Web services

Co to je?

- Část kódu přístupného přes standartní internetové protokoly
- Nezávislé na platformě
- Nezávislé na programovacím jazyce
- Pro komunikaci používají XML, JSIN, ...

Proč a kde je používáme

- Uznávaný a používaný standard
- Využívá standardní protokoly a formáty dat
- Podpora skrz mnoho jazyků (Java, .NET, C++, ...)
- Ideální pro spojování systémů

Architektura

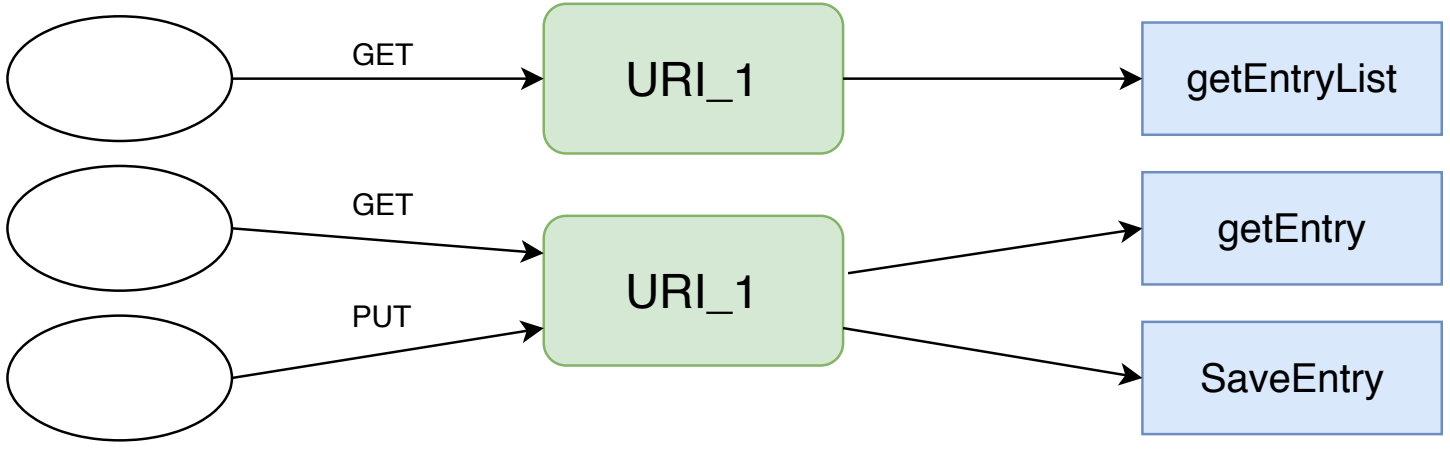
- Poskytovatel - Server kde služba běží
- Klient - Aplikace volající poskytovatele
- Registry - Uchovává informace o umístění služeb

REST a SOAP

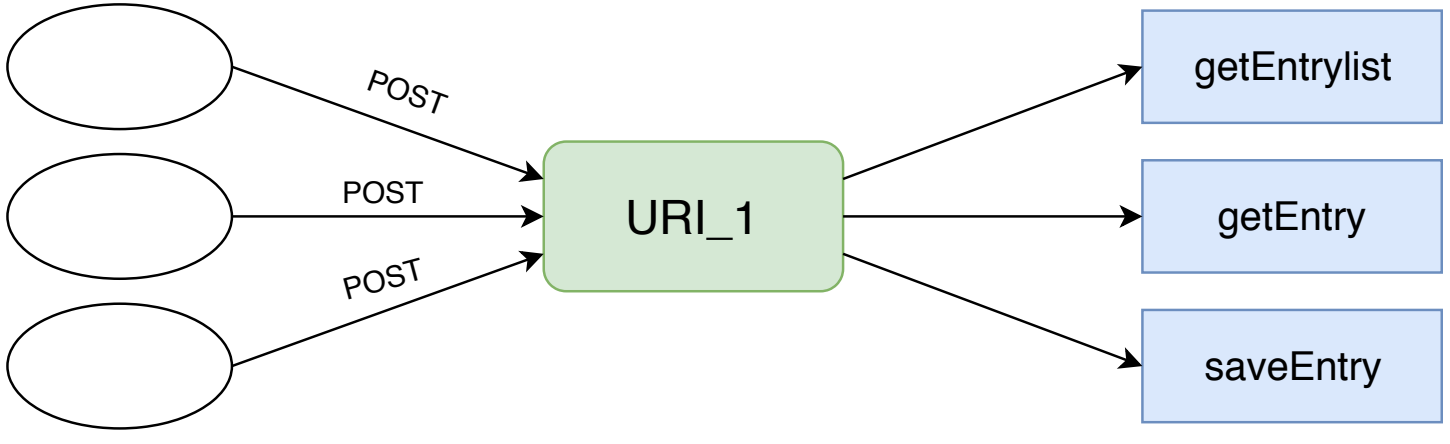
- SOAP - Simple Object Access Protocol
  - komunikační protokol
  - definován formát zpráv
  - postaveno výhradně na XML
- REST - Representational State Transfer
  - architektura - NE protokol
  - volný formát zpráv - ne jen XML
  - zdroj - dosažitelný na jednoznačné URL

REST a SOAP

REST



SOAP



REST a SOAP

REST	**SOAP**
architektonický vzor	protokol
přístup ke zdrojům	přístup ke službám
json a jiné datové formáty	XML
WADL - prakticky se nepoužívá	WSDL + XSD
nepodporuje transakce	podporuje transakce
menší objemy dat	náročnější na objemy dat
volání lze kešovat	volání nelze kešovat

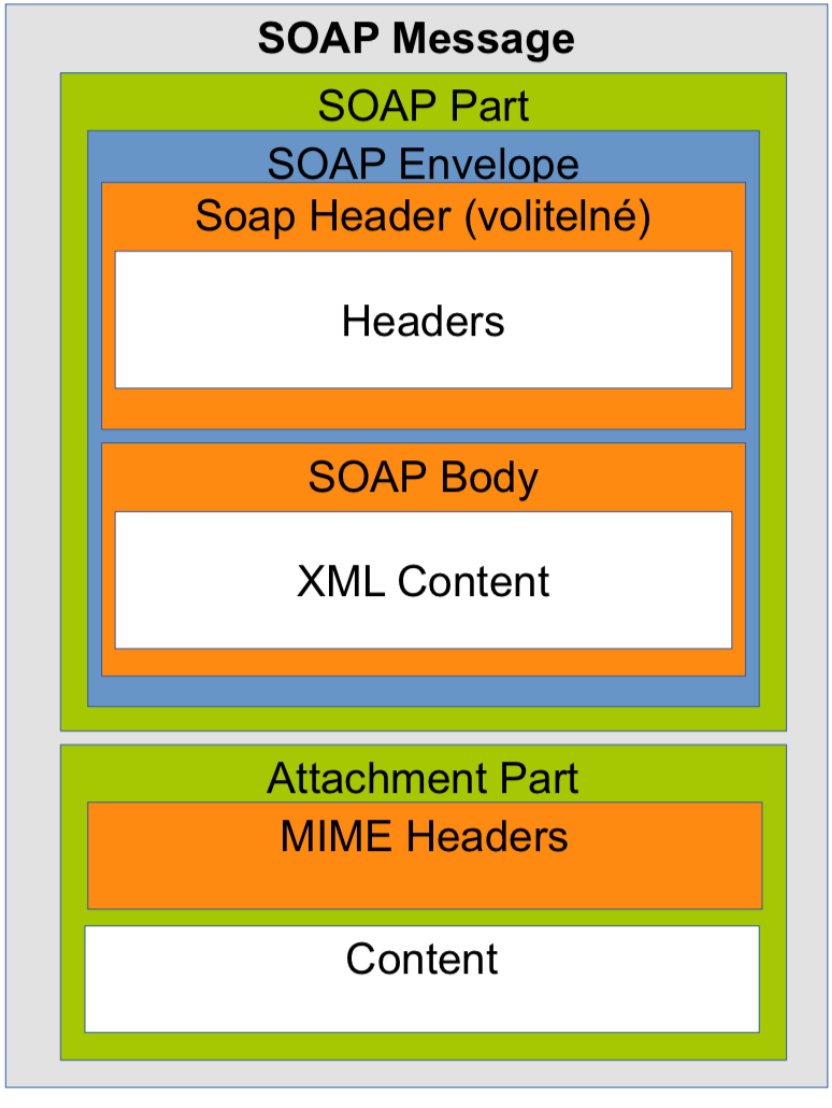
SOAP

- protokol definující pravidla
  - pro kódování přenášených dat
  - pro strukturu zprávy
  - napojení na transportní protokol
- postaveno na XML
- komunikace probíhá pomocí předávání zpráv

SOAP

- Pro popis a vystavení služeb se používá WSDL(Web Services Description language)
- Pro specifikace datových typů a struktury zpráv využívá XSD(Xml Scheme Definition)

SOAP - syntax



SOAP MESSAGE

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001
/12/soap-envelope"
soap:encodingStyle="http://www.w3.
org/2001/12/soap-encoding">

<soap:Header">
<age>26</age>
</soap:Header>

<soap:Body xmlns:m="namespace">
  <m:msgRq>
    <m:rqName>Hugo</m:rqName>
  </m:msgRq>
  <soap:Fault>
  </soap:Fault>
</soap:Body>
```

SOAP - syntax

- Envelope
  - identifikuje že jde o soap zprávu
  - root element každé soap zprávy
  - obsahuje namespace

SOAP - syntax

- Header
  - nepovinný element
  - pokud je použit, musí být prvním podelementem envelope
  - první element v hlavičce musí obsahovat namespace
  - obsahuje informace specifické pro aplikaci(zabezpečení, šifrování,...)

```
<soap:Header>
<m:Trans xmlns:m="http://www.w3schools.com/transaction/">123</m:Trans>
```

SOAP - syntax

- Body
  - povinný element
  - obsahuje předávané a vracené informace
  - první podelement může obsahovat namespace

SOAP - syntax

- Fault
  - návratový element, obsahuje chybové kódy a popis
    - <faultcode> - identifikace chyby
    - <faultstring> - popis chyby srozumitelný pro člověka
    - <faultactor> - původce chyby
    - <detail> - aplikační detail

SOAP - XSD

- Popis struktury XML dokumentu
  - atributy a elementy
  - obsah elementů
  - datové typy
  - fixní hodnoty

SOAP - XSD

```
<?xml version="1,0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLScheme"
targetNamespace="http://cizek.org"
xmlns="http://cizek.org"
elementFormDefault="qualified">

<xs:element name="note">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="from" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

SOAP - XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLScheme"
targetNamespace="http://cizek.org"
xmlns="http://cizek.org"
elementFormDefault="qualified">
```

- schema - kořek každého XSD
- xmlns:xs - definuje prefix 'xs' pro pro typy z namespace ../XMLScheme
- targetNamespace - typy definované v našem XSD budou patřit do namespace ../cizek.org
- xmlns - definuje defaultní (neprefixový) namespace
- elementFormDefault="qualified" - všechny námi lokální definované elementy budou patřit do target namespace

SOAP - XSD

simple type

```
<xs:element name="to" type="xs:string"/>
<xs:element name="from" type="xs:string"/>
```

- elementy typu string - obsahují pouze text, nemohou obsahovat jiný element nebo atribut
- nejpoužívanější typy - string, decimal, integer, boolean, date, time

SOAP - XSD

attribute

```
<xs:attribute name="id" type="xs:integer" use="required"/>
```

- definice stejná jako i simple type, ale jsou dostupné pouze pro komplexní typy
- atributy jsou defaultně nepovinné
- pro jeho vynucení můžeme použít 'use="required"'

SOAP - XSD

default a fixed hodnoty

- default - hodnot je přeřazena pokud není explicitně zadána jiná

```
<xs:element name="to" type="xs:string" default="cizek"/>
```

- fixed - hodnot je přeřazena vždy a nelze ji měnit

```
<xs:element name="to" type="xs:string" fixed="cizek"/>
```

SOAP - XSD

vlastní restrikce

```
<xs:element name="test">
    <xs:simpleType>
        <xs:restriction base="xs:integer">
.
.
.
```

- můžeme definovat vlastní restrikce
- musíme vždy definovat základní typ z kterého budeme vycházet. To lze pomocí atributu 'base'

SOAP - XSD

vlastní restrikce

- integer

```
<xs:minInclusive value="0"/>
<xs:maxInclusive value="1000"/>
```

- string
  - výčet hodnot

```
<xs:enumeration value="A"/>
<xs:enumeration value="B"/>
```

  - omezení regulárním výrazem

```
<xs:pattern value="[A-Z]*\s[0-9]{2,3}"/>
```

  - délka textu

```
<xs:length value="8"/>
<xs:minLength value="2"/>
```

SOAP - XSD

complexní typy

- element který obsahuje další elementy a atributy

```
<xs:element name="test">
    <xs:complexType>
        <xs:all>
            <xs:element name="A" type="xs:string"/>
            ...
        </xs:all>
    <xs:attribute name="id" type="xs:integer"/>
...
```

- order indicator - určuje pořadí a zastoupení elementů
  - all - každý element 1x
  - sequence - každý element 1x + musí jít za sebou
  - choice - vybereme právě jeden z elementů

SOAP - XSD

TEST

- vytvořte XSD pro element 'customer' obsahující:
  - firstname - jméno
  - lastname - příjmení
  - age - věk
  - id - číselný atribut (id např z DB)

SOAP - XSD

TEST

- vytvořte XSD pro element 'customer' obsahující:
  - firstname - jméno (3-20 znaků)
  - lastname - příjmení (3-20 znaků)
  - age - věk (0-110)
  - id - číselný atribut (id např z DB) (>0)

SOAP - XSD

TEST

```xml
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://cizek.org"
  targetNamespace="http://cizek.org">

  <element name="customer">
    <complexType>
      <sequence>
        <element ref="tns:firstname"/>
        <element ref="tns:lastname"/>
        <element ref="tns:age"/>
      </sequence>
      <attribute name="id" type="integer"
        use="required"/>
    </complexType>
  </element>

  <element name="lastname">
    <simpleType>
      <restriction base="string">
        <minLength value="3"/>
      </restriction>
    </simpleType>
  </element>
```

```xml
  <element name="firstname">
    <simpleType>
      <restriction base="string">
        <minLength value="3"/>
      </restriction>
    </simpleType>
  </element>

  <element name="age">
    <simpleType>
      <restriction base="integer">
        <minInclusive value="0"/>
        <maxInclusive value="110"/>
      </restriction>
    </simpleType>
  </element>

</schema>
```

SOAP - WSDL

- Web Service Description Language
- Popisuje webovou službu
- Postaveno na XML
- Může být generovaný nebo statický

SOAP – WSDL

- WSDL definuje
  - types - datové typy (XML schema)
  - message - definice příchozích a odchozích zpráv
  - portType - souhrn operací
  - operation - jednotlivé operace
  - binding - možnosti přístupu

SOAP - WSDL

```
<message name="Request">
 <part name="param1" element="number"/>
</message>
<message name="Response">
 <part name="result" element="resultElement"/>
</message>

<portType name="Numbers">
 <operation name="isOdd">
  <documentation>doc</documentation>
  <input message="tns:Request"/>
  <output message="tns:Response"/>
 </operation>
</portType>

<binding name="NumberService" type="Numbers">
 <SOAP:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="isOdd">
  <SOAP:operation style="rpc" soapAction=""/>
  <input>
   <SOAP:body use="literal"/>
  </input>
  <output>
   <SOAP:body use="literal"/>
  </output>
 </operation>
</binding>
```

```
<service name="Service">
 <documentation>number service</documentation>
 <port name="PrvniSluzba" binding="tns:PrvniSluzba">
  <SOAP:address location="http://localhost:10000"/>
 </port>
</service>
```

SOAP – WSDL

scénáře přenosu zpráv

- One - Way
  - klient pošle request a nechce response
- Request - Response
  - klient pošle request a čeká na response
- Solicit - Response
  - služba(server) žádá o volání a klient ji potom volá
- Notification
  - služba(server) předá klientovi informace

SOAP - WSDL

scénáře přenosu zpráv

- Scénář se definuje ve WSDL v části 'portType'. Scénář je odvozen z pořadí zpráv u operace

```
<portType name="helloPortType">
 <operation name="sayHello">
  <input name="in" message="msgRq"/>
 </operation>
</portType>
```

One-Way

```
<portType name="helloPortType">
 <operation name="sayHello">
  <input name="in" message="msgRq"/>
  <output name="out" message="msgRs"/>
 </operation>
</portType>
```

Request-Response

Solicit-Response

```
<portType name="helloPortType">
 <operation name="sayHello">
  <output name="out" message="msgRs"/>
  <input name="in" message="msgRq"/>
 </operation>
</portType>
```

Notification

```
<portType name="helloPortType">
 <operation name="sayHello">
  <output name="out" message="msgRs"/>
 </operation>
</portType>
```

SOAP - WSDL

Styly SOAP zpráv

- definují jak vypadá tělo(body element) zprávy
- definuje se v části 'binding'

```
<soap:binding transport="http://.../http"
 style="document"/>
```

- 'Document' - v body je XML dokument podle XSD
- 'RPC' - v body je jméno vzdálené procedury a její parametry

SOAP - WSDL

Kódování SOAP zpráv

- Definován pro input a output každé operace

```
<soap:operation soapaction="http..."> 
 <wsdl:input><soap:body use="literal"/>...
 <wsdl:output><soap:body use="literal"/>...
```

- literal - formát je dán XSD
- encoded - formát je dán SOAP kódováním

SOAP - WSDL

Kódování SOAP zpráv

RPC/encoded

```
<soap:envelope>
    <soap:body>
        <myMethod>
            <x xsi:type="xsd:int">5</x>
        </myMethod>
    </soap:body>
</soap:envelope>
```

RPC/literal

```
<soap:envelope>
    <soap:body>
        <myMethod>
            <x>5</x>
        </myMethod>
    </soap:body>
</soap:envelope>
```

Document/literal

```
<soap:envelope>
    <soap:body>
        <x>5</x>
    </soap:body>
</soap:envelope>
```

SOAP – WSDL

Kódování SOAP zpráv

- Co kdy zvolit a proč?
  - většinou document/litaral
  - RPC se hodí ve speciálních případech

SOAP - WSDL

Kódování SOAP zpráv

- Máme metody:

```
public void method1(int x, float y)
public void method2(int x)
public void method3(int x, float y)
```

SOAP - WSDL

Kódování SOAP zpráv

- Máme metody:

```
public void method1(int x, float y)
public void method2(int x)
public void method3(int x, float y)
```

- Při kombinace document/literal nebudeme schopni rozeznat metody 1 a 3 protože mají stejné parametry

SOAP - WSDL

Kódování SOAP zpráv

- Máme metody:

```
public void method1(int x, float y)
public void method2(int x)
public void method3(int x, float y)
```

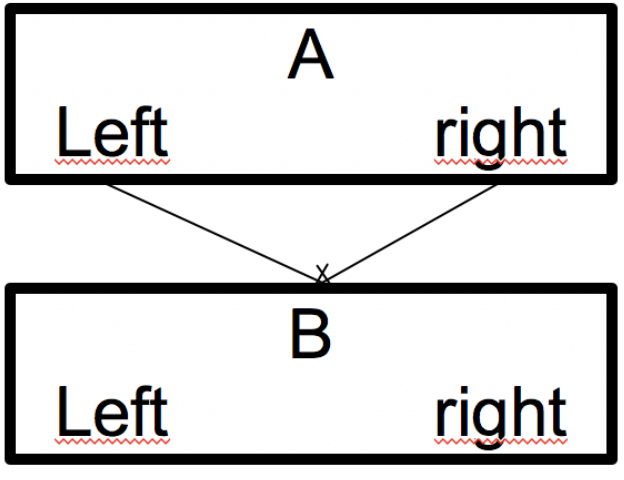
- Při kombinace document/literal nebudeme schopni rozeznat metody 1 a 3 protože mají stejné parametry
- RPC/literal zajistí že pošleme i jméno

SOAP - WSDL

Kódování SOAP zpráv

- Přenos datových grafů
- RPC/encoded má attribute href

```
<A>
    <name>A</name>
    <left href="123"/>
    <right href="123"/>
</A>
<B id="123">
    <name>B</name>
    <left xsi:nil="true"/>
    <right xsi:nil="true"/>
</B>
```

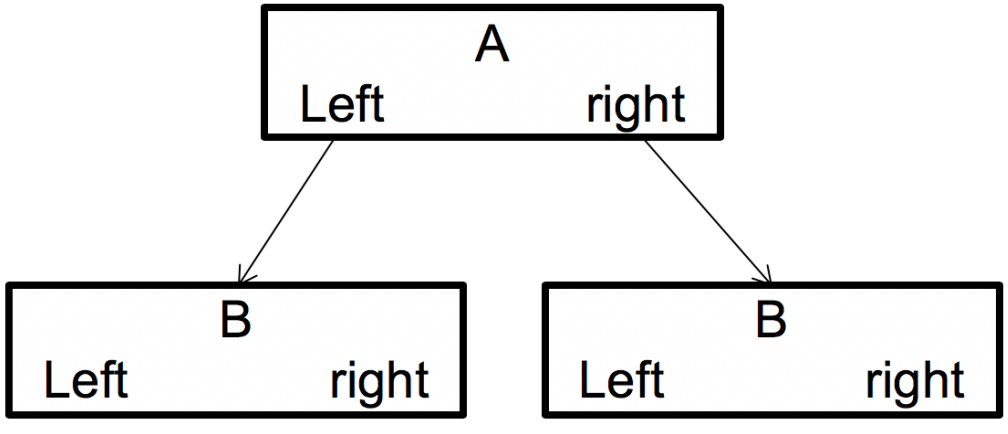


SOAP - WSDL

Kódování SOAP zpráv

- bez atributu href

```
<A>
    <name>A</name>
    <left>
        <name>B</name>
        <left xsi:nil="true"/>
        <right xsi:nil="true"/>
    </left>
    <right>
        <name>B</name>
        <left xsi:nil="true"/>
        <right xsi:nil="true"/>
    </right>
</A>
```



SOAP - Prakticky

- git checkout
- import projet JAX_WS_SERVER_CF

- vytvořme jednoduchou WS
  - vstup: jméno
  - výstup: hello 'jméno'

SOAP - Prakticky

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:tns="http://ws.cizek.cz"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://ws.cizek.cz">

    <!-- Definujeme elementy -->
    <xs:element name="sayHelloRequest" type="tns:sayHelloRequest"/>
    <xs:element name="sayHelloResponse" type="tns:sayHelloResponse"/>

    <!-- Definici komplexních typů -->
    <xs:complexType name="sayHelloRequest">
        <xs:sequence>
            <xs:element name="name" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="sayHelloResponse">
        <xs:sequence>
            <xs:element name="greeting" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

SOAP - Prakticky

```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:tns="http://ws.cizek.cz"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             targetNamespace="http://ws.cizek.cz">

    <types>
        <xsd:schema>
            <xsd:import namespace="http://ws.cizek.cz" schemaLocation="hello.xsd"/>
        </xsd:schema>
    </types>

    <message name="sayHelloRequest">
        <part name="parameters" element="tns:sayHelloRequest"/>
    </message>

    <message name="sayHelloResponse">
        <part name="parameters" element="tns:sayHelloResponse"/>
    </message>

    <portType name="sayHelloPT">
        <operation name="sayHello">
            <input message="tns:sayHelloRequest"/>
            <output message="tns:sayHelloResponse"/>
        </operation>
    </portType>

    <binding name="sayHelloPortBinding" type="tns:sayHelloPT">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
        <operation name="sayHello">
            <input>
                <soap:body parts="parameters" use="literal"/>
            </input>
            <output>
                <soap:body use="literal"/>
            </output>
        </operation>
    </binding>

    <service name="sayHelloService">
        <port name="sayHelloPort" binding="tns:sayHelloPortBinding">
            <soap:address location="http://localhost:8888/ws/hello"/>
        </port>
    </service>

</definitions>
```

SOAP - Prakticky

- build nyní vytvoří interface

- implementací interface vytvoříme endpoint

```
@WebService
public class HelloWorld implements SayHelloPT {

    @Override
    public SayHelloResponse sayHello(SayHelloRequest parameters) {

        SayHelloResponse sayHelloResponse = new SayHelloResponse();

        sayHelloResponse.setGreeting("AHOJ");

        return sayHelloResponse;
    }
}
```

SOAP - Prakticky

- endpoint je třeba poposat do descriptoru

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoints xmlns='http://java.sun.com/xml/ns/jax-ws/ri/runtime' version='2.0'>

    <endpoint name='HelloWorld-endpoint'
              implementation='cz.cizek.ws.HelloWorld'
              url-pattern='/hi' />

</endpoints>
```

- po startu serveru ../soap/hi

SOAP - Prakticky

- klient
  - v maven pluginu wsimport je cesta kde maji byt wsdl/xsd
  - zkopirovat vystavene wsdl a xsd ze serveru
  - build pro vygenerovani objektu

```
URL url = new URL("http://localhost:8888/soap/hi?wsdl");
QName qName = new QName("http://ws.cizek.cz/", "HelloWorldService");

Service service = Service.create(url, qName);
SayHelloPT sayHelloPT = service.getPort(SayHelloPT.class);

SayHelloRequest request = new SayHelloRequest();
request.setName("jikra");

SayHelloResponse response = sayHelloPT.sayHello(request);

System.out.println(response.getGreeting());
```

SOAP - Prakticky

- server + client spring boot CL

REST

- Bezstavová komunikace
  - na straně serveru nejsou držena žádná klientská data jako session atp.
  - Pokud aplikace něco potřebuje(autorizace atd..) musí se tato data vždy poslat

REST

- Adresovatelnost
  - Každý zdroj je dosžitelný pomocí standardizovaného URI

    schema://host:port/path?queryString#fragment

  - schema - protokol (http)
  - host - DNS/IP poskytovatele
  - port - je port ;)
  - path - virtuální hierarchická struktura
  - queryString - množina párů 'klíč=hodnota'
  - fragment - specifikuje část zdroje

REST

- konvence path
  - používáme podstatná jména v množném čísle
    - ../pets - vrátí seznam mazlíčků
    - ../pets/12 - vrátí konkrétního mazlíčka
  - slovesa jen když to jinak nejde
    - convert?from=CZK&to=USD&amount=100

REST

- jakou zvolíme URI pro seznam mazlíčků daného majitele
  - majitel je identifikován celočíselným ID
  - HINT
    - schema://host:port/path?queryString

REST

- jakou zvolíme URI pro seznam mazlíčků daného majitele
  - majitel je identifikován celočíselným ID

../pets?owner=111

  - HINT
    - schema://host:port/path?queryString

REST

- HATEOAS - Hypermedia As The Engine Of Application State
  - známe pouze jednu(vstupní) URL a ten nám vrátí vše co potřebujeme

```
<product id=42>
    <..href="http://..../product/42"/>
    <name>apple</name>
    <price>1</price>
</product>
```

REST

- jednotné rozhraní - http metody mají specifický účel
  - GET - načtení zdrojů, idempotentní, bezpečná
  - PUT - vložení nebo přepsání dat, idempotentní
  - DELETE - odstraní zdroj, idempotentní
  - POST - aktualizace dat, **není idempotentní**
  - HEAD - jako GET ale vrací jen kód a hlavičku
  - OPTIONS - vrací komunikační možnosti serveru

REST

- reprezentace zdrojů
  - při komunikace dochází k výměně zdrojů, resp. jejich reprezentace

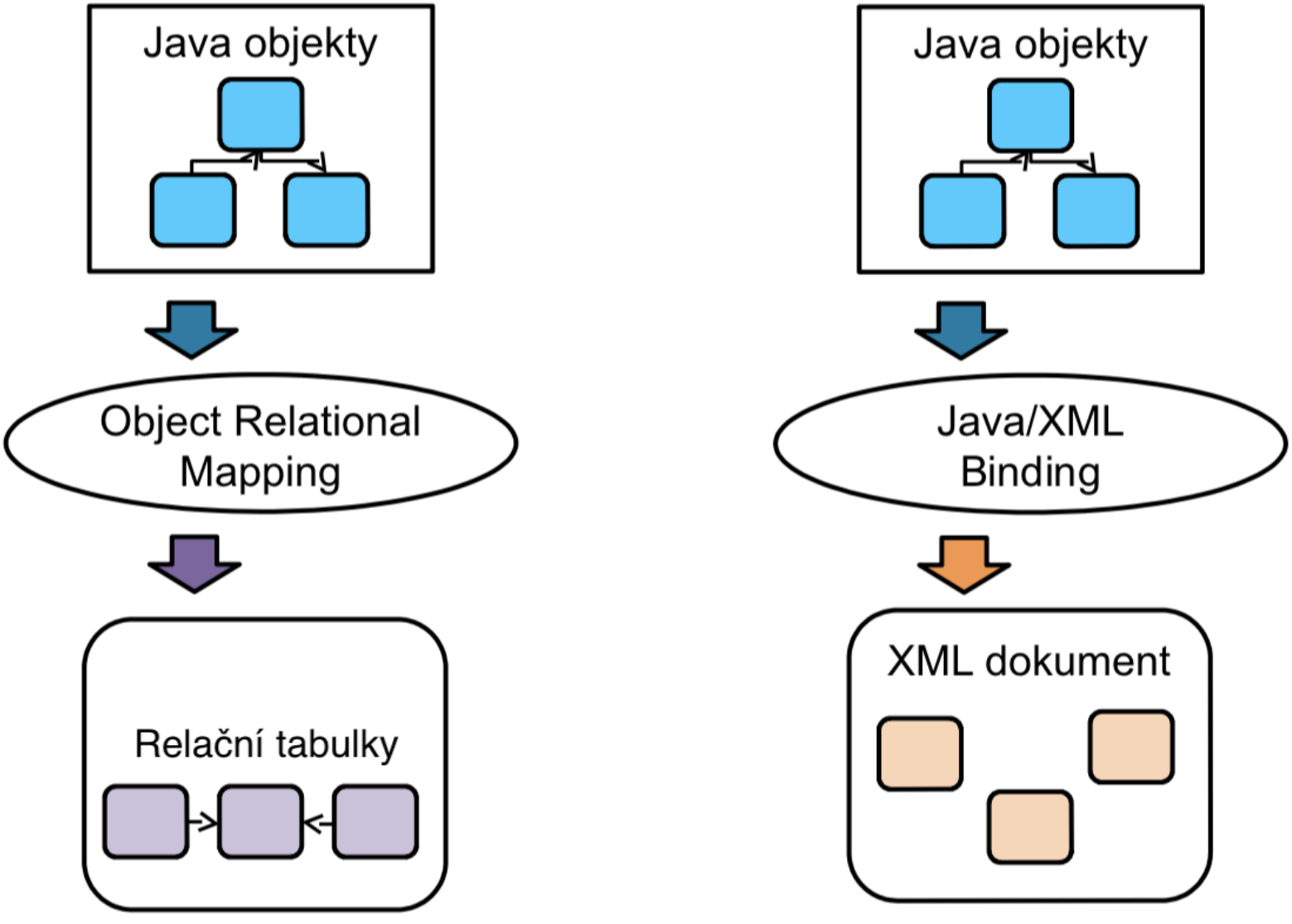
```
<product id=42>
    <name>apple</name>
    <price>1</price>
</product>
```

  - je tato reprezentace dostatečná?

REST

- jak reprezentace zdroje vznikne
  - procesem marshalizace
  - mapování Java objektů do textového formátu a zpět

REST


Java objekty
Object Relational Mapping
Relační tabulky
Java objekty
Java/XML Binding
XML dokument

REST

- pro mapování slouží JAXB - Java Architecture for XML Binding
- Entita (mapovaný objekt) musí obsahovat anotaci @XmlRootElement
- Atributy entity které chceme mapovat jsou anotovány @XmlElement nebo @XmlAttribute

REST

- Mapovaná entita

```
@XmlRootElement(name=“per”)
public class Person {

    @XmlElement(name=“data”)
    private Name name;

    @XmlAttribute
    private int id;

    //... }


public class Name {

    private String name;
    private String surname;

     //...
}
```

REST

- Vystavená služba

```
@Path(“hello”)
public class HelloResource {

    @GET
    @Produces(“text/plain”)
    public String sayHello() {
        return “Hello”;
    }

    @POST
    @Path(“{name}”)
    public String sayHello(@PathParam(“name”)
                           @DefaultValue(“Hugo”) String name) {
        return “Hello, ” + name;
    }
}
```

REST - prakticky

Co budeme potřebovat:

- git
- maven
- embeded tomcat
- spring boot
- java 8

REST - prakticky

- git checkout : git clone https://github.com/jikra/ws.git
- import projektu do IDE (idea)

- test projektu
  - clean install
  - tomcat:run

REST - prakticky

JAX_RS-server

- vytvoříme helloWorld controller
- přídáme servelt a mapování ve web.xml

```
@Path("/hello")
public class HelloWorldController {

    @GET
    @Path("/helloWorld")
    public Response sayHello() {

        return Response.status(Response.Status.OK)
            .entity("Hello World").build();
    }
}
```

REST - prakticky

JAX_RS-server

- vytvoříme helloWorld controller
- přídáme servelt a mapování ve web.xml

```
    <servlet>
        <servlet-name>jersey-serlvet</servlet-name>
        <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-c
        <init-param>
            <!-- balíčky v kterých se budou hledat služby -->
            <param-name>com.sun.jersey.config.property.packages</param-name>
            <param-value>cz.cizek.rest.controller</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>jersey-serlvet</servlet-name>
        <url-pattern>/api/*</url-pattern>
    </servlet-mapping>
```

REST - prakticky

JAX_RS server - query magic

- path
- query
- context query
- matrix
- form

Header

- param
- context

REST - prakticky

JAX_RS server - soubory

- soubory
  - download
  - upload

REST - prakticky

JAX_RS server - objekty

- objekty
  - get
  - add - s klientem

REST - prakticky

JAX_RS client

- pustit test
- objekty
  - ziskani objektu
  - unmarshalizace
  - odeslani

REST - prakticky

SPRING rest server

- run test
- hello

REST - prakticky

SPRING rest client

- resttemplate
- commandlinerunner pro vykonani testu
- zavolani rest serveru a ziskani objektu

REST - prakticky

Testování služeb SoapUI

- call
- test
- mock

REST - prakticky

Base validace

- requestBody
- base validace
  - @validate - 400 bad request
  - exception handler
  - resttemplate a errorhandler