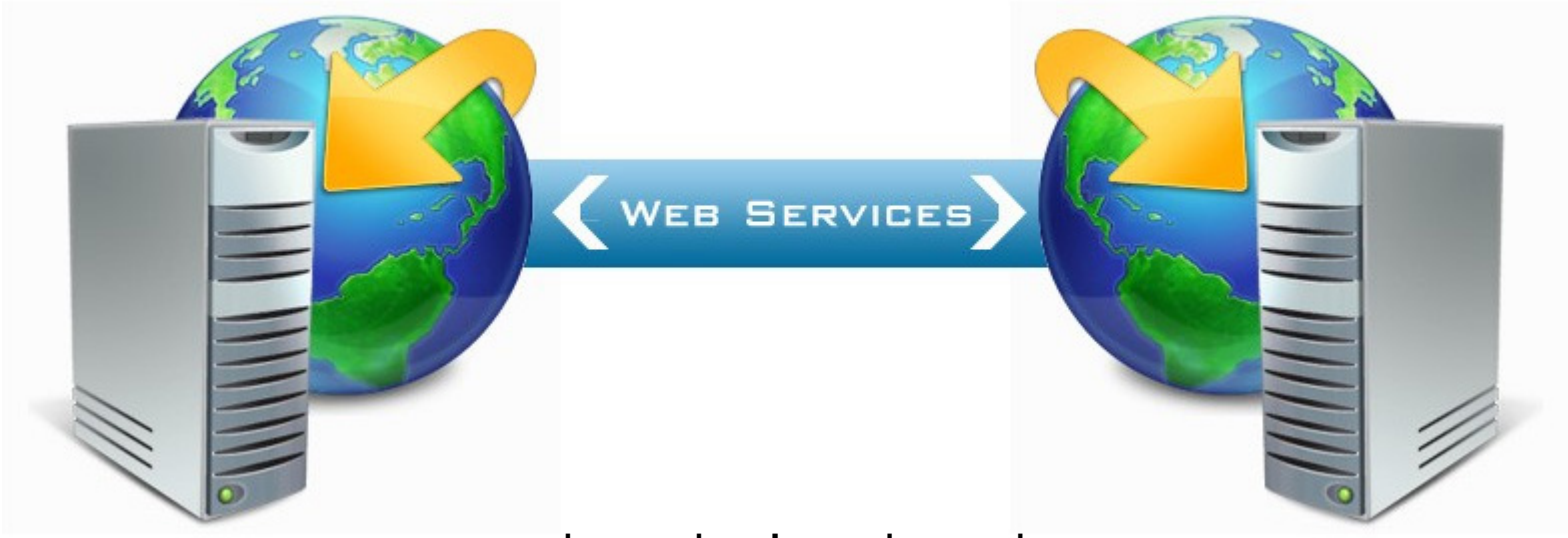
Web-Services

Modul 13
Web-Services



web service != web service

Modul 13
Web-Services

Web-Service

Protocol: HTTP
Data-Transfer: XML (SOAP) / JSON (REST)
Platform: C# / Java / Pearl / PHP / …

JSON (JavaScript Object Notation)

```
newObject = {
 "first": "Jimmy",
 "last": "James",
 "age": 29,
 "sex": "M",
 "salary": 63000,
 "registered": false
}
```

Modul 13
Web-Services: SOAP

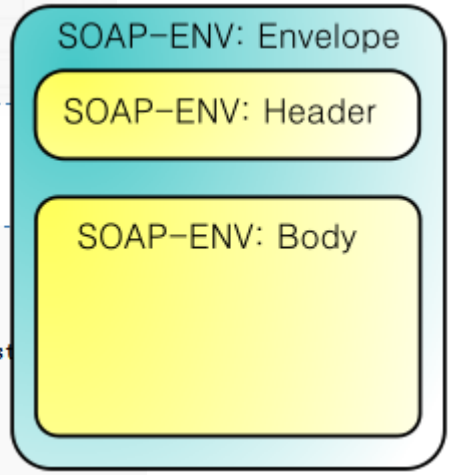
SOAP (*Simple Object Access Protocol)*

Request

```
<?xml version="1.0"?>
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope">
    <s:Body>
        <m:TitleInDatabase xmlns:m="http://www.lecture-db.de/soap">
            DOM, SAX und SOAP
        </m:TitleInDatabase>
    </s:Body>
</s:Envelope>
```

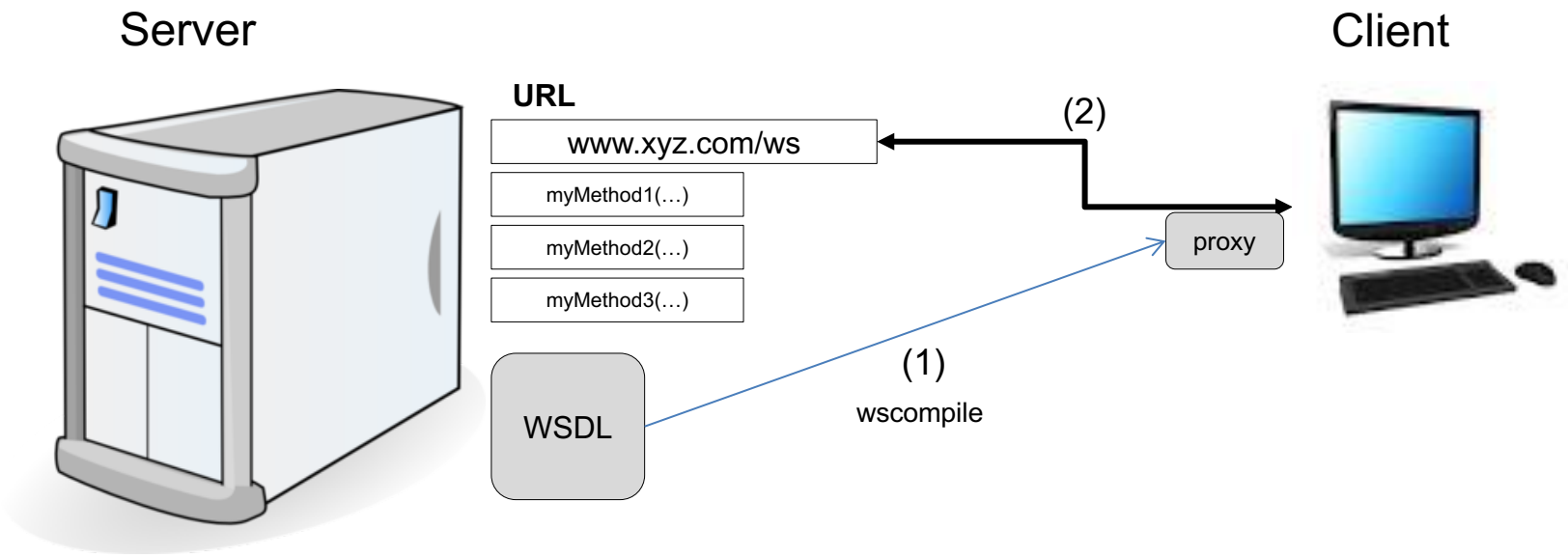
Response

```
<?xml version="1.0"?>
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope">
    <s:Header>
        <m:RequestID xmlns:m="http://www.lecture-db.de/soap">a3f5c109b</m:Request
    </s:Header>
    <s:Body>
        <m:DbResponse xmlns:m="http://www.lecture-db.de/soap">
            <m:title value="DOM, SAX und SOAP">
                <m:Choice value="1">Arbeitsbericht Informatik</m:Choice>
                <m:Choice value="2">Seminar XML und Datenbanken</m:Choice>
            </m:title>
        </m:DbResponse>
    </s:Body>
</s:Envelope>
```



Modul 13
Web-Services: SOAP

SOAP (*Simple Object Access Protocol)*



REST (Representational State Transfer)

Request

```
http://www.acme.com/phonebook/UserDetails/12345
```

```
http://www.acme.com/phonebook/UserDetails?firstName=John&lastName=Doe
```

Response

```
{
  "id": "12345"
  "name":"Hans",
  "vorname":"Mueller",
  "phone":"0123 456789"
}
```

JSON

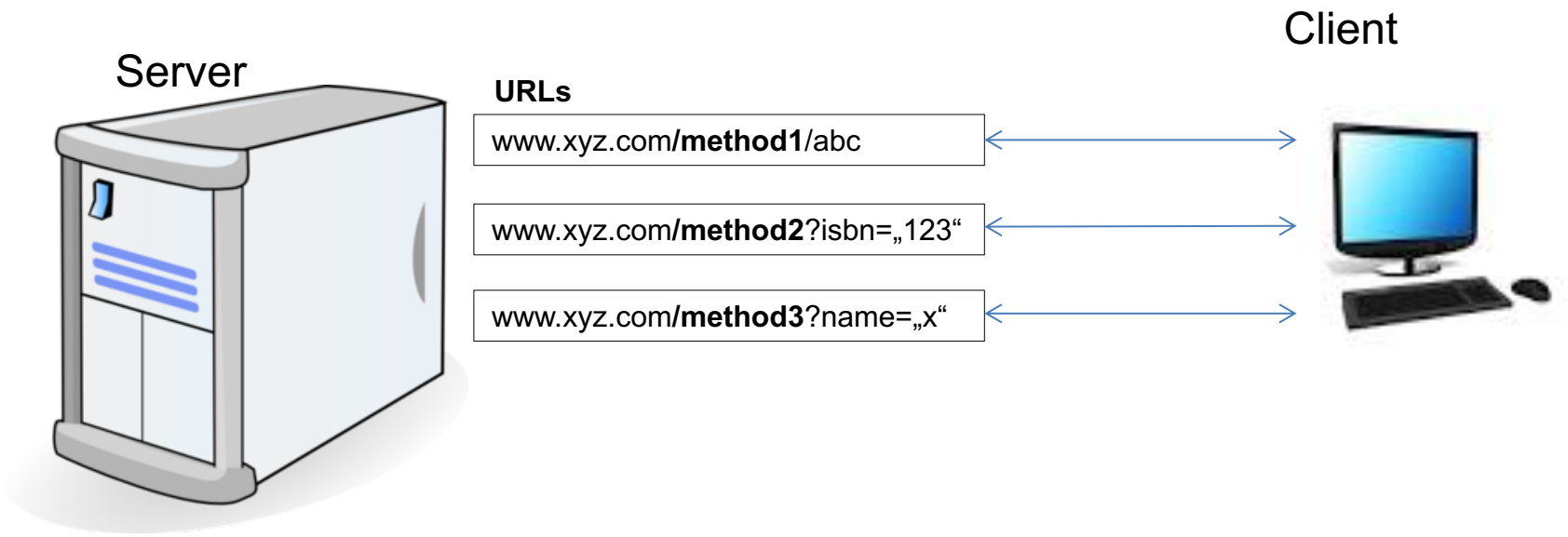
or

```
<parts-list>
 <part id="3322">
  <name>ACME Boomerang</name>
  <desc>
   Used by Coyote in <i>Zoom at the Top</i>, 1962
  </desc>
  <price currency="usd" quantity="1">17.32</price>
  <uri>http://www.acme.com/parts/3322</uri>
 </part>
 <part id="783">
  <name>ACME Dehydrated Boulders</name>
  <desc>
   Used by Coyote in <i>Scrambled Aches</i>, 1957
  </desc>
  <price currency="usd" quantity="pack">19.95</price>
  <uri>http://www.acme.com/parts/783</uri>
 </part>
</parts-list>
```

XML

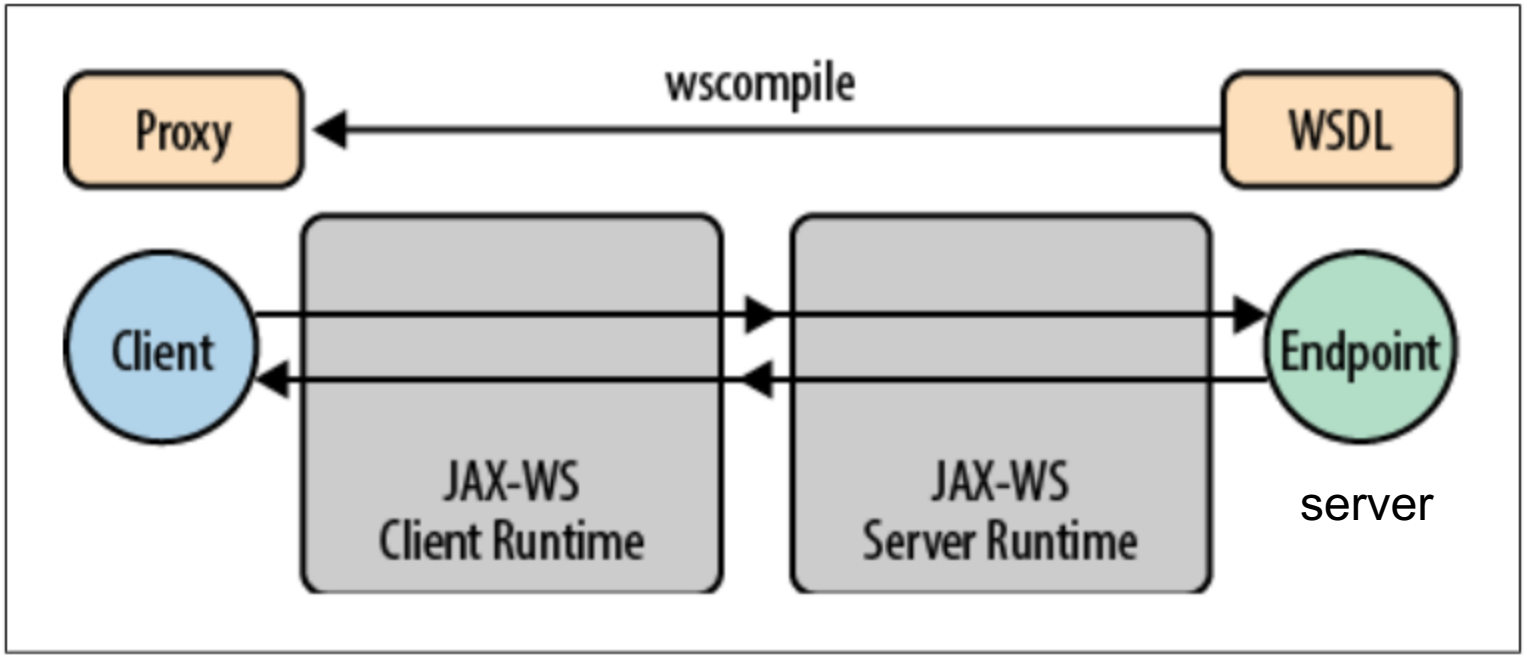
or …

REST (Representational State Transfer)

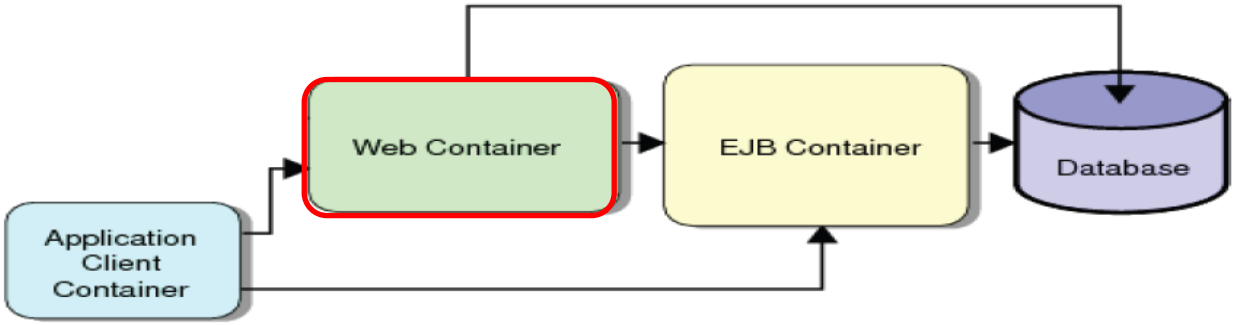


SOAP based
Web-Service in Java

Modul 13
JAX-WS Endpoints



Modul 13
JAX-WS Servlet Endpoints



A JAX-WS endpoint is not a servlet in the traditional sense. The JAX-WS framework provides a servlet implementation used to handle HTTP requests and process SOAP messages. A JAX-WS web endpoint:

- Is a standard Java class created just to provide web server functionality
- Is multi-threaded
- Does not require an EJB container

A JAX-WS web endpoint:

- Has a `@javax.jws.WebService` class annotation
- Contains business methods that are public and not final or static
- Has exposed web service methods that are annotated with `@javax.jws.WebMethod`
- Cannot be an abstract or final class
- Requires a default no-arg constructor

Modul 13
Simple Web Component Endpoint Example

```
package example;

import javax.jws.*;
@WebService public class SayHello {

     @WebMethod public String getGreeting(String name)  {
         return "Hello " + name;
     }
}
```

Modul 13
JAX-WS Servlet Endpoint Configuration

The `web.xml` file in your web archive (WAR) must be updated to provide a usable URL for the web service.

```
<servlet>
    <servlet-name>hello</servlet-name>
    <servlet-class>example.SayHello</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>hello</servlet-name>
    <url-pattern>/sayhello</url-pattern>
</servlet-mapping>
```

Modul 13
JAX-WS Allowed Data Types

JAX-WS unterstützen primitive Datentypen wie Strings automatisch.

Komplexe Objekte müssen durch Programmierung in XML-Format überführt werden. Hierzu kann *Java Architecture for XML Binding* **(JAXB)** verwendet werden.

The key to creating a web service client, in any language, is having a copy of the web service's WSDL. The WSDL describes the operations, arguments, and return values used in a web service.

The WSDL for a JAX-WS web service is generated automatically when the web service is deployed.

Although not part of the specification, to retrieve the generated WSDL from the Sun Application Server, you can use your browser by requesting a URL similar to:

```
http://localhost:8080/WSApp-war/SayHelloService?WSDL
```

To access a web service from a JAX-WS client you need:

- The WSDL
- A Proxy object to handle the creation of SOAP messages and HTTP communication. This proxy type class is know as a Port in JAX-WS.
- To generate the `Port` class or source code and any other required artifacts for the web service.

The JAX-WS reference implementation and the Sun Java Application Server provide an application known as `wsimport` to create all the required client code. The `wsimport` application can also be run as an Ant task.

Modul 13
A JAX-WS Client Example

In the following example, the Service class is used to instantiate a Port or proxy. The Port handles all SOAP message creation and transmission.

```
import javax.xml.ws.WebServiceRef;

public class WSTest {

public WSTest() { }

    public static void main(String[] args) {
       SayHelloService service = new SayHelloService();
       SayHello port = service.getSayHelloPort();
       System.out.println(port.sayHello("Duke"));
    }
}
```

http://www.programmableweb.com

http://www.webservicex.net/ws/default.aspx

http://predic8.de/soap/blz-webservice.htm

http://www.service-repository.com/

…

Example:
http://www.webservicex.net/geoipservice.asmx?WSDL
http://api.chartlyrics.com/apiv1.asmx?WSDL