParam("value") String value)
        throws IOException {
      return cHandler.handleCommand("deposit", id + ";" + value);
    }
    @PUT @Path("accounts/withdraw/{id}") @Produces("application/plain")
    public String putWithdraw(@PathParam("id") String id, @FormParam("value") String value)
        throws IOException {
      return cHandler.handleCommand("withdraw", id + ";" + value);
    }
    @GET @Path("/accounts/balance/{id}") @Produces("application/plain")
    public String getBalance(@PathParam("id") String id) throws IOException {
      return cHandler.handleCommand("getBalance", id);
    }
  }
```