Lecture 6: Web Services & REST APIs

Olivier Liechti
AMT

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Agenda

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

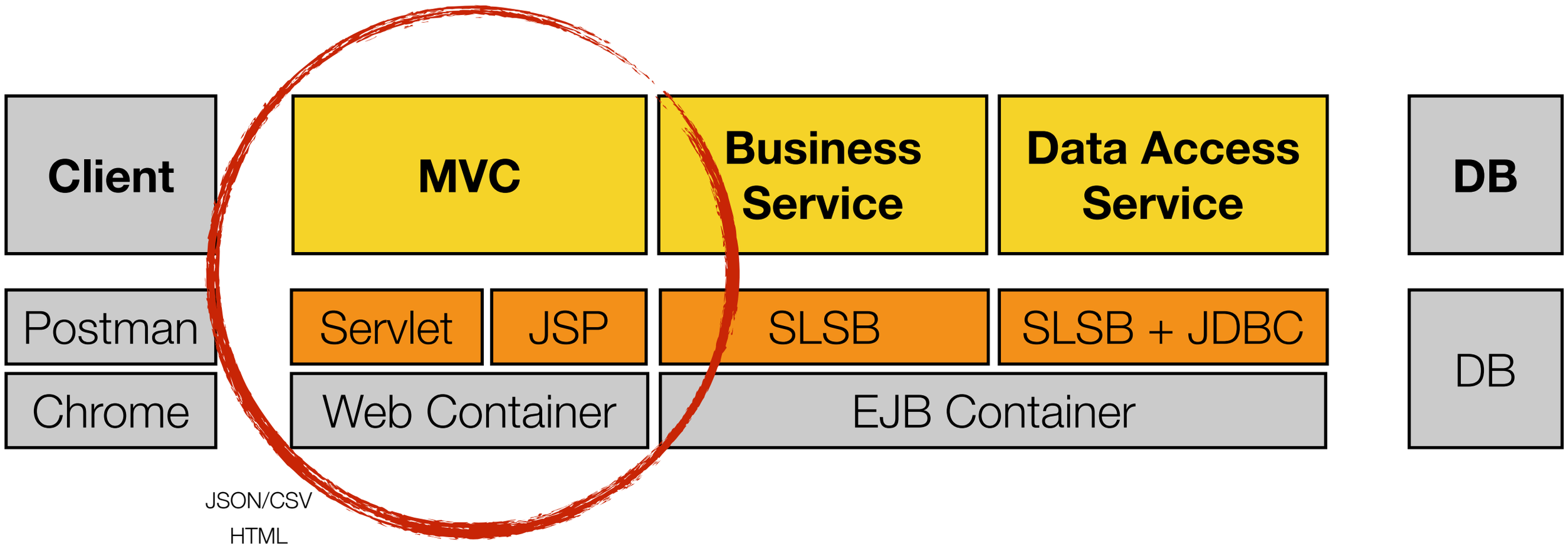
13h00 - 13h30	30’	**Lecture** Introduction: services, remote service invocation, web services architecture, service registry
13h30 - 14h00	30’	**Lecture/Demo** “Big” web services with Java EE
14h00 - 15h00	60’	**Lab** Implement and experiment.
15h00 - 15h30	*30’*	***Break***
15h30 - 16h00	30’	**Lecture** REST APIs
16h00 - 16h30	30’	**Demo** REST APIs with Java EE
16h30 - 18h00	90’	**Lab** Time to experiment with JAX-RS

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

What do we know about the presentation tier?

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

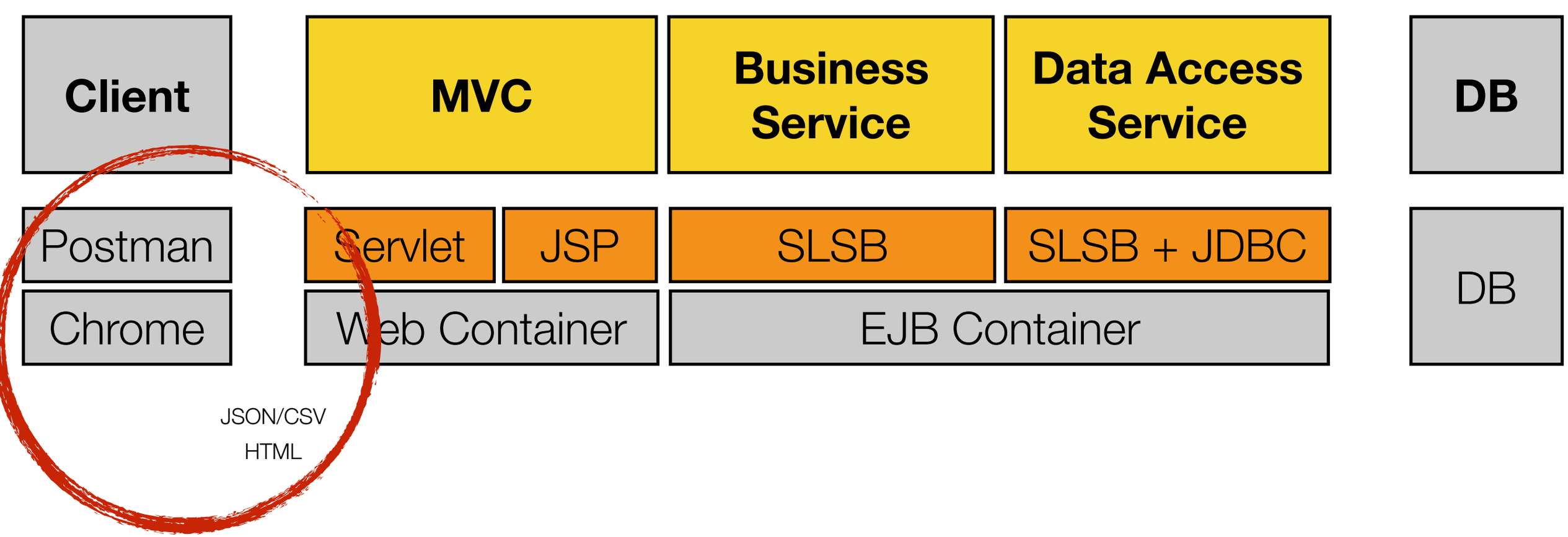
Until now, the presentation tier was all about MVC, servlets and JSPs...



heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

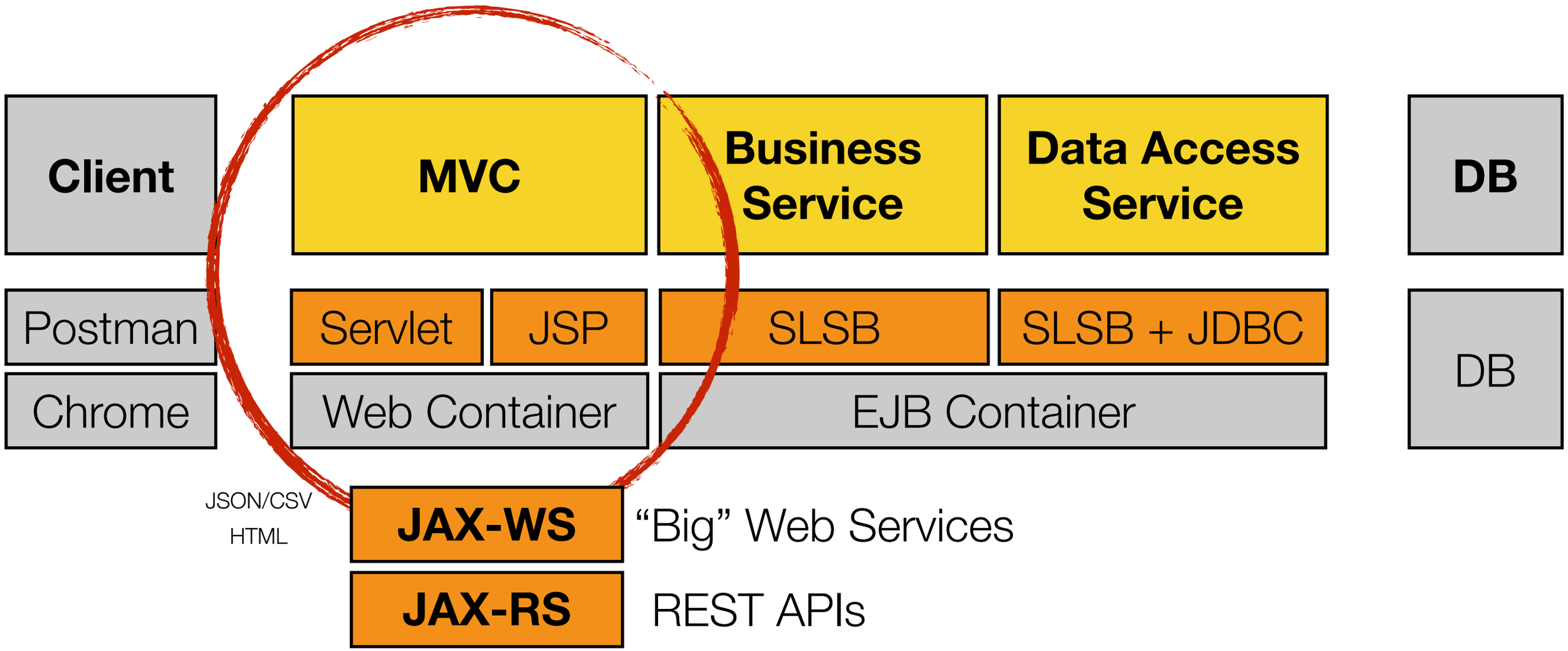
... we were mostly thinking about a Web UI (used by a human).

We hinted about the ability to send/receive machine-understandable data (used by software agents)



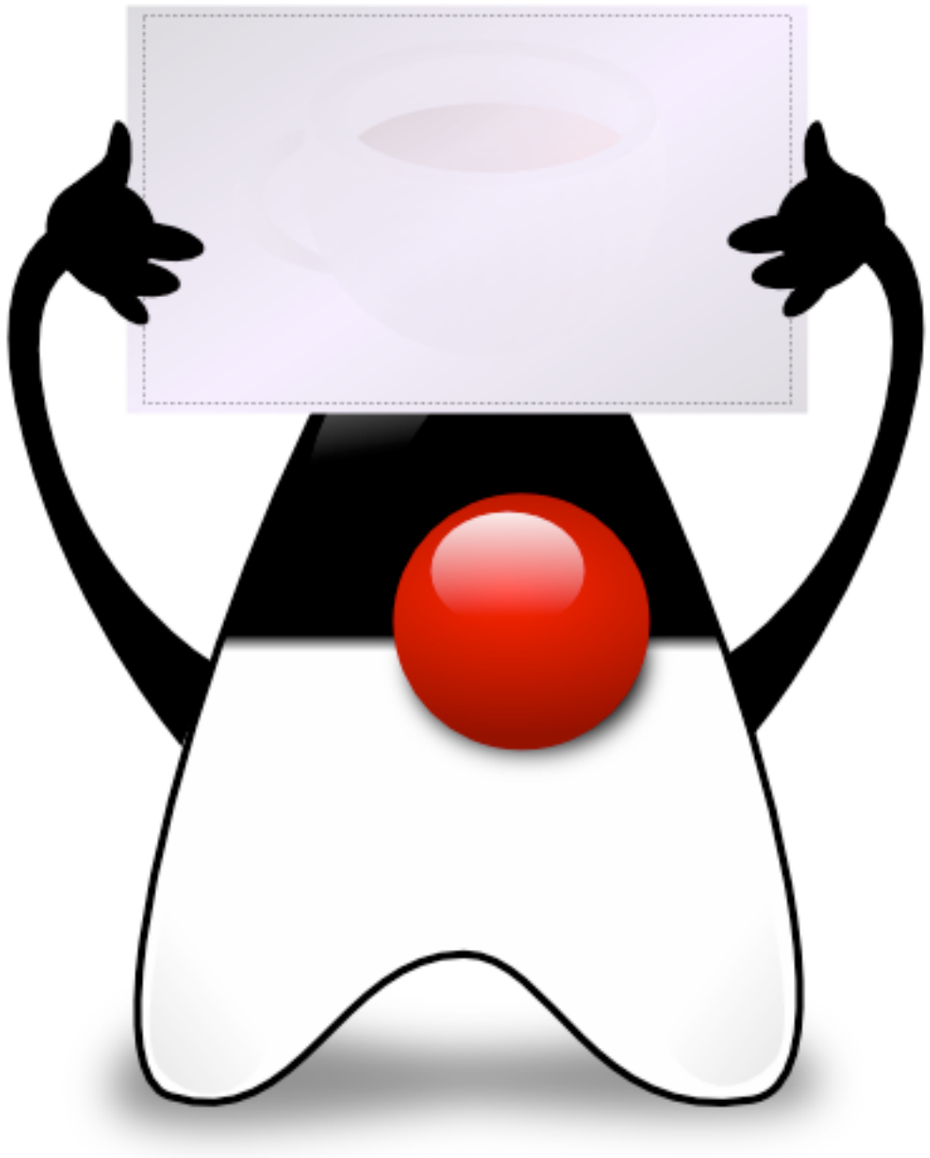
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Today, we will look at APIs that complement servlets and JSPs and that focus on software clients. We will talk about web services.



heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

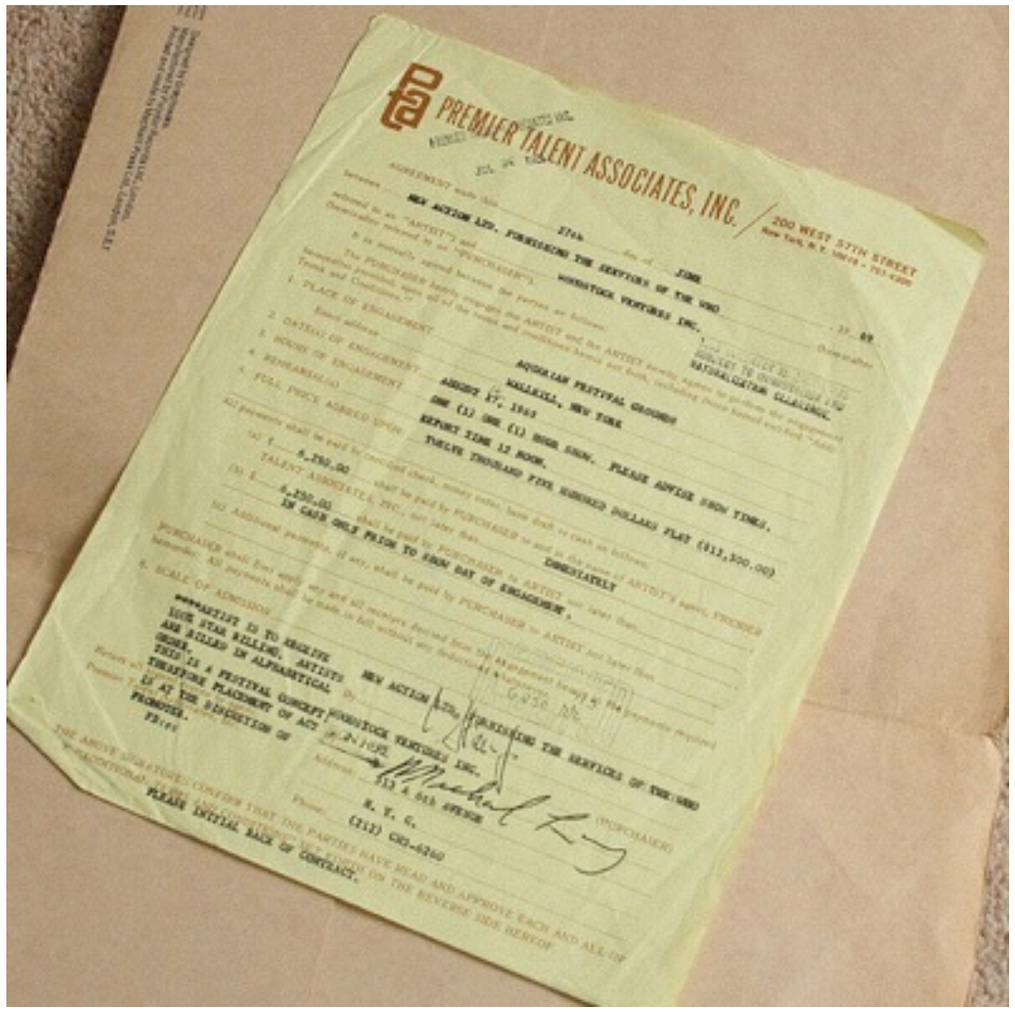


The concept of “Service”

The Concept of Service

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- The abstraction of service is **generic**:
  - It is key in the design of many systems.
  - “Small” systems (applications)
  - “Big” systems (information systems)
  - “Distributed” systems
- But what is a “**service**”?
  - It is “something” that provides access to some functionality.
  - The functionality should be described in an interface, which defines a contract between the client and the service provider.

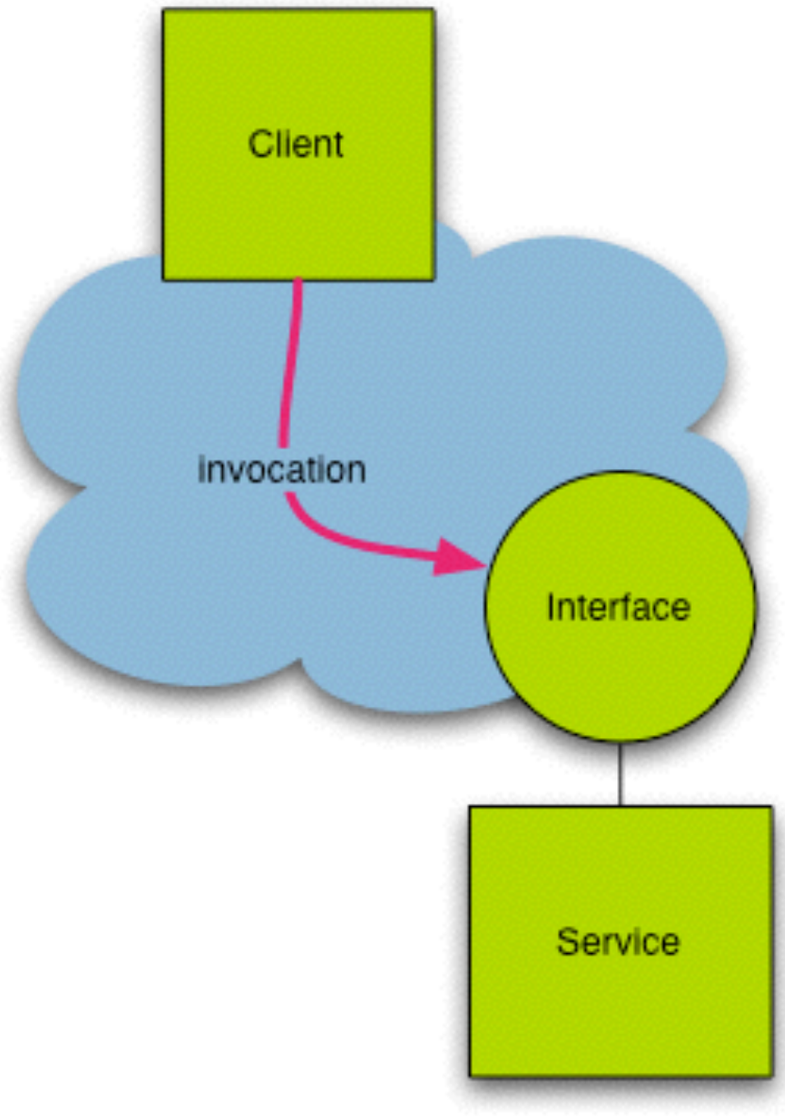


http://flickr.com/photos/ian-s/2655970731/sizes/l/#cc_license

How Can I Implement a Service?

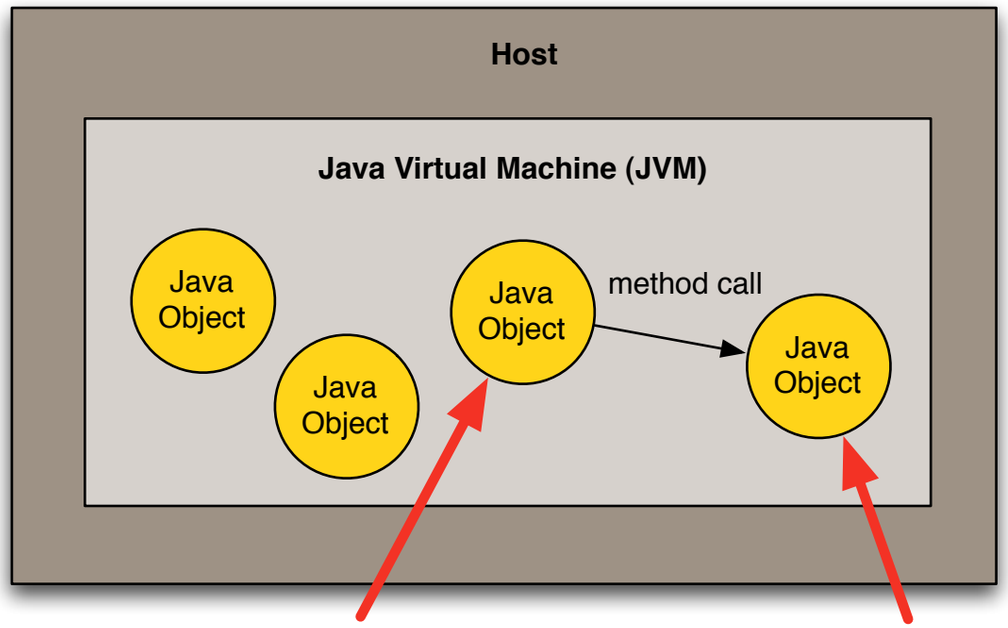
heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- **Lots of different technologies** can be used to implement services.
- Just think about a basic **Java application**:
  - A class that implements an interface is an example of a service!
  - In that case, service providers and clients live in the same VM - service invocation ha
- Now, think about **distributed Java** applications:
  - RMI makes it possible to invoke a service running in a different JVM.
  - This is an example of Remove Procedure Call.



A Service inside a Single JVM

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



The Client

```
public class ClockClient

  private ClockService service;

...

  public String showTime() {
    System.out.println("It is " +
      service.getTime());
  }
}
```

ClockClient.java

The Service

```
public interface ClockService {

  public String getTime();
}
```

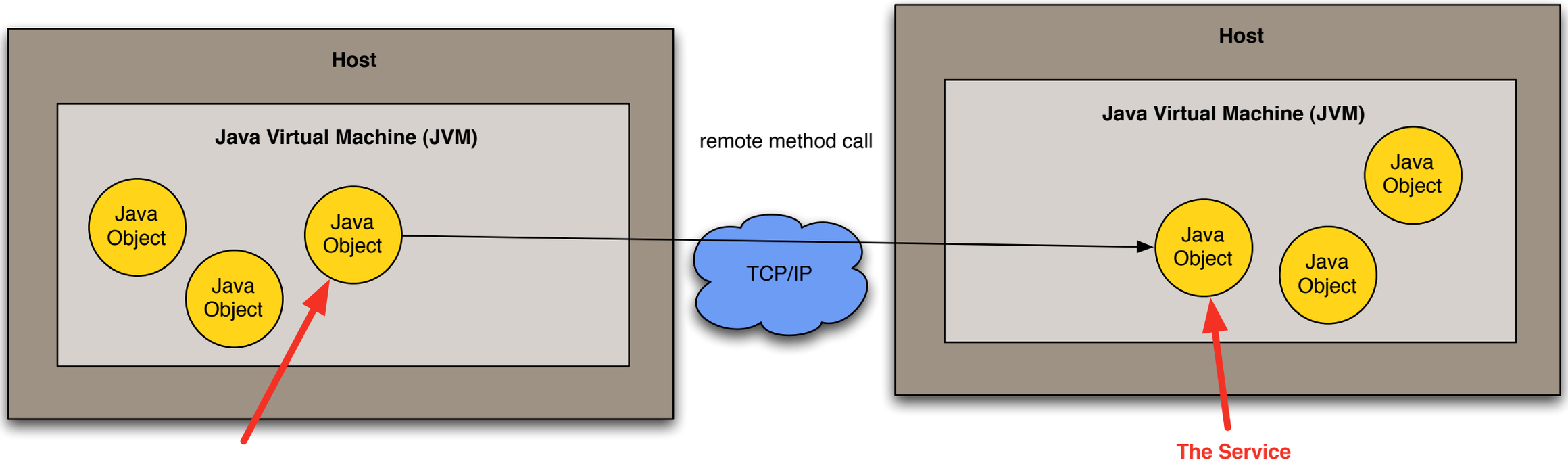
ClockService.java

```
public class ClockServiceImpl
  implements ClockService {

  public String getTime() {
    return new java.util.Date();
  }
}
```

ClockServiceImpl.java

A Service in a Remote JVM

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud




```
public class ClockClient

}
```

générer du XML (SOAP) ?

ouvrir un socket?

parser du XML (réponse SOAP)

?

ClockClient.java

```
public interface ClockService {

  public String getTime();
}
```

ClockService.java

```
public class ClockServiceImp
  implements ClockService {

}
```

générer du XML (SOAP) ?

écouter sur un socket?

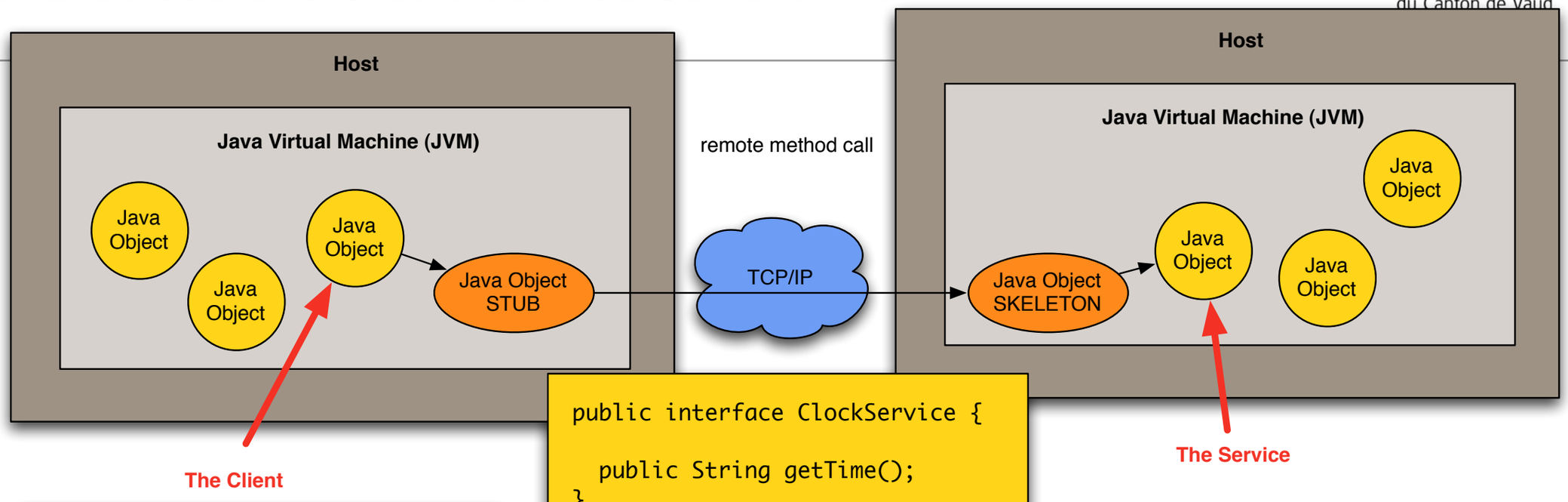
parser du XML (SOAP)

?

ClockServiceImpl.java

A Service in a Remote JVM

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud




```
public interface ClockService {

  public String getTime();
}
```

ClockService.java

```
public class ClockClient

  private ClockService service;

...

  public String showTime() {
    System.out.println("It is " +
      service.getTime());
  }
}
```

ClockClient.java

```
public class ClockServiceStub
  implements ClockService {

  public String getTime() {
    ... bla bla... SOCKET ...
    XML... bla bla bla... SOAP ...
    bla bla OUTPUTSTREAM... bla bla
    INPUTSTREAM....

    return time;
}
```

ClockServiceStub.java

```
public class ClockServiceSkeleton {

  private ClockService realService;

  private void setup() { ... }
  private void listen() { ... }
  private void processCall {
    realService.getTime();
  }
}
```

ClockServiceSkeleton.java

```
public class ClockServiceImpl
  implements ClockService {

  public String getTime() {
    return new java.util.Date();
  }
}
```

ClockServiceImpl.java

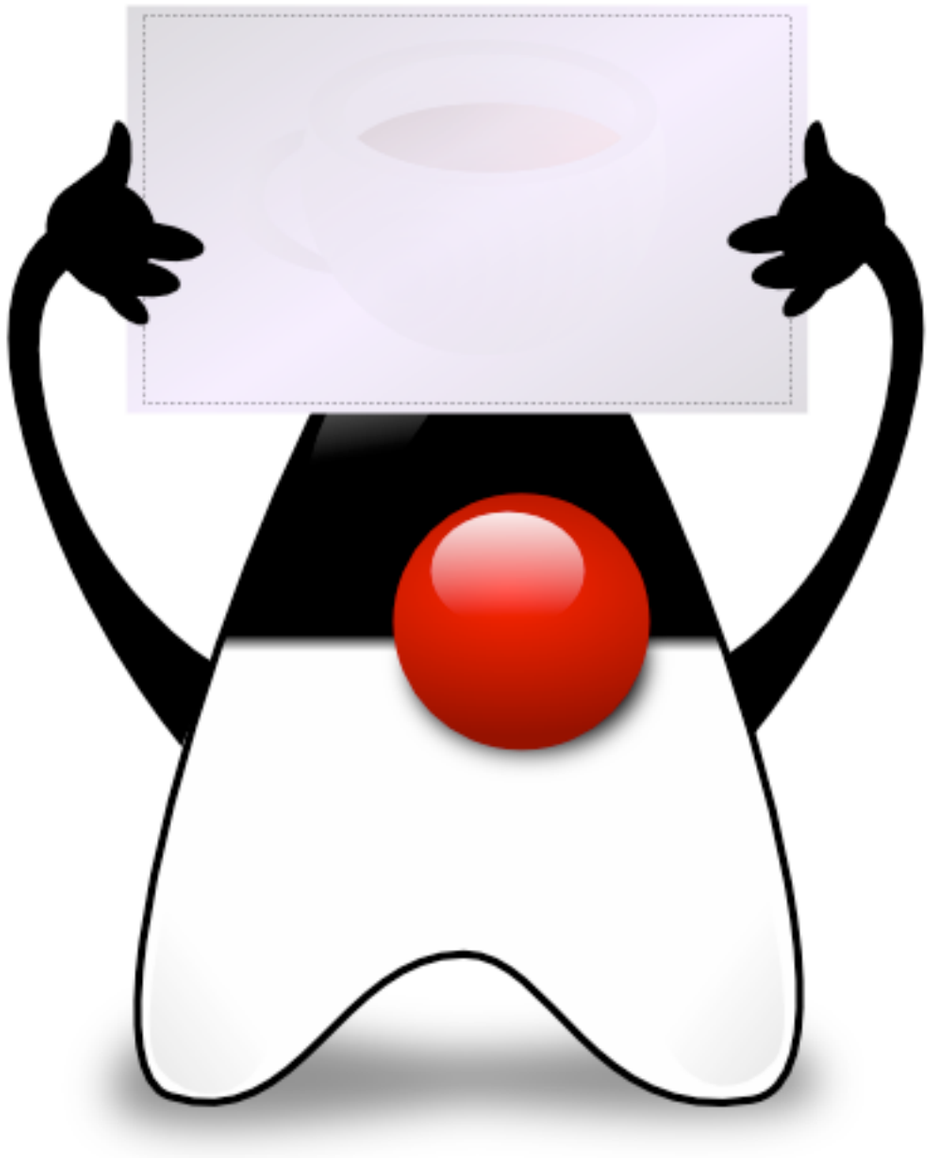
Generated automatically by special tools!

Stubs and Skeletons

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

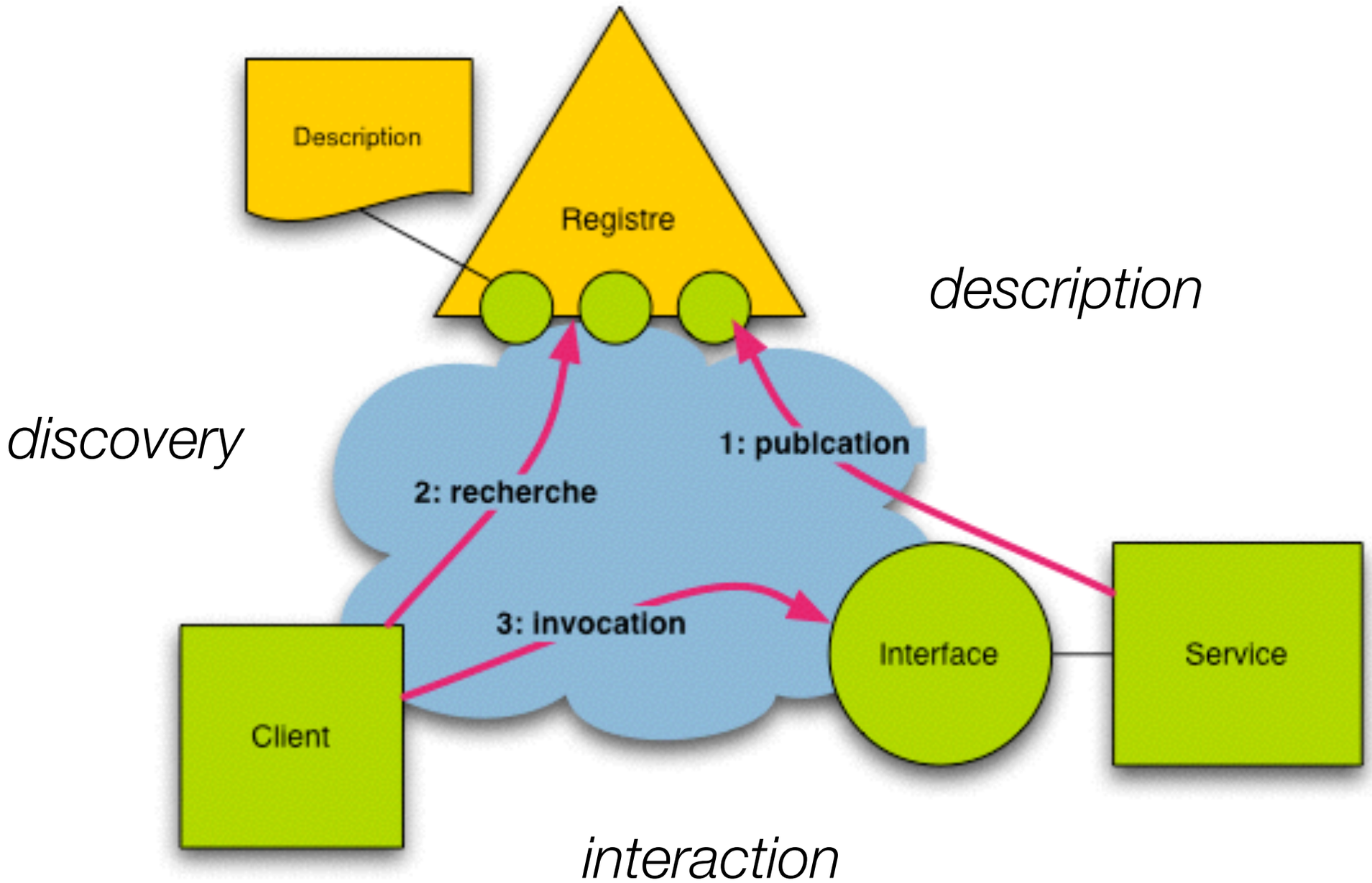
- In distributed computing, a "**stub**" is a software component that acts as a **proxy** and provides a way for a client to interact with a remote service. It is typically **generated** by the distributed computing infrastructure.
  - The stub exposes the service interface to the client, locally.
  - The stub implements this interface by handling the data transformation and remote communication.
    - The client can make local method invocations on the stub and have the impression that he is directly interacting with the remote service.
- A "**skeleton**" is a similar component, also acting as a proxy, but on the server side.
  - The skeleton receives remote invocations from the stub. At this point, it handles data transformation and forwards the invocation to the target service.

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



The “Web Services” Architecture

The Web Services Reference Architecture



description

discovery

interaction

The Concept of Service Registry

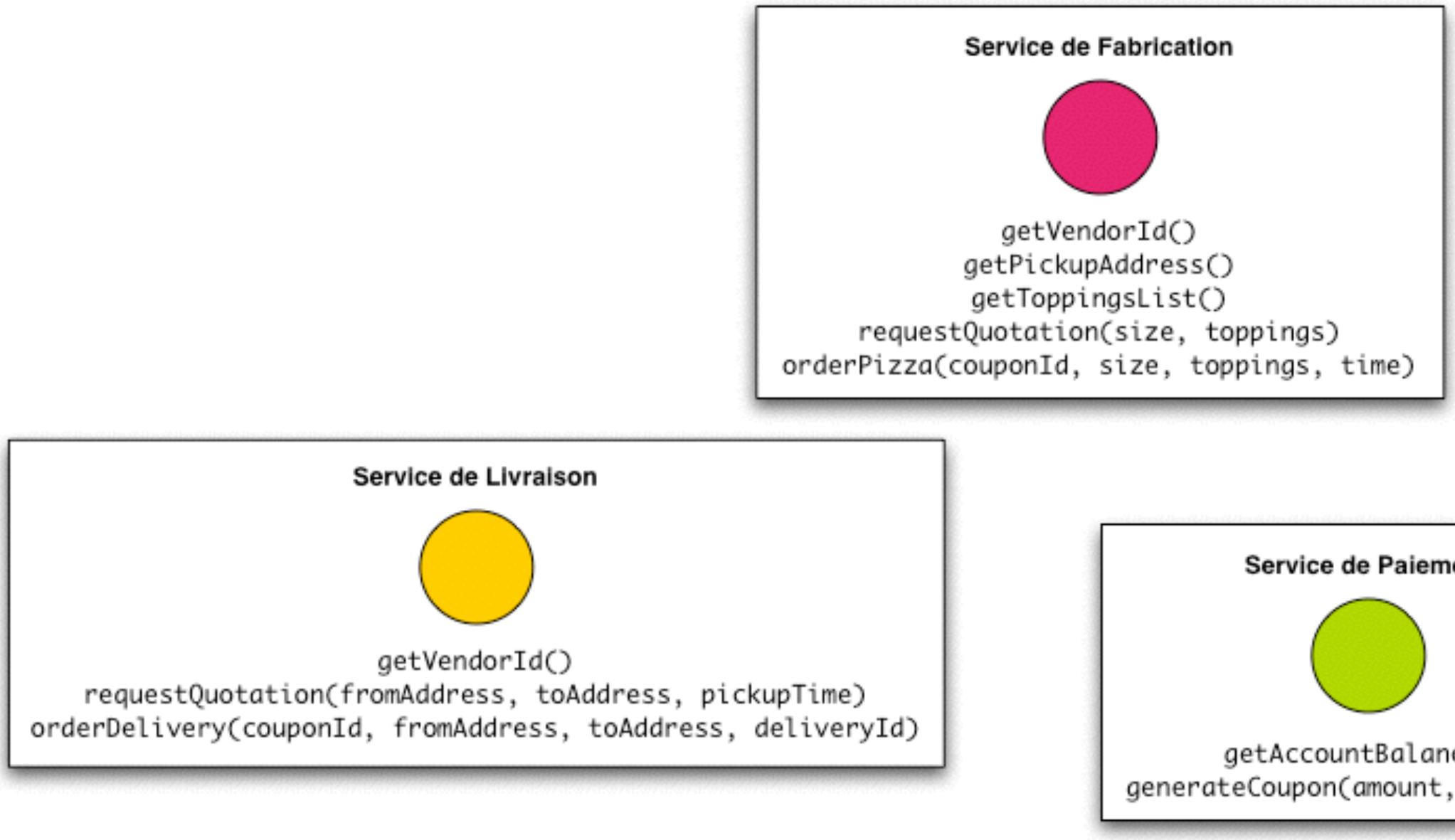
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- **Service Oriented Architectures** go beyond simple service invocation:
  - Both in intra-, extra- and internet scenarios, the number of services is going to increase significantly.
  - More and more, building an application means reusing and combining these services.
  - Whether you call it a “mashup” or a “business process”, it’s the same idea!
- Example:
  - To eat a pizza, I need 1) a “pizza preparation” service, 2) a delivery service, 3) a payment service.
  - Different companies may provide these services (and not necessarily 24/7).
  - Questions: how to identify service providers? how to select service providers?

http://flickr.com/photos/qchristopher/1822166721/sizes/m/#cc_license

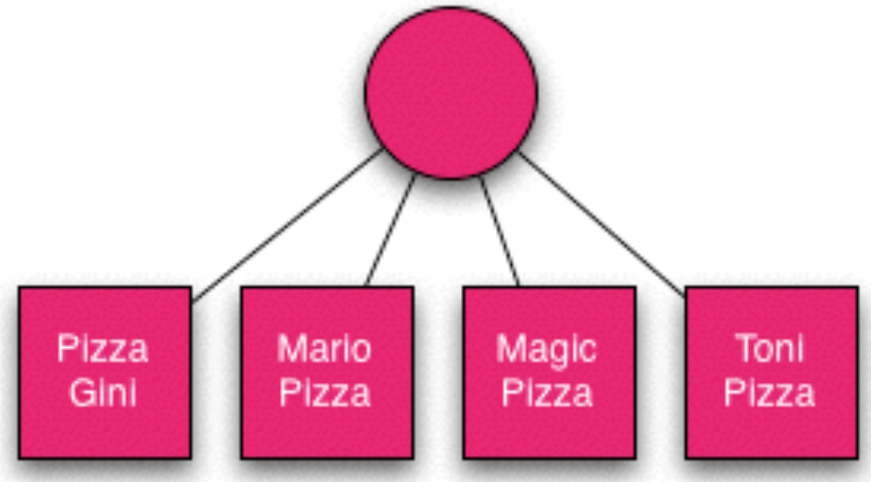
Example: Service Interfaces

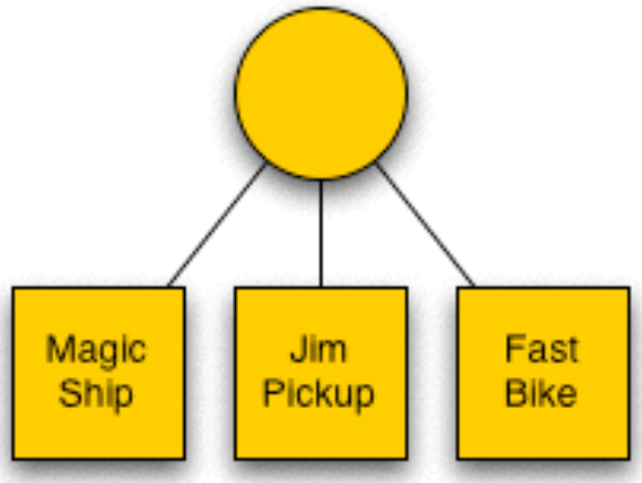
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

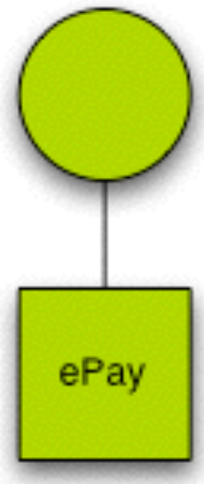


Example: Several Providers for a Service

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

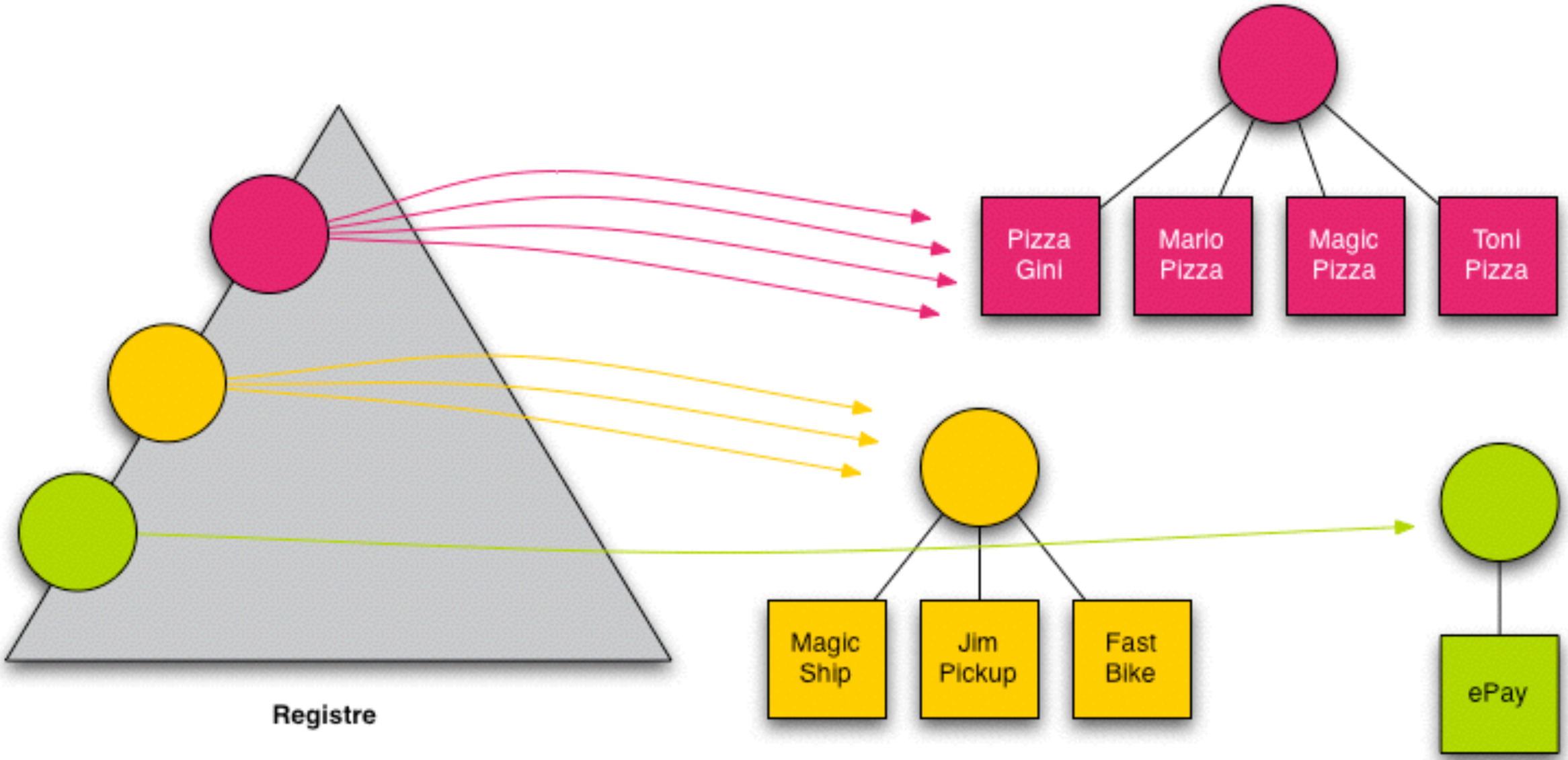






Example: Using a Service "Registry"

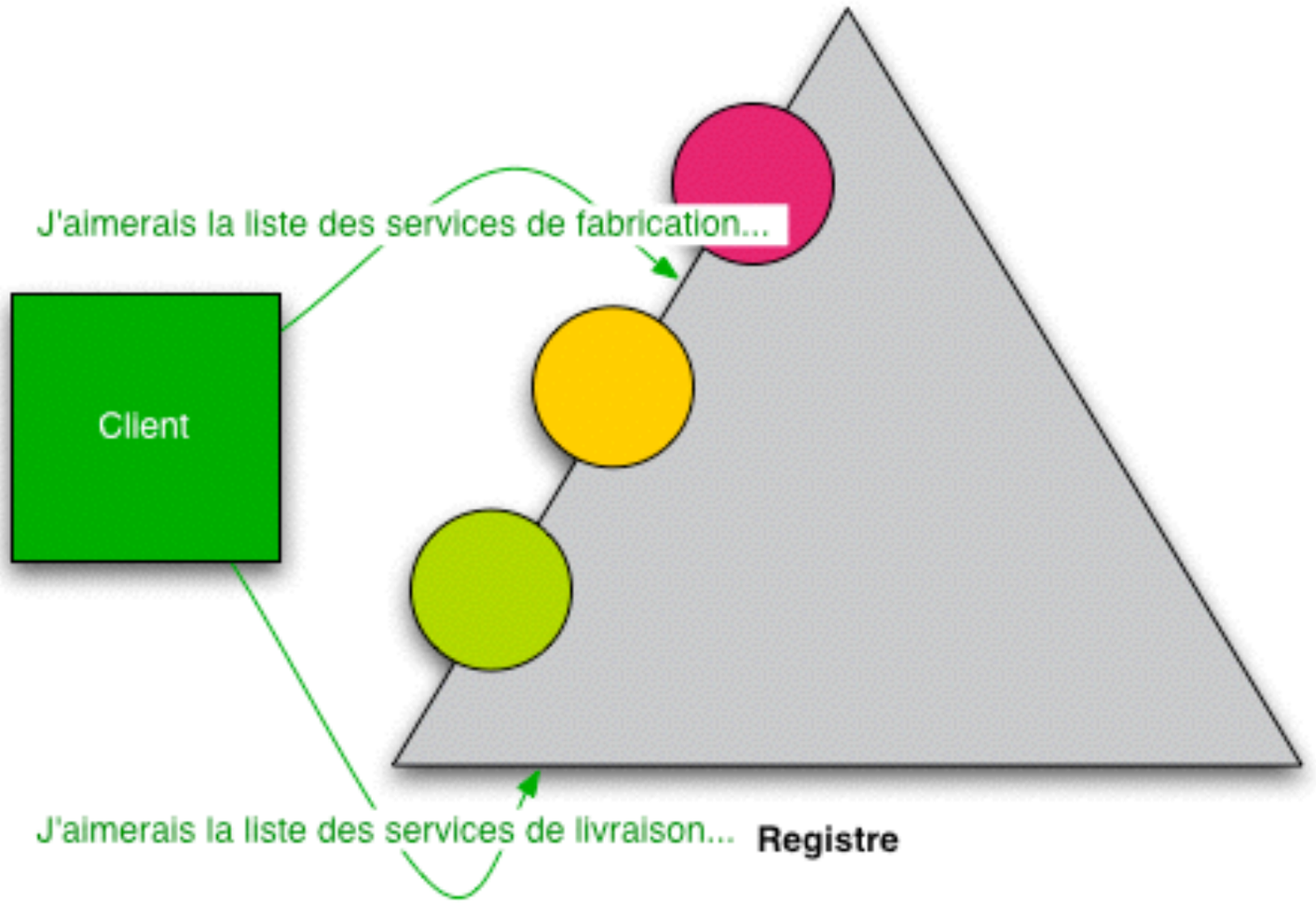
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



White Pages

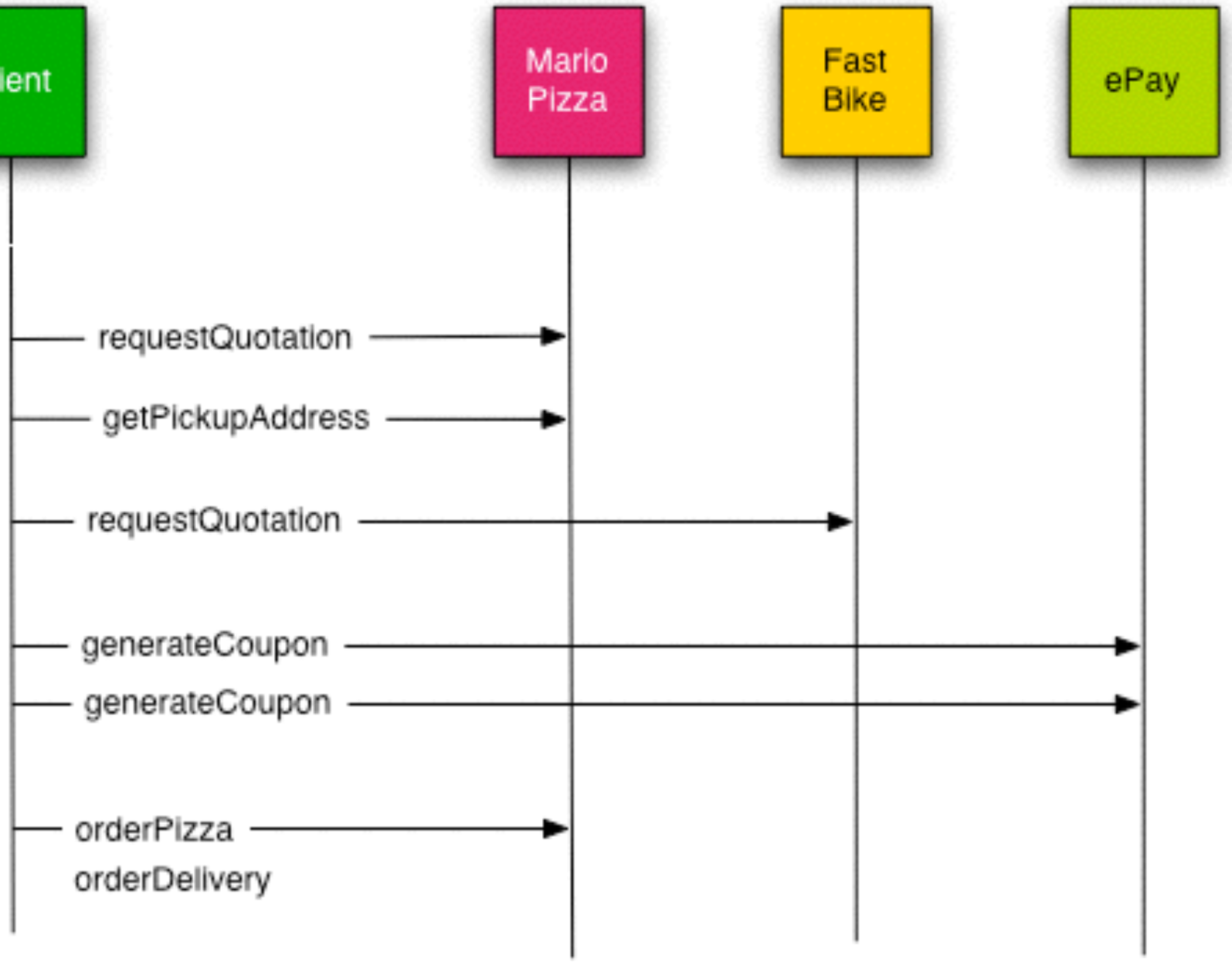
Example: Interacting with The Registry

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



Example: workflow

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



For a **Full** Service Architecture, We Need...

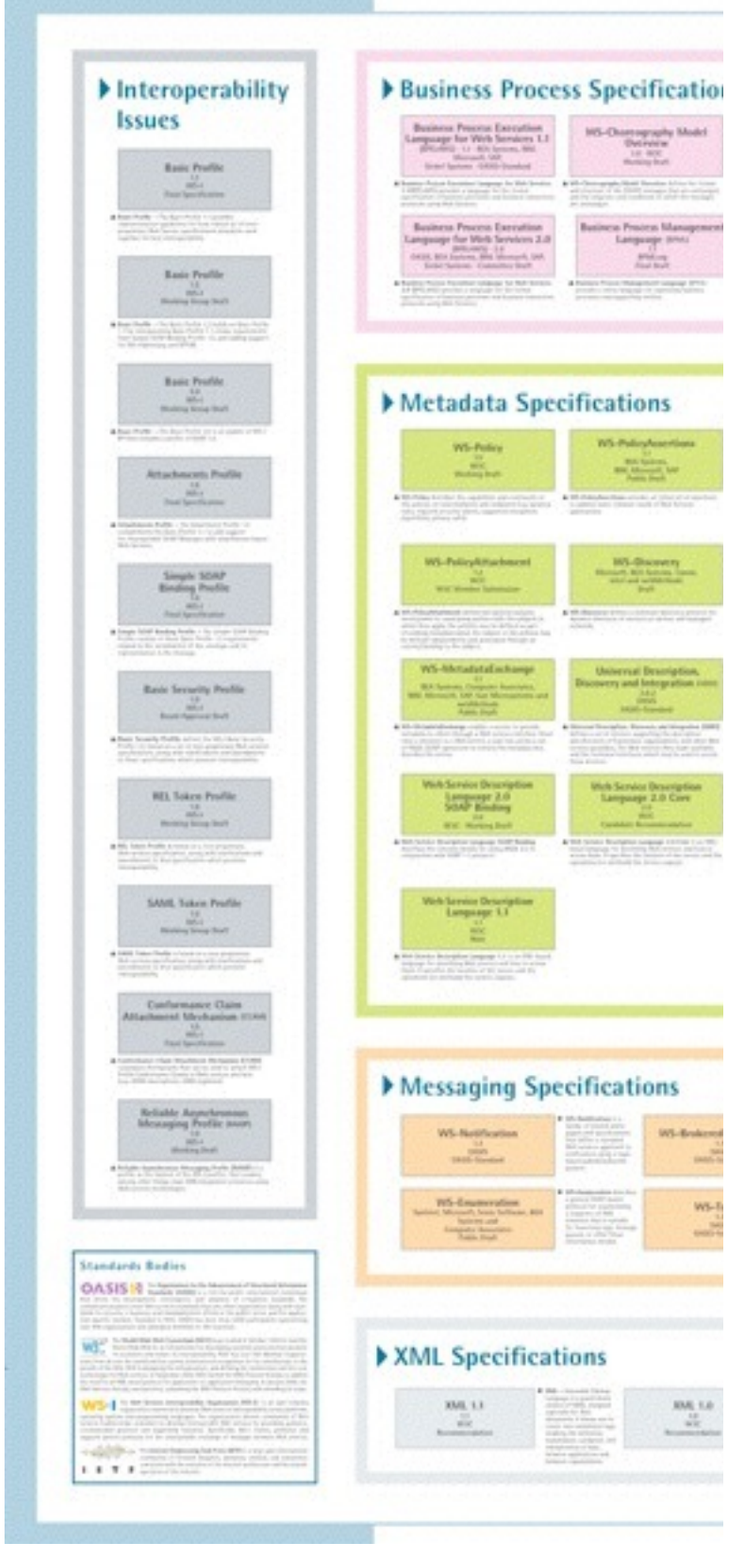
- A standardized format to **describe** service interfaces
  - Example: WSDL
- A standardized protocol to **invoke** services
  - Example: SOAP
- A **registry** service
  - Example: UDDI

The “Big” Web Services Approach

Web Services Standards Overview

http://www.innoq.com/soa/ws-standards/poster/
http://www.innoq.com/soa/ws-standards/

Web Services Standards Overview



Big Web Services

- Approach
  - Services are often designed and developed with a **RPC style** (even if Document-Oriented Services are possible).
- Core Standards
  - Simple Object Access Protocol (**SOAP**)
  - Web Services Description Language (**WSDL**)
- Benefits
  - **Very rich protocol stack** (support for security, transactions, reliable transfer, etc.)
- Problem
  - **Very rich protocol stack** (complexity, verbosity, incompatibility issues, theoretical human readability, etc.)

Let's look at it in practice. How do we use "Big" Web Services with **Java EE**?

- Create a new "**Web Application**" maven project with Netbeans.
- Create a "**Demo**" stateless session bean with a local interface:
  - add a business method that computes the sum of two long values
  - add a business method that returns the current date.
- Build and deploy on Glassfish.

Until today, to invoke the session bean, we would have created a **servlet** and called it from the **browser** (**human user**). With **web services**, we can expose our session bean via another (complementary) interface.

The **@WebService** annotation

```
@Stateless
@WebService
public class Demo implements DemoLocal {

  @Override
  public String getTime() {
    return new Date().toString();
  }

  @Override
  public long computeSum(long v1, long v2) {
    return v1 + v2;
  }

}
```

As usual, adding this annotation is **an indication that you give to the application server**. You are telling him: “When you deploy this SLSB, do what is necessary to expose it via a **SOAP/WSDL**” interface.

The **test interface** generated by Glassfish

DemoService Web Service

localhost:8080/DemoService/Demo?Tester

DemoService Web Service Tester

This form will allow you to test your web service implementation (WSDL File)

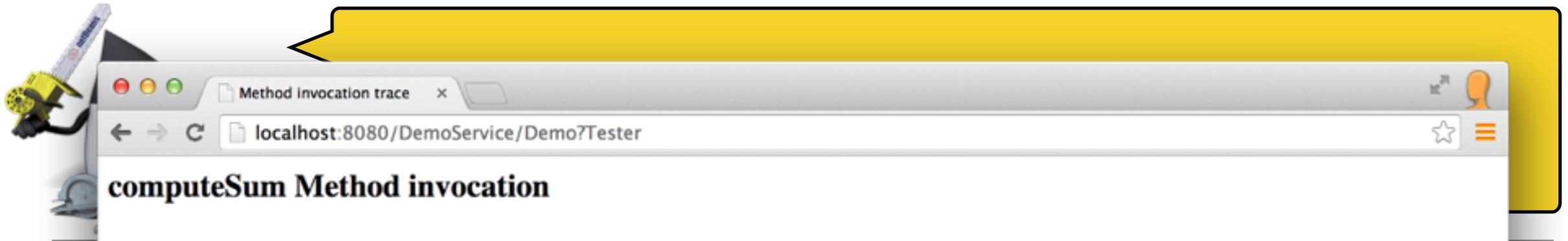
To invoke an operation, fill the method parameter(s) input boxes and click on the button labeled with the method name.

Methods :

public abstract java.lang.String ch.heigvd.amt.demo.ws.services.Demo.getTime()
getTime ()

public abstract long ch.heigvd.amt.demo.ws.services.Demo.computeSum(long,long)
computeSum (24 , 44)

If your SLSB is named "**Demo**", then access the URL **/DemoService/Demo**.
What you see in the browser is a web UI that has been generated by Glassfish that you can use to invoke your service.
With the **WSDL** hyperlink, you can also access the (machine-understandable) description of the web service interface.



Method invocation trace

localhost:8080/DemoService/Demo?Tester

computeSum Method invocation

Method parameter(s)

Type	Value
long	24
long	44

Method returned

long : "**68**"

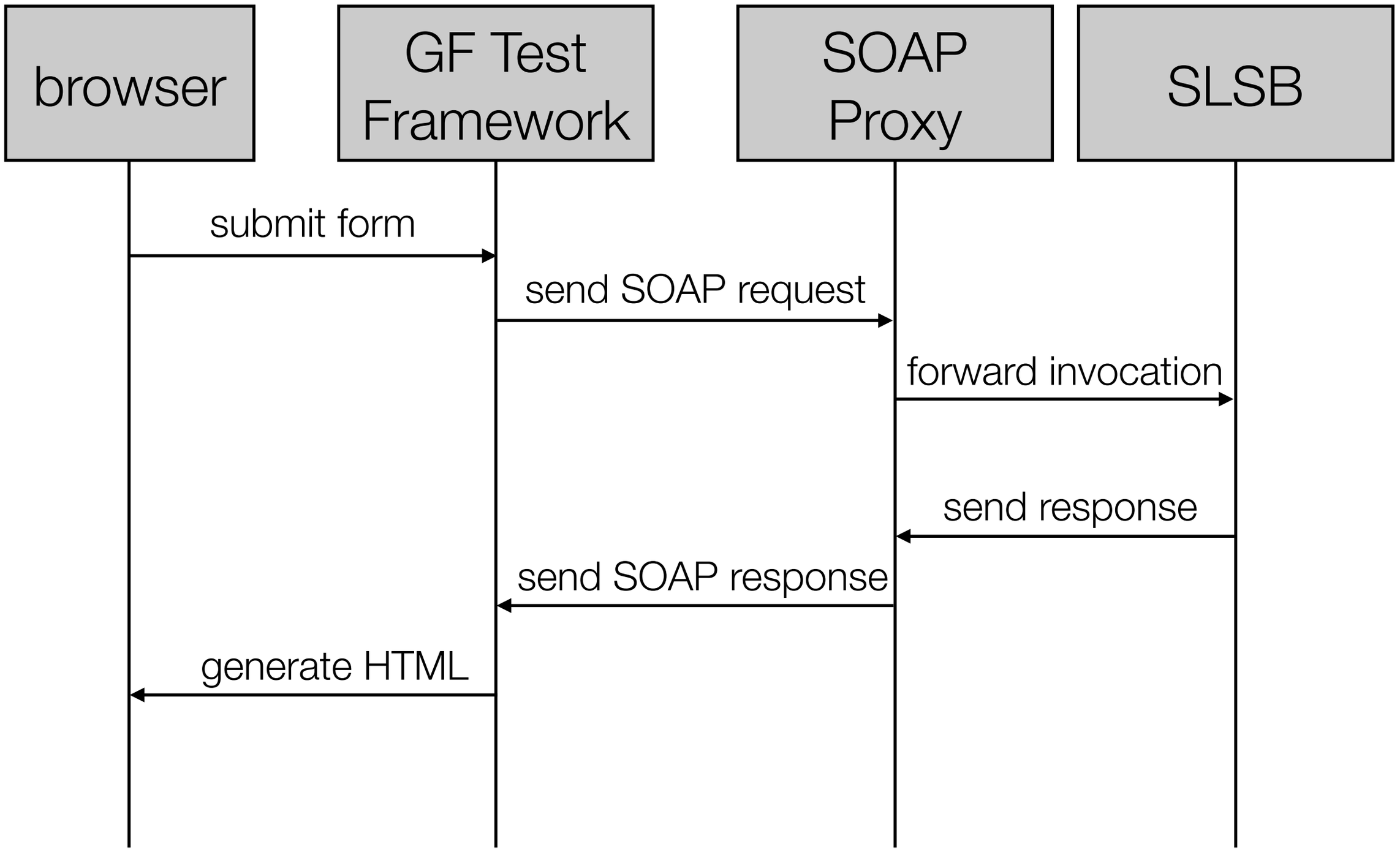
SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schema
    <SOAP-ENV:Header/>
    <S:Body>
        <ns2:computeSum xmlns:ns2="http://services.ws.demo.amt.heigvd.ch/">
            <arg0>24</arg0>
            <arg1>44</arg1>
        </ns2:computeSum>
    </S:Body>
</S:Envelope>
```

SOAP Response

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schema
    <SOAP-ENV:Header/>
    <S:Body>
        <ns2:computeSumResponse xmlns:ns2="http://services.ws.demo.amt.heigvd.ch/">
            <return>68</return>
        </ns2:computeSumResponse>
    </S:Body>
</S:Envelope>
```

What happens when you press on the "computeSum" button?



Capturing from Wi-Fi: en0 (tcp port 8080) and Loopback: lo0 (tcp port 8080) [Wireshark 1.12.1 (v1.12.1-0-g01b65bf from master-1.12)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: http Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
5	0.000195000	::1	::1	HTTP	703	POST /DemoService/Demo?Tester HTTP/1.1 (applica
12	0.001970000	127.0.0.1	127.0.0.1	HTTP/XML	302	POST /DemoService/Demo HTTP/1.1
19	0.007471000	127.0.0.1	127.0.0.1	HTTP/XML	61	HTTP/1.1 200 OK
21	0.009199000	::1	::1	HTTP	1790	HTTP/1.1 200 OK (text/html)

```
Frame 5: 703 bytes on wire (5624 bits), 703 bytes captured (5624 bits) on interface 1
Null/Loopback
Internet Protocol Version 6, Src: ::1 (::1), Dst: ::1 (::1)
Transmission Control Protocol, Src Port: 63814 (63814), Dst Port: 8080 (8080), Seq: 1, Ack: 1, Len: 627
Hypertext Transfer Protocol
HTML Form URL Encoded: application/x-www-form-urlencoded
```

When you press on the “compute sum” button, an HTML form is submitted. The HTTP request is processed by Glassfish (test framework)

```
0000  1e 00 00 00 60 07 aa c2  02 93 06 40 00 00 00 00   ....`... ...@....
0010  00 00 00 00 00 00 00 00  00 00 00 01 00 00 00 00   ........ ........
0020  00 00 00 00 00 00 00 00  00 00 00 01 f9 46 1f 90   ........ .....F..
0030  67 f2 a9 39 e0 a0 4e 4a  80 18 23 d7 02 9b 00 00   g..9..NJ ..#.....
0040  01 01 08 0a 13 09 e0 f1  13 09 e0 f1 50 4f 53 54   ........ ....POST
```

Wi-Fi: en0, Loopback: lo0: <l... | Packets: 32 · Displayed: 4 (12.5%) | Profile: Default

Capturin

File Edit View Go Ca

Filter: tcp.stream eq 0

No.	Time
1	0.000000000
2	0.000076000
3	0.000092000
4	0.000102000
5	0.000195000
6	0.000232000
21	0.009199000
22	0.009230000
28	30.256264000

Frame 5: 703 bytes on w
Null/Loopback
Internet Protocol Versi
Transmission Control Pr
Hypertext Transfer Prot
HTML Form URL Encoded:

0000 1e 00 00 00 60 07 a
0010 00 00 00 00 00 00 0
0020 00 00 00 00 00 00 0
0030 67 f2 a9 39 e0 a0 4
0040 01 01 08 0a 13 09 e

Wi-Fi: en0, Loopback:

Follow TCP Stream (tcp.stream eq 0)

Stream Content

```
POST /DemoService/Demo?Tester HTTP/1.1
Host: localhost:8080
Connection: keep-alive
Content-Length: 57
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Origin: http://localhost:8080
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_5) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/38.0.2125.111 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Referer: http://localhost:8080/DemoService/Demo?Tester
Accept-Encoding: gzip,deflate
Accept-Language: en-US,en;q=0.8,fr;q=0.6,de;q=0.4

action=computeSum&PARAMcomputeSum0=34&PARAMcomputeSum1=12HTTP/1.1 200 OK
Server: GlassFish Server Open Source Edition  4.1
X-Powered-By: Servlet/3.1 JSP/2.3 (GlassFish Server Open Source Edition  4.1  Java/
Oracle Corporation/1.8)
Server: grizzly/2.3.15
Pragma: no-cache
Content-Type: text/html;charset=UTF-8
Date: Thu, 06 Nov 2014 05:43:45 GMT
Content-Length: 1395

<HTML lang=en><HEAD><TITLE>Method invocation trace</TITLE></HEAD><H2><A> computeSum </A>
Method invocation</H2><BR><HR><h4>Method parameter(s)</h4><table
border="1"><tr><th>Type</th><th>Value</th></tr><tr><td>long</td><td><pre>34</pre></td></
tr><tr><td>long</td><td><pre>12</pre></td></tr></table><HR><h4>Method returned</
h4>long : "<b>46</b>"<HR><h4>SOAP Request</h4><HR><blockquote><pre xml:lang>&lt;?xml
version="1.0" encoding="UTF-8"?&gt;&lt;S:Envelope xmlns:S="http://schemas.xmlsoap.org/
soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"&gt;
    &lt;SOAP-ENV:Header/&gt;
    &lt;S:Body&gt;
        &lt;ns2:computeSum xmlns:ns2="http://services.ws.demo.amt.heigvd.ch/"&gt;
            &lt;arg0&gt;34&lt;/arg0&gt;
            &lt;arg1&gt;12&lt;/arg1&gt;
```

Entire conversation (2341 bytes)

Find | Save As | Print | ASCII | EBCDIC | Hex Dump | C Arrays | Raw

Help | Filter Out This Stream | Close

n=65535 Len=0 MSS=163
=0 Ack=1 Win=65535 Le
k=1 Win=146800 Len=0
63814 [ACK] Seq=1 Ack
ster HTTP/1.1 (appli
k=628 Win=146176 Len=
ml)
Ack=1715 Win=145088 L
=1715 Ack=628 Win=146

Profile: Default

Capturing from Wi-Fi: en0 (tcp port 8080) and Loopback: lo0 (tcp port 8080) [Wireshark 1.12.1 (v1.12.1-0-g01b65bf from master-1.12)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: http Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
5	0.000195000	::1	::1	HTTP	703	POST /DemoService/Demo?Tester HTTP/1.1 (applica
12	0.001970000	127.0.0.1	127.0.0.1	HTTP/XML	302	POST /DemoService/Demo HTTP/1.1
19	0.007471000	127.0.0.1	127.0.0.1	HTTP/XML	61	HTTP/1.1 200 OK
21	0.009199000	::1	::1	HTTP	1790	HTTP/1.1 200 OK (text/html)

```
Frame 5: 703 bytes on wire (5624 bits), 703 bytes captured (5624 bits) on interface 1
Null/Loopback
Internet Protocol Version 6, Src: ::1 (::1), Dst: ::1 (::1)
Transmission Control Protocol, Src Port: 63814 (63814), Dst Port: 8080 (8080), Seq: 1, Ack: 1, Len: 627
Hypertext Transfer Protocol
HTML Form URL Encoded: application/x-www-form-urlencoded
```

When Glassfish processes the form, it sends a SOAP message to your web service (i.e. to your session bean). It receives a SOAP response and uses it to return the HTML response.

```
0000  1e 00 00 00 60 07 aa c2  02 93 06 40 00 00 00 00   ....`... ...@....
0010  00 00 00 00 00 00 00 00  00 00 00 01 00 00 00 00   ........ ........
0020  00 00 00 00 00 00 00 00  00 00 00 01 f9 46 1f 90   ........ .....F..
0030  67 f2 a9 39 e0 a0 4e 4a  80 18 23 d7 02 9b 00 00   g..9..NJ ..#.....
0040  01 01 08 0a 13 09 e0 f1  13 09 e0 f1 50 4f 53 54   ........ ....POST
```

Wi-Fi: en0, Loopback: lo0: <l... Packets: 32 · Displayed: 4 (12.5%) Profile: Default

Capturing from Wi-Fi: en0 (tcp port 8080) and Loopback: lo0 (tcp port 8080) [Wireshark 1.12.1 (v1.12.1-0-g01b65bf from master-1.12)]

File Edit View Go Ca

Filter: tcp.stream eq 1

No.	Time
7	0.001506000
8	0.001547000
9	0.001557000
10	0.001564000
11	0.001956000
12	0.001970000
13	0.001981000
14	0.001991000
15	0.007329000

Frame 12: 302 bytes on
Null/Loopback
Internet Protocol Versi
Transmission Control Pr
[2 Reassembled TCP Segr
Hypertext Transfer Prot
eXtensible Markup Langu

0000 02 00 00 00 45 00
0010 7f 00 00 01 7f 00
0020 aa 47 7a 63 80 18

Frame (302 bytes) Reass

=65535 Len=0 MSS=163
=0 Ack=1 Win=65535 Le
=1 Win=146976 Len=0
53815 [ACK] Seq=1 Ack
bled PDU]
TP/1.1
=459 Win=146528 Len=
=705 Win=146272 Len=
bled PDU]

Wi-Fi: en0, Loopback: lo0: <l... Packets: 32 · Displayed: 19 (59.4%) Profile: Default

Follow TCP Stream (tcp.stream eq 1)

Stream Content

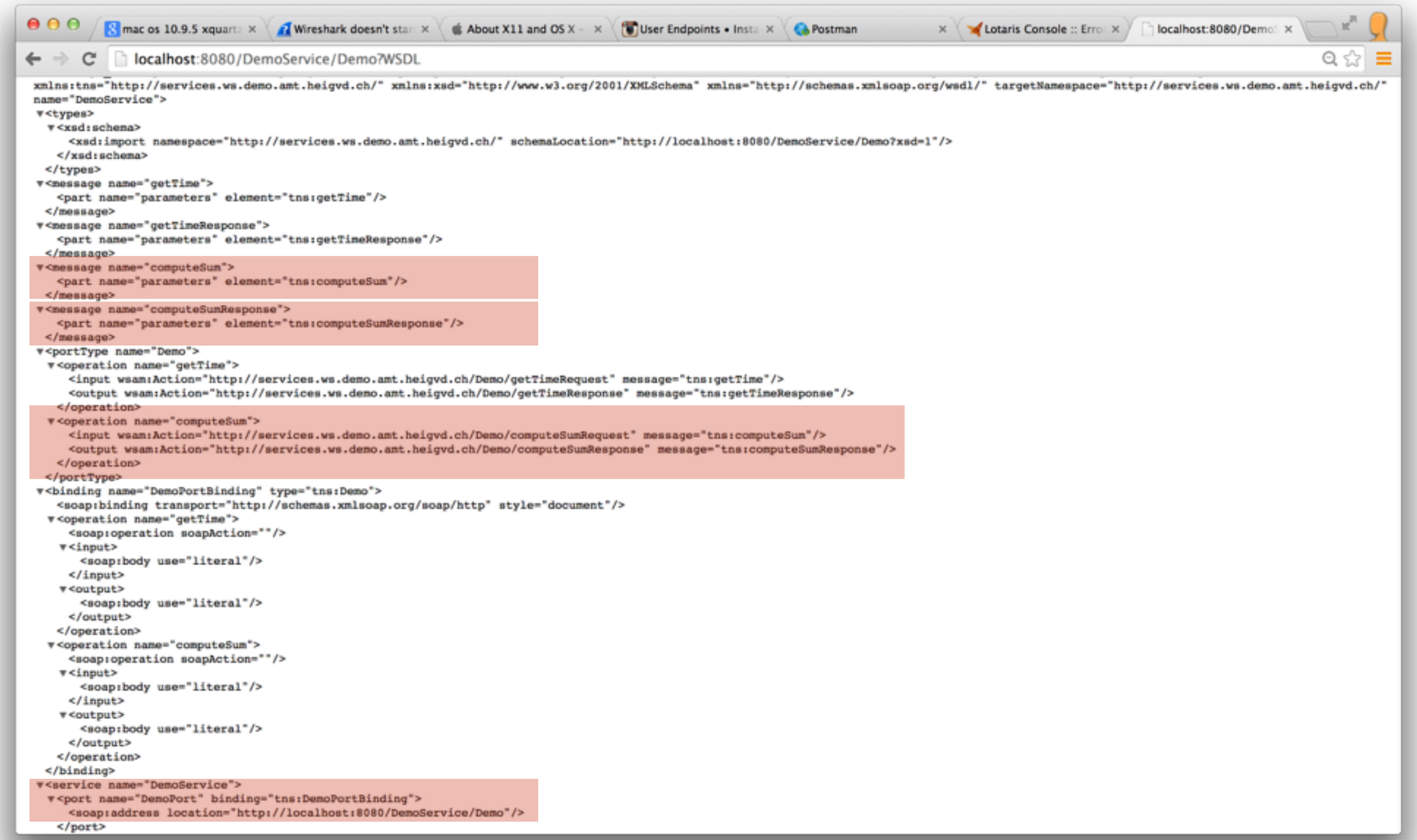
```
POST /DemoService/Demo HTTP/1.1
Accept: text/xml, multipart/related
Content-Type: text/xml; charset=utf-8
SOAPAction: "http://services.ws.demo.amt.heigvd.ch/Demo/computeSumRequest"
User-Agent: Metro/2.3.1-b419 (branches/2.3.1.x-7937; 2014-08-04T08:11:03+0000) JAXWS-
RI/2.2.10-b140803.1500 JAXWS-API/2.2.11 JAXB-RI/2.2.10-b140802.1033 JAXB-API/2.2.12-
b140109.1041 svn-revision#unknown
Host: localhost:8080
Connection: keep-alive
Content-Length: 246

<?xml version='1.0' encoding='UTF-8'?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/
envelope/"><S:Body><ns2:computeSum xmlns:ns2="http://
services.ws.demo.amt.heigvd.ch/"><arg0>34</arg0><arg1>12</arg1></ns2:computeSum></S:Body></
S:Envelope>HTTP/1.1 200 OK
Server: GlassFish Server Open Source Edition  4.1
X-Powered-By: Servlet/3.1 JSP/2.3 (GlassFish Server Open Source Edition  4.1  Java/Oracle
Corporation/1.8)
Server: grizzly/2.3.15
Content-Type: text/xml; charset=utf-8
Date: Thu, 06 Nov 2014 05:43:45 GMT
Transfer-Encoding: chunked

6e
<?xml version='1.0' encoding='UTF-8'?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/
envelope/"><S:Body>
8d
<ns2:computeSumResponse xmlns:ns2="http://services.ws.demo.amt.heigvd.ch/"><return>46</
return></ns2:computeSumResponse></S:Body></S:Envelope>
0
```

Entire conversation (1279 bytes)

Find | Save As | Print | ASCII | EBCDIC | Hex Dump | C Arrays | Raw

Help | Filter Out This Stream | Close

The WSDL file describes the interface of the web service. It is generated by Glassfish.




```
xmlns:tns="http://services.ws.demo.amt.heigvd.ch/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://services.ws.demo.amt.heigvd.ch/"
name="DemoService">
▼<types>
  ▼<xsd:schema>
     <xsd:import namespace="http://services.ws.demo.amt.heigvd.ch/" schemaLocation="http://localhost:8080/DemoService/Demo?xsd=1"/>
   </xsd:schema>
 </types>
▼<message name="getTime">
   <part name="parameters" element="tns:getTime"/>
 </message>
▼<message name="getTimeResponse">
   <part name="parameters" element="tns:getTimeResponse"/>
 </message>
▼<message name="computeSum">
   <part name="parameters" element="tns:computeSum"/>
 </message>
▼<message name="computeSumResponse">
   <part name="parameters" element="tns:computeSumResponse"/>
 </message>
▼<portType name="Demo">
  ▼<operation name="getTime">
     <input wsam:Action="http://services.ws.demo.amt.heigvd.ch/Demo/getTimeRequest" message="tns:getTime"/>
     <output wsam:Action="http://services.ws.demo.amt.heigvd.ch/Demo/getTimeResponse" message="tns:getTimeResponse"/>
   </operation>
  ▼<operation name="computeSum">
     <input wsam:Action="http://services.ws.demo.amt.heigvd.ch/Demo/computeSumRequest" message="tns:computeSum"/>
     <output wsam:Action="http://services.ws.demo.amt.heigvd.ch/Demo/computeSumResponse" message="tns:computeSumResponse"/>
   </operation>
 </portType>
▼<binding name="DemoPortBinding" type="tns:Demo">
   <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
  ▼<operation name="getTime">
     <soap:operation soapAction=""/>
    ▼<input>
       <soap:body use="literal"/>
     </input>
    ▼<output>
       <soap:body use="literal"/>
     </output>
   </operation>
  ▼<operation name="computeSum">
     <soap:operation soapAction=""/>
    ▼<input>
       <soap:body use="literal"/>
     </input>
    ▼<output>
       <soap:body use="literal"/>
     </output>
   </operation>
 </binding>
▼<service name="DemoService">
  ▼<port name="DemoPort" binding="tns:DemoPortBinding">
     <soap:address location="http://localhost:8080/DemoService/Demo"/>
   </port>
```

Let's create a Java client for our web service.

- Create a new "**Java Application**" maven project with Netbeans.
- Use the **New Web Service Client** wizard.
  - Select the project that you just created to deploy your web service into Glassfish (and select DemoService)
  - Select the appropriate package (it is a good idea to add a "wsclient" or "client" level).
- The Java platform supports web services with the JAX-WS API and associated tools. The wsimport tool generates JAX-WS artifacts from a WSDL file.

https://netbeans.org/bugzilla/show_bug.cgi?id=241570

Let's create a Java client for our web service.

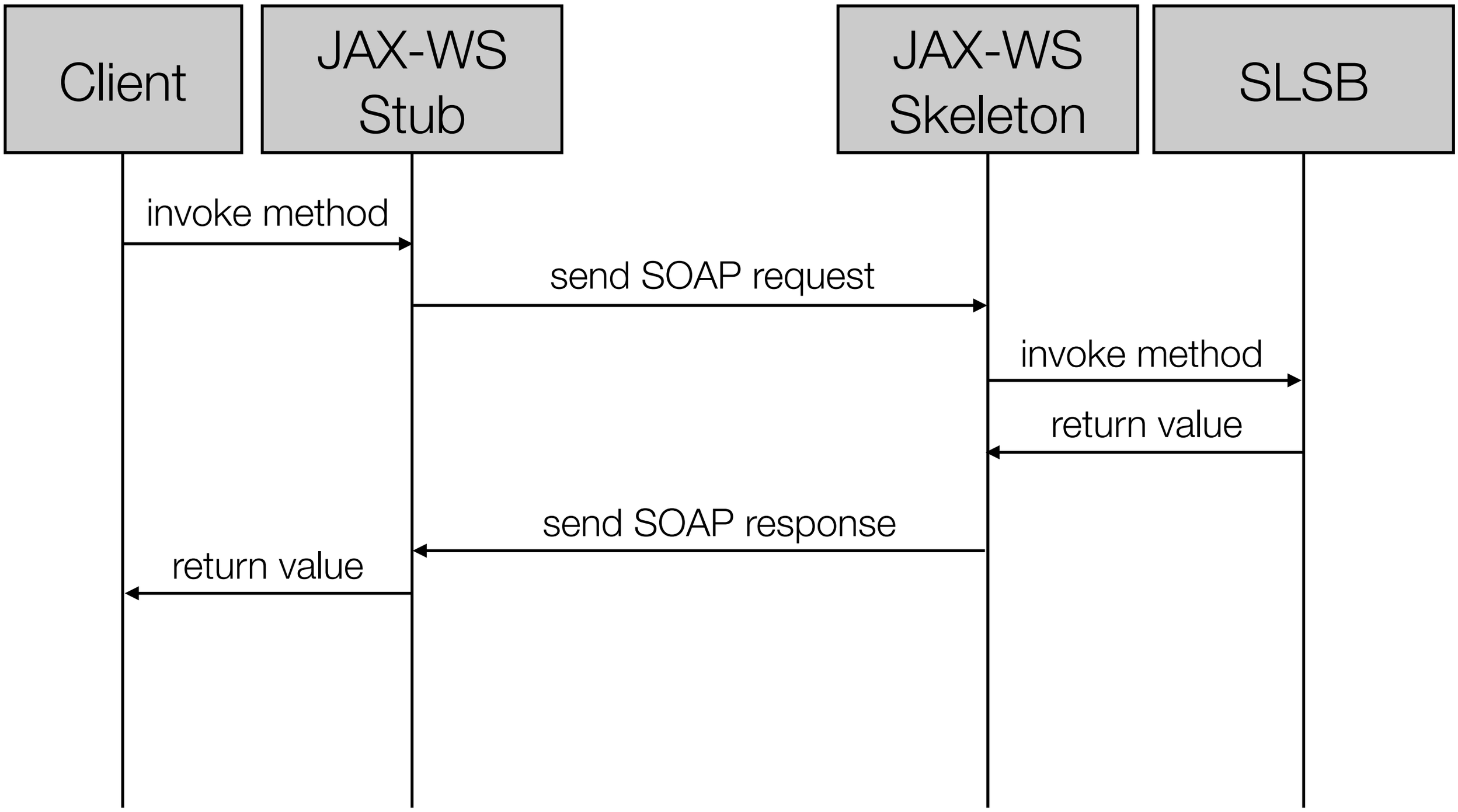
```
<plugin>
  <groupId>org.jvnet.jax-ws-commons</groupId>
  <artifactId>jaxws-maven-plugin</artifactId>
  <version>2.3</version>
  <configuration>
    <!-- Needed with JAXP 1.5 -->
    <vmArgs>
      <vmArg>-Djavax.xml.accessExternalSchema=all</vmArg>
    </vmArgs>
  </configuration>
</plugin>
```

Because of this bug, you need to add the fragment above to your POM.xml
https://netbeans.org/bugzilla/show_bug.cgi?id=241570

We can now use the web service stubs from our Java class.

```
public class Client {

  /**
   * @param args the command line arguments
   */
  public static void main(String[] args) {
    DemoService service = new DemoService();
    Demo demo = service.getDemoPort();
    System.out.println("It is now " + demo.getTime());
    System.out.println("21 + 32 = " + demo.computeSum(21, 32));
  }

}
```

What happens when you press on the “computeSum” button?



Capturing from Wi-Fi: en0 (tcp port 8080) and Loopback: lo0 (tcp port 8080) [Wireshark 1.12.1 (v1.12.1-0-g01b65bf from master-1.12)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: xml Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
6	0.008215000	127.0.0.1	127.0.0.1	HTTP/XML	250	POST /DemoService/Demo HTTP/1.1
12	0.012184000	127.0.0.1	127.0.0.1	HTTP/XML	61	HTTP/1.1 200 OK
16	0.018675000	127.0.0.1	127.0.0.1	HTTP/XML	286	POST /DemoService/Demo HTTP/1.1
23	0.024375000	127.0.0.1	127.0.0.1	HTTP/XML	61	HTTP/1.1 200 OK

```
Frame 6: 250 bytes on wire (2000 bits), 250 bytes captured (2000 bits) on interface 1
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
Transmission Control Protocol, Src Port: 64794 (64794), Dst Port: 8080 (8080), Seq: 348, Ack: 1, Len: 194
[2 Reassembled TCP Segments (541 bytes): #5(347), #6(194)]
Hypertext Transfer Protocol
eXtensible Markup Language
```

```
System.out.println("It is now " + demo.getTime());
System.out.println("21 + 32 = " + demo.computeSum(21, 32));
```

```
0000  02 00 00 00 45 00 00 f6  17 73 40 00 40 06 00 00   ....E... .s@.@...
0010  7f 00 00 01 7f 00 00 01  fd 1a 1f 90 8c e0 7c cc   ........ ......|.
0020  fb 7f 08 eb 80 18 23 e2  fe ea 00 00 01 01 08 0a   ......#. ........
```

Frame (250 bytes) | Reassembled TCP (541 bytes)

Wi-Fi: en0, Loopback: lo0: <l... | Packets: 29 · Displayed: 4 (13.8%) | Profile: Default

Follow TCP Stream (tcp.stream eq 0)

Stream Content

```
POST /DemoService/Demo HTTP/1.1
Accept: text/xml, multipart/related
Content-Type: text/xml; charset=utf-8
SOAPAction: "http://services.ws.demo.amt.heigvd.ch/Demo/getTimeRequest"
User-Agent: JAX-WS RI 2.2.9-b130926.1035 svn-
revision#5f6196f2b90e9460065a4c2f4e30e065b245e51e
Host: localhost:8080
Connection: keep-alive
Content-Length: 194

<?xml version="1.0" ?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/
envelope/"><S:Body><ns2:getTime xmlns:ns2="http://services.ws.demo.amt.heigvd.ch/"></
ns2:getTime></S:Body></S:Envelope>HTTP/1.1 200 OK
Server: GlassFish Server Open Source Edition  4.1
X-Powered-By: Servlet/3.1 JSP/2.3 (GlassFish Server Open Source Edition  4.1  Java/
Oracle Corporation/1.8)
Server: grizzly/2.3.15
Content-Type: text/xml; charset=utf-8
Date: Thu, 06 Nov 2014 06:41:36 GMT
Transfer-Encoding: chunked

6e
<?xml version='1.0' encoding='UTF-8'?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/
soap/envelope/"><S:Body>
a1
<ns2:getTimeResponse xmlns:ns2="http://services.ws.demo.amt.heigvd.ch/"><return>Thu Nov
06 07:41:36 CET 2014</return></ns2:getTimeResponse></S:Body></S:Envelope>
0

POST /DemoService/Demo HTTP/1.1
Accept: text/xml, multipart/related
Content-Type: text/xml; charset=utf-8
SOAPAction: "http://services.ws.demo.amt.heigvd.ch/Demo/computeSumRequest"
User-Agent: JAX-WS RI 2.2.9-b130926.1035 svn-
revision#5f6196f2b90e9460065a4c2f4e30e065b245e51e
Host: localhost:8080
Connection: keep-alive
Content-Length: 230

<?xml version="1.0" ?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/
envelope/"><S:Body><ns2:computeSum xmlns:ns2="http://
services.ws.demo.amt.heigvd.ch/"><arg0>21</arg0><arg1>32</arg1></ns2:computeSum></
S:Body></S:Envelope>HTTP/1.1 200 OK
```

Entire conversation (2291 bytes)

Find | Save As | Print | ASCII | EBCDIC | Hex Dump | C Arrays | Raw

Help | Filter Out This Stream | Close

Simple Object Access Protocol (SOAP)

- Description
  - SOAP is a lightweight protocol for service invocation.
  - SOAP defines the structure of messages exchanged by clients and services.
  - SOAP messages can be exchanged via different transport protocols. HTTP is only one these protocols.
- Origin
  - SOAP a été spécifié suite à l'explosion de XML, en 1998.
- Specifications
  - La spécification SOAP 1.2 est une recommendation du W3C (27 avril 2007)
  - http://www.w3.org/TR/soap/

SOAP Version 1.2

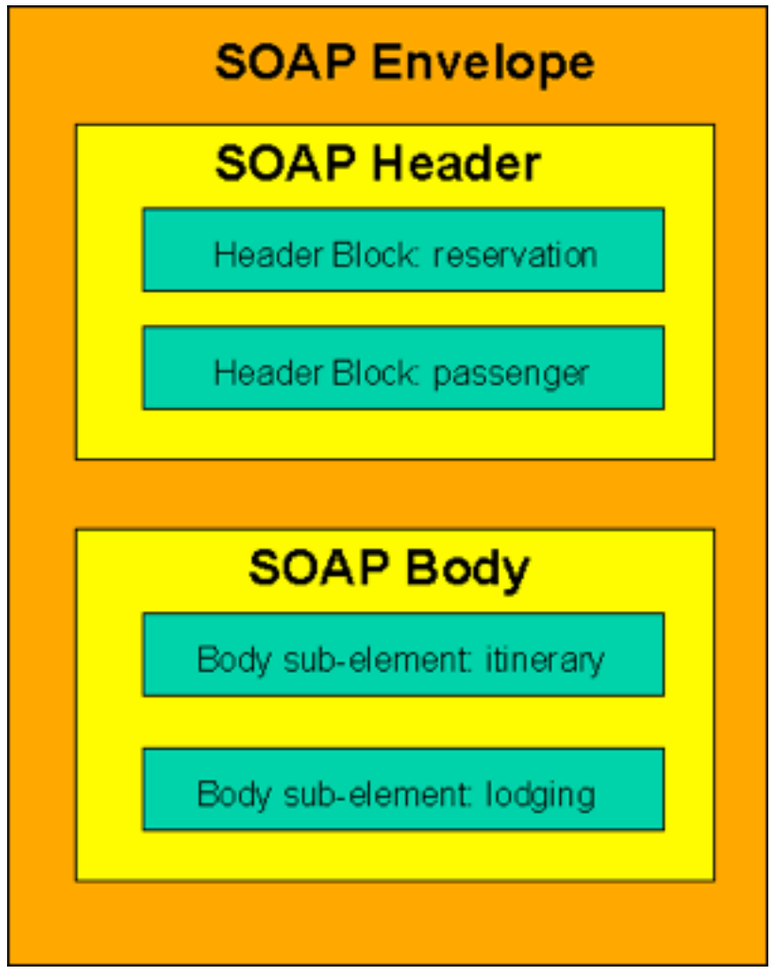
Latest version of SOAP Version 1.2 specification: http://www.w3.org/TR/soap12

W3C Recommendation (Second Edition) 27 April 2007

SOAP Version 1.2 Part0: Primer
http://www.w3.org/TR/2007/REC-soap12-part0-20070427/ (errata)
SOAP Version 1.2 Part1: Messaging Framework
http://www.w3.org/TR/2007/REC-soap12-part1-20070427/ (errata)
SOAP Version 1.2 Part2: Adjuncts
http://www.w3.org/TR/2007/REC-soap12-part2-20070427/ (errata)
SOAP Version 1.2 Specification Assertions and Test Collection
http://www.w3.org/TR/2007/REC-soap12-testcollection-20070427/ (errata)

Structure of a SOAP Message

- Header
  - Used to capture properties of the message or of the exchange.
  - Example: security management.
  - One of the extension points (all the properties have not been defined a priori).
- Body
  - The applicative payload.
  - Can capture a method invocation, with parameters.
  - Can capture a document (e.g. an order) to be processed by the service.



http://www.w3.org/TR/2007/REC-soap12-part0-20070427/

Web Services Description Language (WSDL)

- SOAP is useful:
  - when we know “where” the service is (i.e. we know the service endpoint)
  - when we know the signature of the methods supported by the service
- But SOAP does not help in:
  - searching / looking up a service that complies to a certain interface
  - automatically generating a “stub” to be used on the client side
  - hence, we need a way to formally describe service interfaces!!
- WSDL: Web Services Description Language addresses this need
  - and thus allows the **automation** of procedures when dealing with web services.

As communications protocols and message formats are standardized in the web community, it becomes increasingly possible and important to be able to **describe the communications in some structured way**.

WSDL addresses this need by defining an XML grammar for **describing network services as collections of communication endpoints capable of exchanging messages**.

WSDL service definitions provide **documentation for distributed systems** and serve as a recipe for **automating** the details involved in applications communication.

http://www.w3.org/TR/wsdl

WSDL Structure

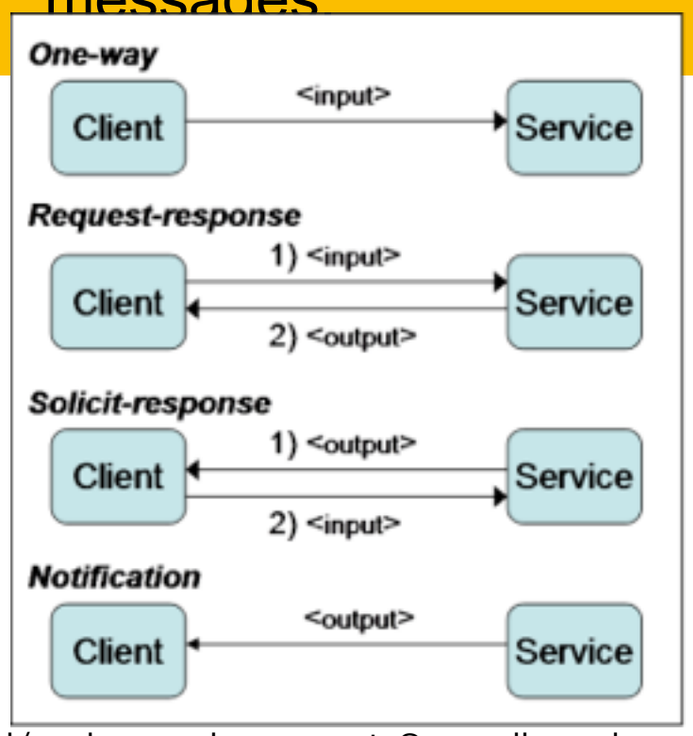
heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
<wsdl:definitions name="nmtoken"? targetNamespace="uri"?>
    <import namespace="uri" location="uri"/>*
    <wsdl:documentation .... /> ?

    <wsdl:types> ?
        <wsdl:documentation .... />?
        <xsd:schema .... />*
        <-- extensibility element --> *
    </wsdl:types>

    <wsdl:message name="nmtoken"> *
        <wsdl:documentation .... />?
        <part name="nmtoken" element="qname"? type="qname"?/> *
    </wsdl:message>

    <wsdl:portType name="nmtoken">*
        <wsdl:documentation .... />?
        <wsdl:operation name="nmtoken">*
            <wsdl:documentation .... /> ?
            <wsdl:input name="nmtoken"? message="qname">?
                <wsdl:documentation .... /> ?
            </wsdl:input>
            <wsdl:output name="nmtoken"? message="qname">?
                <wsdl:documentation .... /> ?
            </wsdl:output>
            <wsdl:fault name="nmtoken" message="qname"> *
                <wsdl:documentation .... /> ?
            </wsdl:fault>
        </wsdl:operation>
    </wsdl:portType>
```

Data Type Definitions:
Person, Contract, Account, etc.

Message: abstract representation of data exchanged

Port type: set of operations. Each operation has input and output messages



http://www.ajaxonomy.com/2008/xml/web-services-part-2-wsdl-and-wadl

Structure WSDL

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
<wsdl:binding name="nmtoken" type="qname">*
    <wsdl:documentation .... />?
    <-- extensibility element --> *
    <wsdl:operation name="nmtoken">*
       <wsdl:documentation .... /> ?
       <-- extensibility element --> *
       <wsdl:input> ?
           <wsdl:documentation .... /> ?
           <-- extensibility element -->
       </wsdl:input>
       <wsdl:output> ?
           <wsdl:documentation .... /> ?
           <-- extensibility element --> *
       </wsdl:output>
       <wsdl:fault name="nmtoken"> *
           <wsdl:documentation .... /> ?
           <-- extensibility element --> *
       </wsdl:fault>
    </wsdl:operation>
</wsdl:binding>

<wsdl:service name="nmtoken"> *
    <wsdl:documentation .... />?
    <wsdl:port name="nmtoken" binding="qname"> *
       <wsdl:documentation .... /> ?
       <-- extensibility element -->
    </wsdl:port>
    <-- extensibility element -->
</wsdl:service>
<-- extensibility element --> *
</wsdl:definitions>
```

Binding: defines message format and protocol details for operations and messages defined by a particular portType.

Port: A port defines an individual endpoint by specifying a single address for a binding.

Service: set of related ports.

JAX-WS Lab

Lab

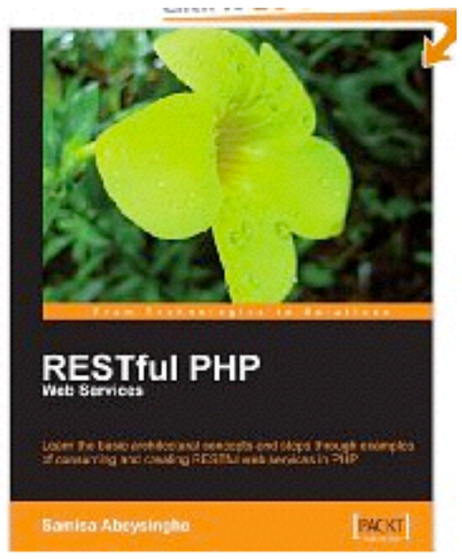
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- **Create a GUI client application (Swing)**
  - The user can enter **sensor data** (sensor id, type, value, etc.) in text fields.
  - The user can press a "**Send**" button.
- **Create a service (Java EE)**
  - A **Stateless Session Bean** implements a "createMeasure" method, which accepts sensor data as parameters.
  - Note: you can use a list of primitive types in the method signature, but you can also use a "SensorData" POJO. You will notice that JAX-WS handles the serialization of the data for you!
- **Integrate the client and the service with JAX-WS**
  - Expose the SLSB with the @WebService interface. Test it with the Glassfish test framework.
  - Generate the JAX-WS stubs in the GUI application and use them to invoke the service.
  - Use Wireshark to inspect the communication and validate the request/response sequence.

The REST Approach

MSE | MASTER OF SCIENCE IN ENGINEERING

RESTful Web Services



The REST Architectural Style

- REST: **REpresentational State Transfer**
- REST is an **architectural style** for building distributed systems.
- REST has been introduced in **Roy Fielding's Ph.D. thesis** (Roy Fielding has been a contributor to the HTTP specification, to the apache server, to the apache community).
- The WWW is **one example** for a distributed system that exhibits the characteristics of a REST architecture.

Principles of a REST Architecture

- The state of the application is captured in a **set of resources**
  - Users, photos, comments, tags, albums, etc.
- Every resource can be **identified with a standard format** (e.g. URL)
- Every resource can have **several representations**
- There is one **unique interface for interacting** with resources (e.g. HTTP methods)
- The communication protocol is:
  - client-server
  - stateless
  - cacheable
- These properties have a positive impact on systemic qualities (scalability, performance, availability, etc.).
  - Reference: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Reference

- Very good article, with presentation of key concepts and illustrative examples:
  - http://www.infoq.com/articles/rest-introduction

HTTP is a protocol for interacting with "**resources**"

What is a “Resource”

- At first glance, one could think that a “resource” is a file on a web server:
  - an HTML document, an XML document, a PNG document
- That fits the vision of the “static content” web
- But of course, the web is now more than a huge library of hypermedia documents:
  - through the web, we interact with services and a lot of the content is dynamic.
  - more and more, through the web we interact with physical objects (machines, sensors, actuators)
  - We need a more generic definition for resources!

What is a “Resource”?

- A resource is "something" that can be named and uniquely identified:
  - Example 1: an article published in the "24 heures" newspaper
  - Example 2: the collection of articles published in the sport section of the newspaper
  - Example 3: a person’s resume
  - Example 4: the current price of the Nestlé stock quote
  - Example 5: the vending machine in the school hallway
  - Example 6: the list of grades of the student Jean Dupont
- URL (Uniform Resource Locator) is a mechanism for identifying resources
  - Exemple 1: http://www.24heures.ch/vaud/vaud/2008/08/04/trente-etudiants-partent-rencontre-patrons
  - Exemple 2: http://www.24heures.ch/articles/sport
  - Exemple 5: http://www.smart-machines.ch/customers/heig/machines/8272

Resource vs. Representation

- A "resource" can be something intangible (stock quote) or tangible (vending machine)
- The HTTP protocol supports the exchange of data between a client and a server.
- Hence, what is exchanged between a client and a server is **not** the resource. It is a **representation** of a resource.
- Different representations of the same resource can be generated:
  - HTML representation
  - XML representation
  - PNG representation
  - WAV representation
- **HTTP provides the content negotiation mechanisms!!**

How Do We Interact With Resources?

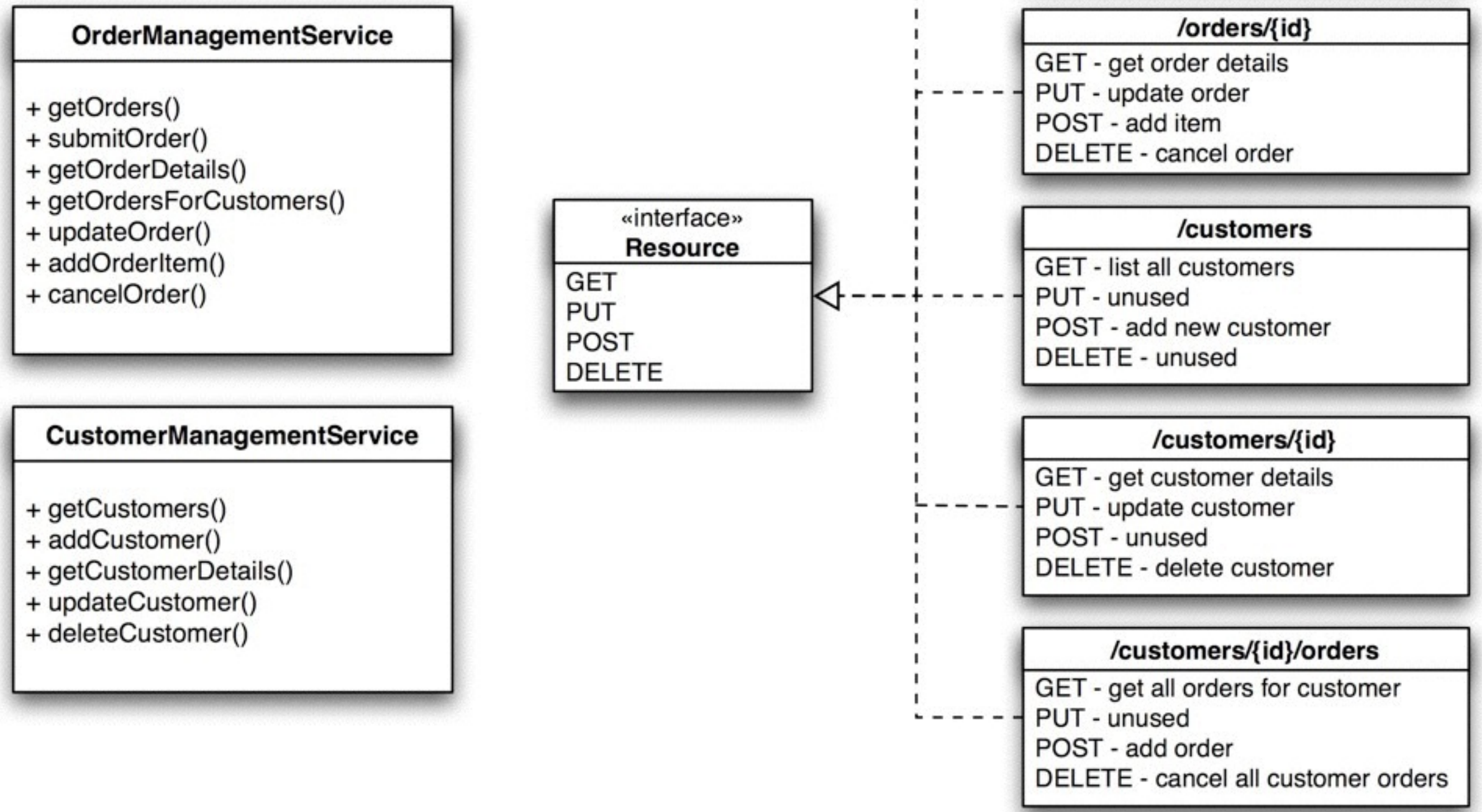
- The HTTP protocol defines the standard methods. These methods enable the interactions with the resources:
  - **GET**: retrieve whatever information is identified by the Request-URI
  - **POST**: used to request that the origin server accept the entity enclosed in the request as a new subordinate of the ressource identified by the Request-URI in the Request-Line
  - **PUT**: requests that the enclosed entity be stored under the supplied Request-URI.
  - **DELETE**: requests that the origin server delete the ressource identified by the Request-URI.
  - HEAD: identical to GET except that the server MUST NOT return a message-body in the response
  - TRACE: used for debugging (echo)
  - CONNECT: reserved for tunneling purposes
  - **PATCH**: used for partial updates

How should I specify/document my REST API?

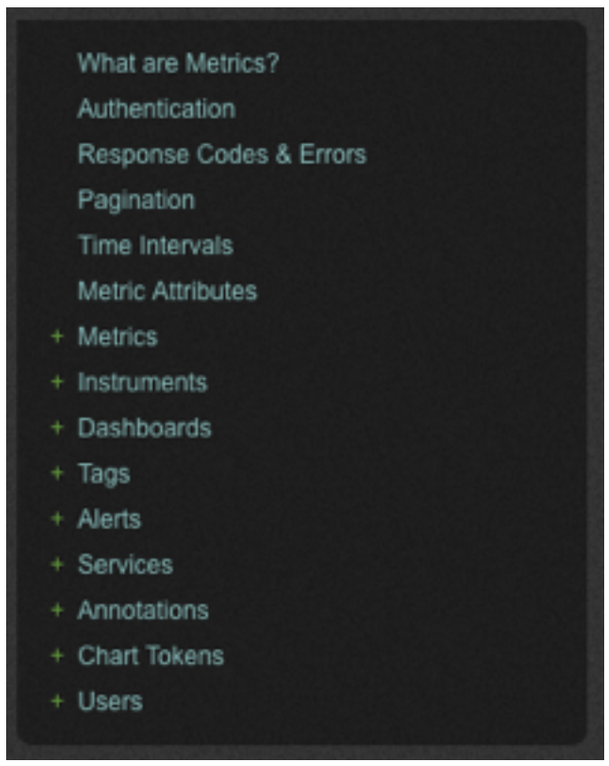
Design a RESTful system

- Start by identifying the **resources** - the **NAMES** in your system.
- Define the **structure of the URLs** that will be mapped to your resources.
- Define the **semantic of the operations** that you want to support on all of your resources (you don’t want to support GET, POST, PUT, DELETE on all resources!).
- Some examples:
  - http://www.photos.com/users/oliechti identifies a resource of type "user". A client can do a "HTTP GET" to obtain a representation of the user or a “HTTP PUT” to update the user.
  - http://www.photos.com/users identifies a resource of type "collection of users". A client can do a "HTTP POST" to add users, or an "HTTP GET" to obtain the list of users.

RPC vs REST

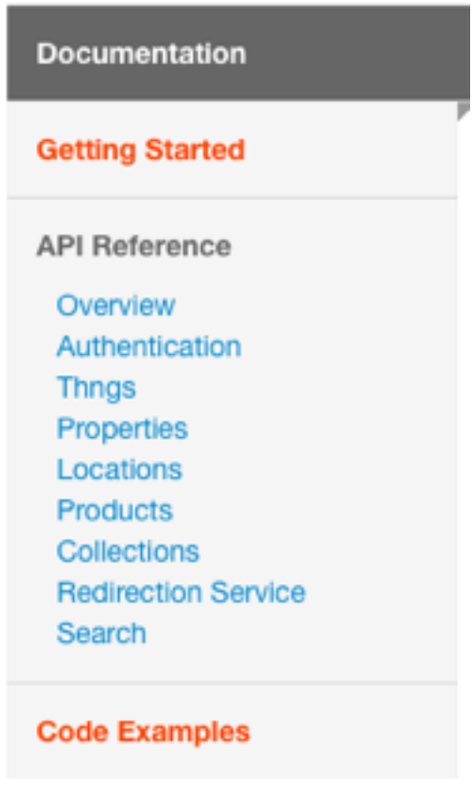

OrderManagementService
+ getOrders()
+ submitOrder()
+ getOrderDetails()
+ getOrdersForCustomers()
+ updateOrder()
+ addOrderItem()
+ cancelOrder()
CustomerManagementService
+ getCustomers()
+ addCustomer()
+ getCustomerDetails()
+ updateCustomer()
+ deleteCustomer()
«interface»
Resource
GET
PUT
POST
DELETE
/orders
GET - list all orders
PUT - unused
POST - add a new order
DELETE - unused
/orders/{id}
GET - get order details
PUT - update order
POST - add item
DELETE - cancel order
/customers
GET - list all customers
PUT - unused
POST - add new customer
DELETE - unused
/customers/{id}
GET - get customer details
PUT - update customer
POST - unused
DELETE - delete customer
/customers/{id}/orders
GET - get all orders for customer
PUT - unused
POST - add order
DELETE - cancel all customer orders

Look at Some Examples



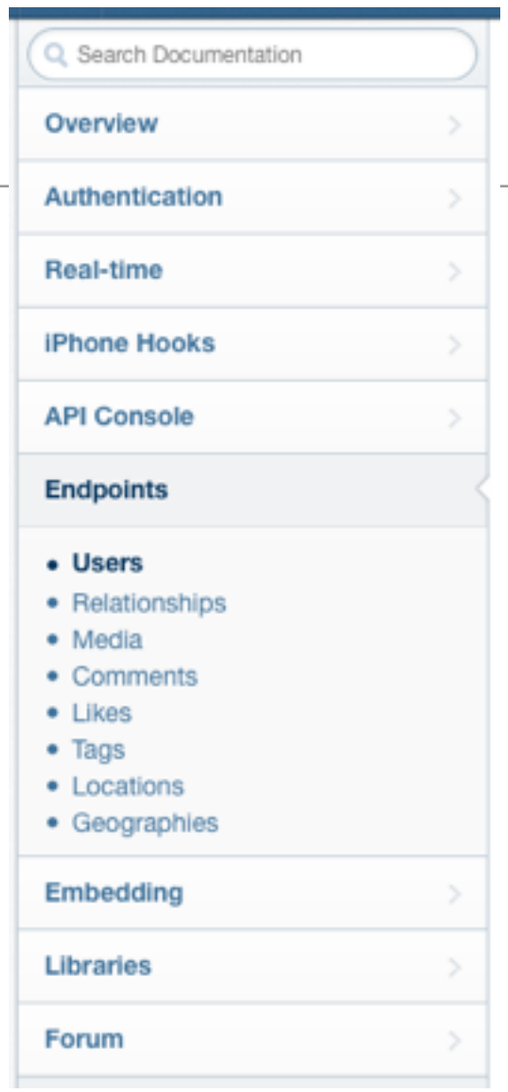
http://dev.librato.com/v1

librato



https://dev.evrythng.com/documentation/api

EVRYTHNG™



http://instagram.com/developer/endpoints/

Instagram

Short description of the resource (domain model)

Examples & payload structure

CRUD method description

navigator

What Are Metrics?

Metrics are custom measurements stored in Librato's Metrics service. These measurements are created and may be accessed programatically through a set of RESTful API calls. There are currently two types of metrics that may be stored in Librato Metrics, **gauges** and **counters**.

Gauges

Gauges capture a series of measurements where each measurement represents the value under observation at one point in time. The value of a gauge typically varies between some known minimum and maximum. Examples of gauge measurements include the requests/second serviced by an application, the amount of available disk space, the current value of $AAPL, etc.

Counters

Counters track an increasing number of occurrences of some event. A counter is unbounded and always monotonically increasing in any given *run*. A new *run* is started anytime that counter is reset to zero. Examples of counter measurements include the number of connections made to an app, the number of visitors to a website, the number of a times a write operation failed, etc.

Metric Properties

Some common properties are supported across all types of metrics:

`name`

Each metric has a name that is unique to its class of metrics e.g. a gauge name must be un
gauges. The name identifies a metric in subsequent API calls to store/query individual meas
can be up to 63 characters in length. Valid characters for metric names are 'A-Za-z0-9.:-_'.

`period`

The **period** of a metric is an integer value that describes (in seconds) the standard reportin
metric. Setting the period enables Metrics to detect abnormal interruptions in reporting and

Examples

Return the metric named *cpu_temp* with up to four measurements at resolution 60.

```
curl \
  -u <user>:<token> \
  -X GET \
  https://metrics-api.librato.com/v1/metrics/cpu_temp?resolution=60&count=4
```

Response Code

```
200 OK
```

Response Headers

```
** NOT APPLICABLE **
```

Response Body

```
{
  "type": "gauge",
  "display_name": "cpu_temp",
  "resolution": 60,

  ": {
e.com": [

    1,
_time": 1234567890,
    84.5

    1,
_time": 1234567950,
    86.7

    1,
_time": 1234568010,
    84.6

    1,
_time": 1234568070,
    89.7

(
": 0,
nsform": null,
ts_short": "&#176;F",
ua": "librato-metrics/0.7.4 (ruby; 1.9.3p194; x86_64-linux) direct-faraday/0.8.4
": null,
ts_long": "Fahrenheit",
cked": true

  "Current CPU temperature in Fahrenheit",
emp"
```

GET /v1/metrics/:name

API VERSION 1.0

Description

Returns information for a specific metric. If time interval search parameters are specified will also include a set of metric measurements for the given time span.

URL

```
https://metrics-api.librato.com/v1/metrics/:name
```

Method

```
GET
```

Measurement Search Parameters

If optional time interval search parameters are specified, the response includes the set of metric measurements covered by the time interval. Measurements are listed by their originating *source* name if one was specified when the measurement was created. All measurements that were created without an explicit source name are listed with the source name **unassigned**.

`source`

Deprecated: Use **sources** with a single source name, e.g **[mysource]**.

`sources`

If **sources** is specified, the response is limited to measurements from those sources. The **sources** parameter should be specified as an array of source names. The response is limited to the set of sources specified in the array.

MSE | MASTER OF SCIENCE IN ENGINEERING

Short description of the whole domain model

Overview

The central data structure in our engine are `Thngs`, which are data containers to store all the data generated by and about any physical object. Various `Properties` can be attached to any Thng, and the content of each property can be updated any time, while preserving the history of those changes. Thngs can be added to various `Collections` which makes it easier to share a set of Thngs with other `Users` within the engine.

`Thng`

An abstract notion of an object which has location & property data associated to it. Also called Active Digital Identities (ADIs), these resources can model real-world elements such as persons, places, cars, guitars, mobile phones, etc.

`Property`

A Thng has various properties: arbitrary key/value pairs to store any data. The values can be updated individually at any time, and can be retrieved historically (e.g. "Give me the values of property X between 10 am and 5 pm on the 16th August 2012").

`Location`

Each Thng also has a special type of Properties used to store snapshots of its geographic position over time (for now only GPS coordinates - latitude and longitude).

`User`

Each interaction with the EVRYTHNG back-end is authenticated and a user is associated with each action. This dictates security access.

`Collection`

A collection is a grouping of Thngs. Col
one collection.

CRUD method description

Creating a new Product

To create a new `Product`, simply POST a JSON document that describes a product to the `/products` endpoint.

```
POST /products
Content-Type: application/json
Authorization: $EVRYTHNG_API_KEY
```

```
{
  *"fn": <String>,
  "description": <String>,
  "brand": <String>,
  "categories": [<String>, ...],
  "photos": [<String>, ...],
  "url": <String>,
  "identifiers": {
    <String>: <String>,
    ... },
  "properties": {
    <String>: <String>,
    ... },
  "tags": [<String>, ...]
}
```

Mandatory Parameters

`fn`

<String> The functional name of the product.

Optional Parameters

`description`

<String> An string that gives more details about the product, a short description.

More details about the Product resource (domain model) & payload structure

Products

Products are very similar to thngs, but instead of modeling an individual object instance, products are used to model a class of objects. Usually, they are used for general classes of thngs, usually a particular model with specific characteristics. Let's take for example a specific TV model (e.g. this one), which has various properties such as a model number, a description, a brand, a category, etc. Products are useful to captor the properties that are common to a set of thngs (so you don't replicate a property "model name" or "weight" for thousands of thngs that are individual instances of a same product category).

The Product document model used in our engine has been designed to be compatible with the hProduct microformat, therefore it can easily be integrated with the hProduct data model and applications supporting microformats.

The Product document model is as follows:

```
<Product>={
  "id": <String>,
  "createdAt": <timestamp>,
  "updatedAt": <timestamp>,
  "fn": <String>,
  "description": <String>,
  "brand": <String>,
  "categories": [<String>, ...],
  "photos": [<String>, ...],
  "url": <String>,
  "identifiers": {
    <String>: <String>,
    ... },
  "properties": {
    <String>: <String>,
        ... },
  "tags": [<String>, ...]
}
```

Cross-cutting concerns

Pagination

Requests that return multiple items will be paginated to 30 items by default. You can specify further pages with the `?page` parameter. You can also set a custom page size up to 100 with the `?per_page` parameter.

Authentication

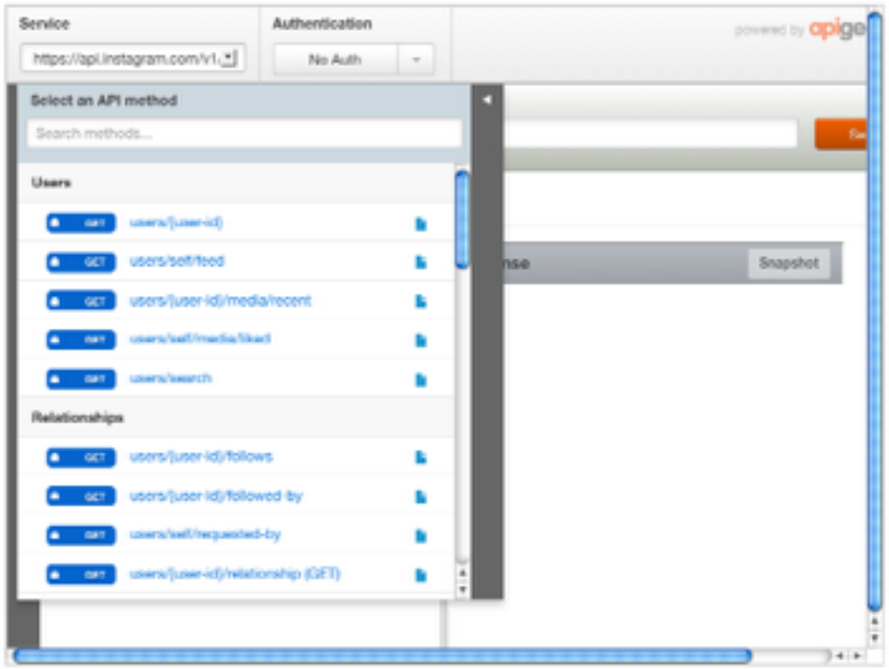
Access to our API is done via HTTPS requests to the `https://api.evrythng.com` domain. Unencrypted HTTP requests are accepted (`http://api.evrythng.com` for low-power device without SSL support), but we **strongly** suggest to use only HTTPS if you store any valuable data in our engine. Every request to our API must include an API key using `Authorization` HTTP header to identify the user or application issuing the request and execute it if authorized.

MSE | MASTER OF SCIENCE IN ENGINEERING

Interactive test console

API Console

Our API console is provided by Apigee. Tap the Lock icon, select OAuth, and you can experiment with making requests to our API. See it in full screen →



List of supported CRUD methods for each resource (R, R/W)

User Endpoints

GET	/users/ user-id	Get basic information about a user.
GET	/users/self/feed	See the authenticated user's feed.
GET	/users/ user-id /media/recent	Get the most recent media published by a user.
GET	/users/self/media/liked	See the authenticated user's list of liked media.
GET	/users/search	Search for a user by name.

Comment Endpoints

GET	/media/ media-id /comments	Get a full list of comments on a media.
POST	/media/ media-id /comments	Create a comment on a media. Please email apide...
DEL	/media/ media-id /comments/ comment-id	Remove a comment.

GET /media/ **media-id** /comments

https://api.instagram.com/v1/media/555/comments?access_token=ACCESS-TOKEN — RESPONSE

```
{
    "meta": {
        "code": 200
    },
    "data": [
        {
            "created_time": "1280780324",
            "text": "Really amazing photo!",
            "from": {
                "username": "snoopdogg",
                "profile_picture": "http://images.instagram.com/profiles/profile_16_75sq_1305612434.jpg",
                "id": "1574083",
                "full_name": "Snoop Dogg"
            },
            "id": "420"
        },
        ...
    ]
}
```

Get a full list of comments on a media.

Required scope: comments

CRUD method description

Cross-cutting concerns

Limits

Be nice. If you're sending too many requests too quickly, we'll send back a 503 error code (server unavailable).

You are limited to 5000 requests per hour per access_token or client_id overall. Practically, this means you should (when possible) authenticate users so that limits are well outside the reach of a given user.

PAGINATION

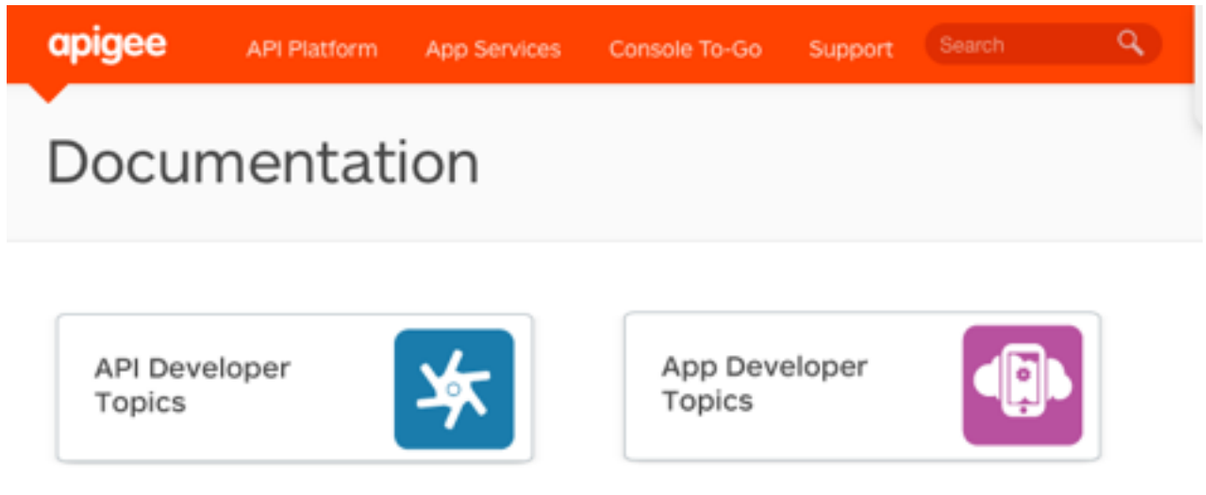
Sometimes you just can't get enough. For this reason, we've provided a convenient way to access more data in any request for sequential data. Simply call the url in the next_url parameter and we'll respond with the next set of data.

The Envelope

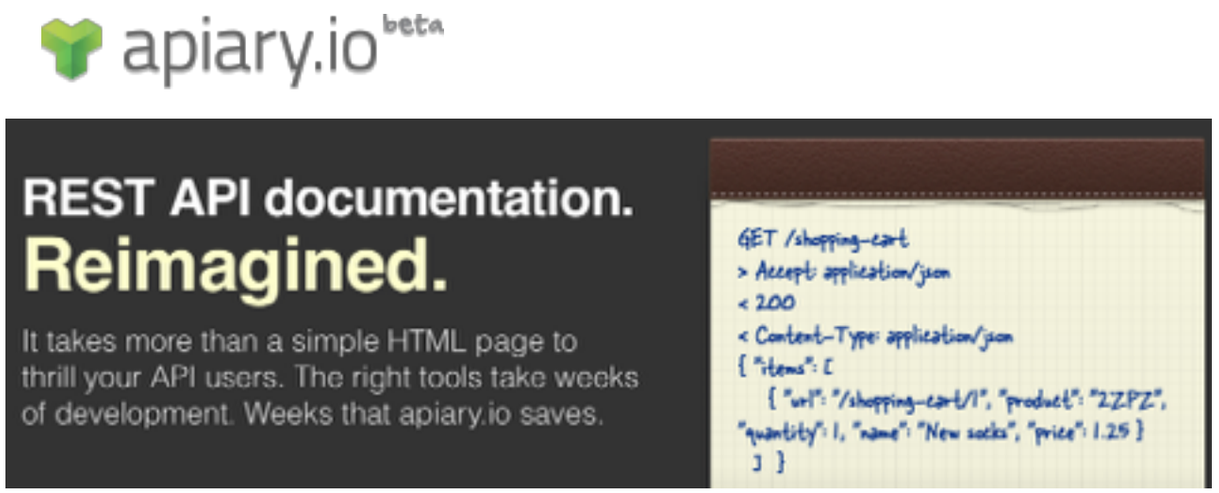
Every response is contained by an envelope. That is, each response has a predictable set of keys with which you can expect to interact:

```
{
    "meta": {
        "code": 200
    },
    "data": {
        ...
    },
    "pagination": {
        "next_url": "...",
        "next_max_id": "13872296"
    }
}
```

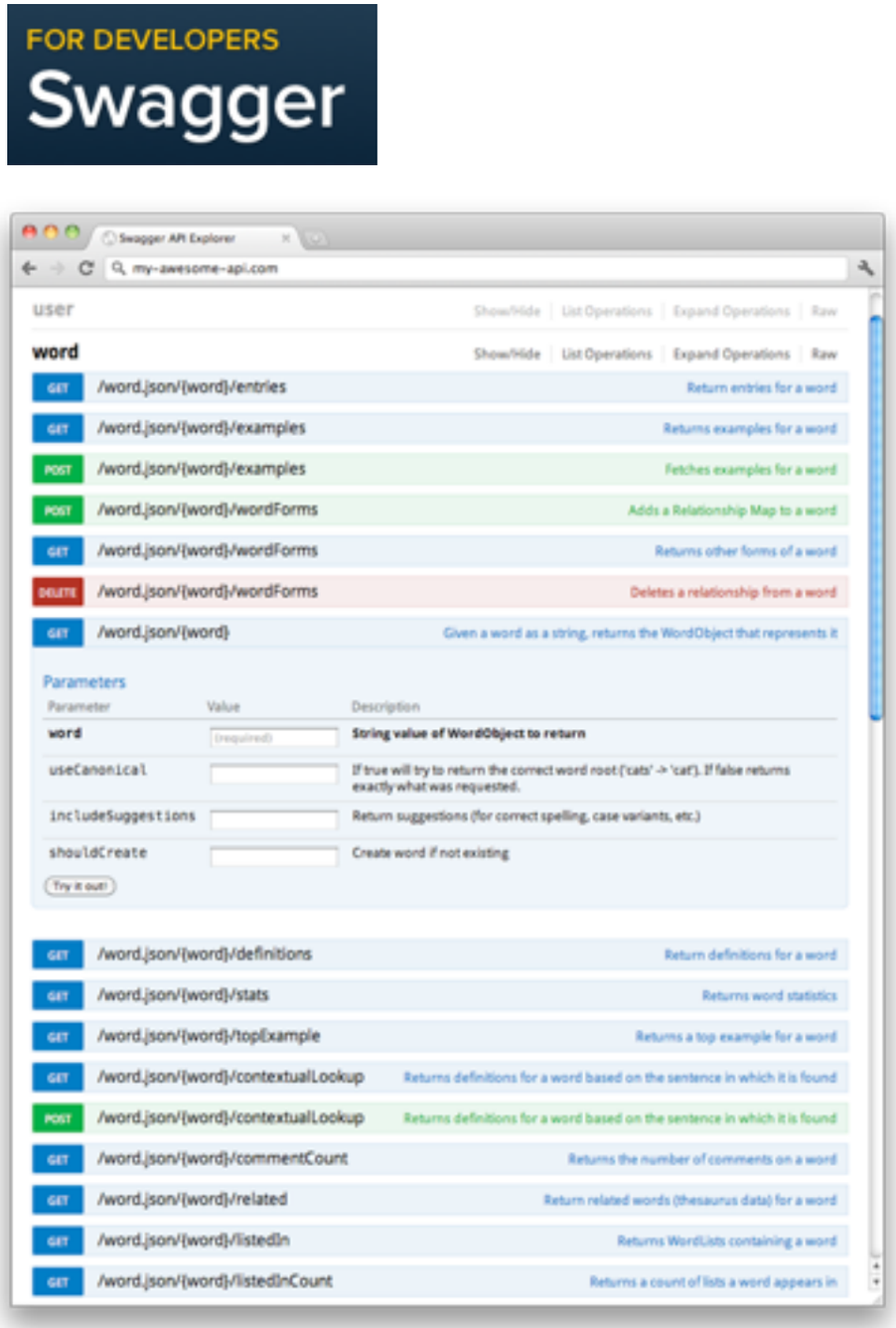
Some Tools that Might Help/Inspire You



http://apigee.com/docs/



http://apiary.io/

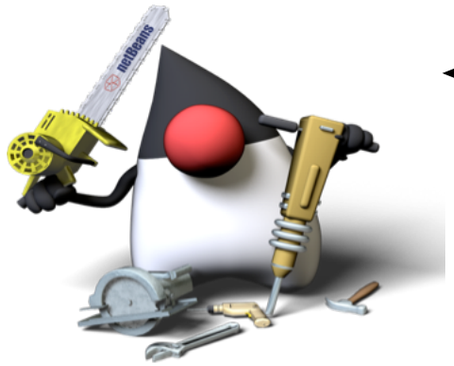


https://developers.helloreverb.com/swagger/

How to write a “RESTful” Web Service?

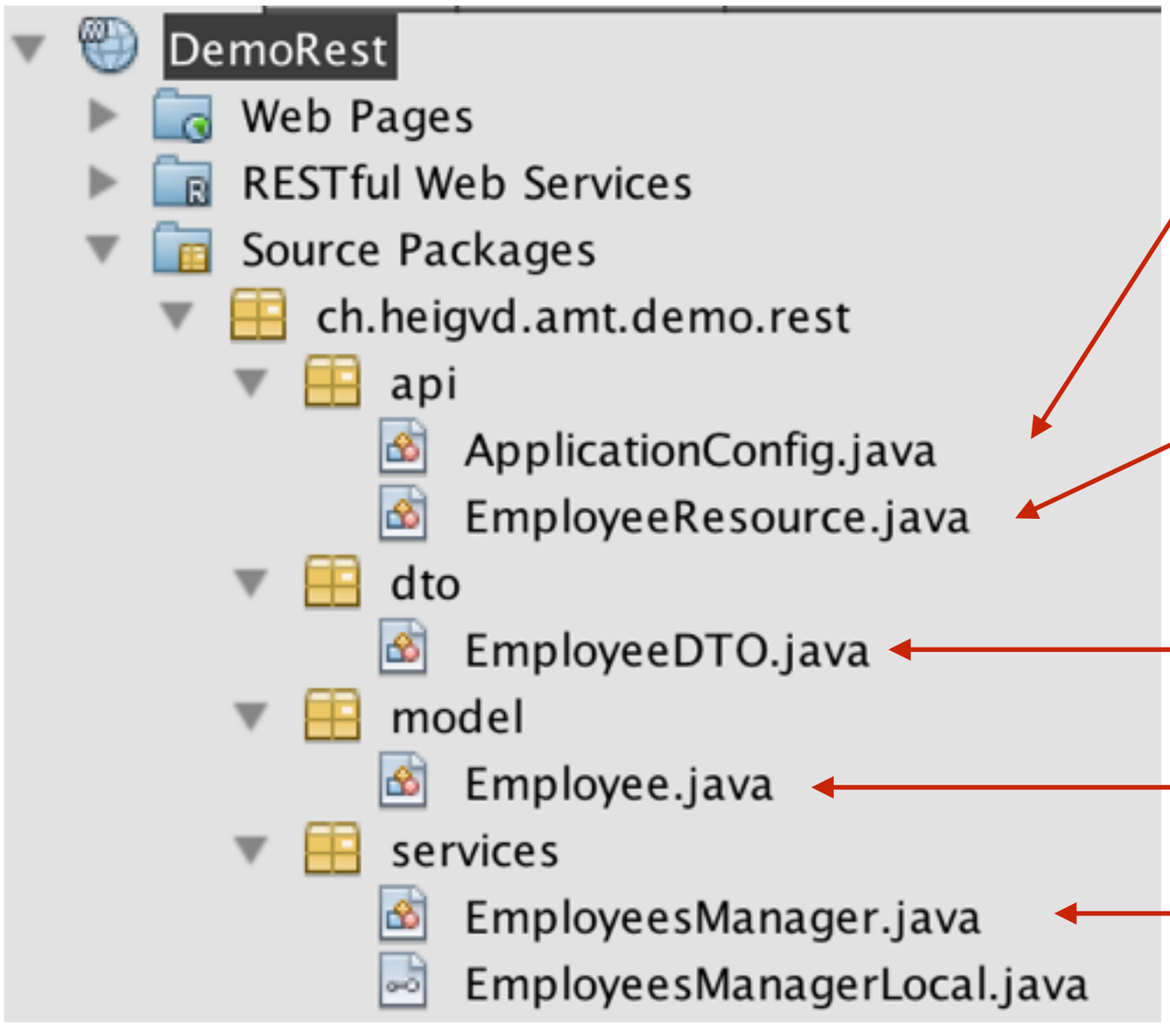
- On the server side, one could do everything in a FrontController servlet:
  - Parse URLs
  - Do a mapping between URLs and Java classes that represent resources
  - Generate the different representations of resources
  - etc.

- But of course, there are frameworks that do exactly that for us.
- It is true for nearly every platform and language, including Java.
- There is even a JSR for that: JAX-RS (JSR 311).
  - Oracle provides the reference implementation, in the Jersey project (open source).



Let's look at it in practice. How do we implement a REST API with **Java EE**?

- Create a new "**Web Application**" maven project with Netbeans.



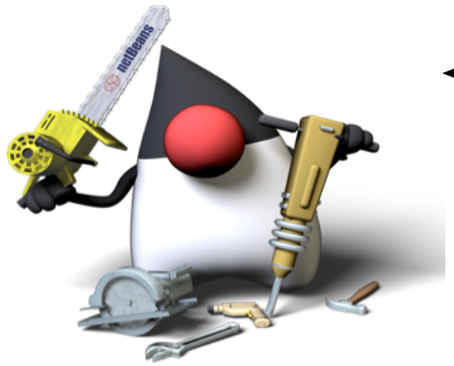
Configuration code for the REST API layer.

This is the **core REST stuff**. It uses Inversion of Control to route/map HTTP requests onto service methods. It handles content negotiation in a declarative way. It relies on JAXB for serialization. Note that even if is logically a presentation-tier component, it is also a session bean so that we can inject EJB references.

This is something new: **Data Transfer Objects** are recommended for various reasons, even if they require extra work.

This would typically be a JPA entity. Since we don't use a DB, it is a simple **POJO**. Notice that we have a salary attribute here.

You should be familiar with these. To keep things simple in this example, we have implemented a singleton, with an in-memory data store. So, we have **merged the business and DAO layers**.

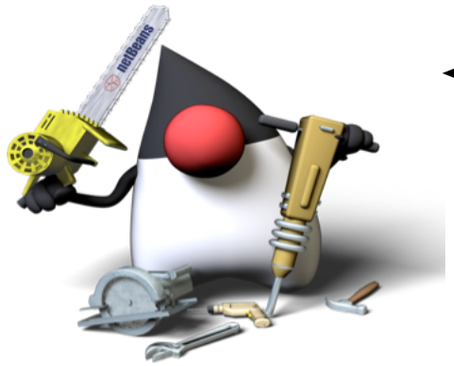


As usual, we have model classes in our architecture.

```
package ch.heigvd.amt.demo.rest.model;

public class Employee {

  private long id;
  private String firstName;
  private String lastName;
  private String email;
  private double salary;

  public Employee() {
  }

  public Employee(long id, String firstName, String lastName, String email, double salary) {
    this.id = id;
    this.firstName = firstName;
    this.lastName = lastName;
    this.email = email;
    this.salary = salary;
  }

  public long getId() {
    return id;
  }

  public void setId(long id) {
    this.id = id;
  }

  ... other getter and setters

}
```

Don't forget to add a **no-arg constructor** if you add constructor with arguments (remember what we said about Java reflection and frameworks in a previous lecture...)

That is an example for a **sensitive attribute**, which we do not want to expose to everyone. Keeping control on the exposed data is one reason why we recommend to use **DTOs** in addition to the model classes.



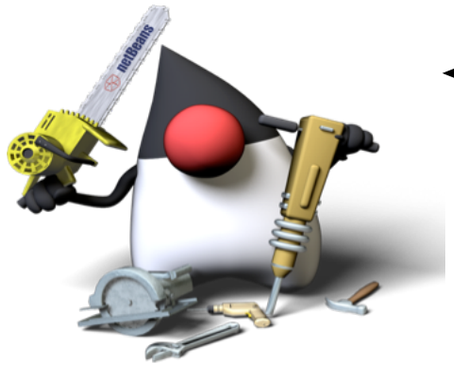
We now introduce Data Transfer Objects (DTOs), which are POJOs.

```
package ch.heigvd.amt.demo.rest.dto;

public class EmployeeDTO {

  private long id;
  private String firstName;
  private String lastName;
  private String email;

  public long getId() {
    return id;
  }

  public void setId(long id) {
    this.id = id;
  }

  ... other getter and setters

}
```

We have the same attributes as in the model class, except for the salary attribute.

Note that for one model class, we could have several DTOs. Imagine of a “PublicEmployeeDTO” and a “FullEmployeeDTO” (which would contain the salary attribute). Depending on the service interface, we would use one or the other.

Also, in large applications, we often have a difference in the granularity between the DTO and the model layers. We could have a single EmployeeDTO class that would map to several model classes (Employee class, Address class, Department class, etc.). This is important, because for remote interfaces, we want to limit the number of request/replies but on the server side, we like to have a fine-grained object oriented model.



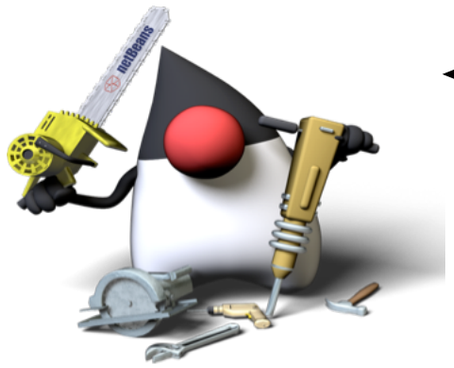
To keep things simple, let's create a flat business+DAO+store layer...

```
package ch.heigvd.amt.demo.rest.services;

@Singleton
public class EmployeesManager implements EmployeesManagerLocal {

  private Map<Long, Employee> data = new HashMap<>();
  public EmployeesManager() {
    data.put(1L, new Employee(1, "Olivier", "Liechti", "olivier.liechti@heig-vd.ch", 128));
    data.put(2L, new Employee(2, "Yannick", "Iseli", "yannick.iseli@heig-vd.ch", 256));
    data.put(3L, new Employee(3, "Homer", "Simpson", "homer.simpson@heig-vd.ch", 512));
  }

  public Employee findEmployeeById(long id) {
    Employee testEmployee = data.get(id);
    return testEmployee;
  }

  public List<Employee> findAllEmployees() {
    return new ArrayList(data.values());
  }

  public long createEmployee(Employee employee) {
    employee.setId(data.size()+1);
    data.put(employee.getId(), employee);
    return employee.getId();
  }

  public void updateEmployee(Employee employee) {
    data.put(employee.getId(), employee);
  }

  public void deleteEmployee(long id) {
    data.remove(id);
  }
}
```

Since this is our in-memory data store, we must have only one!

That's not a real implementation, but our focus this week is on the REST layer, not on the data layer.

Note that no error handling code has been included in the example, for the sake of brevity.

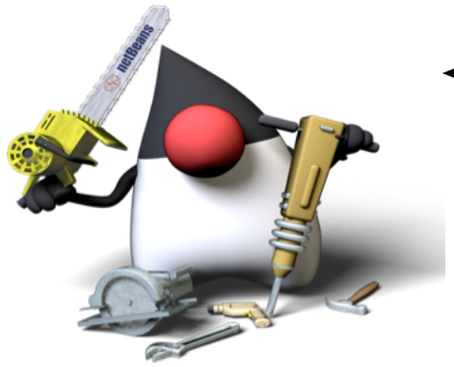


Let's now dig into the real REST layer

```
package ch.heigvd.amt.demo.rest.api;

@Path("employees")
@Stateless
public class EmployeeResource {

  @EJB
  EmployeesManagerLocal employeesManager;

  @Context
  private UriInfo context;

  /**
   * Creates a new instance of EmployeeResource
   */
  public EmployeeResource() {
  }

to be continued over the next slides...
```

Inversion of Control. HTTP requests are processed by an "invisible" front controller provided by JAX-RS. If the incoming request has a URL that starts with "employees", work will be delegated to this class.

I hate this, but it is really helpful. I hate it, because logically, this class belongs to the presentation tier (same tier as the servlets and JSPs). For this reason, it is confusing that this class is a SLSB. However, because it is an EJB, I can inject references to my business and DAO services. This has been added in Java EE 6.

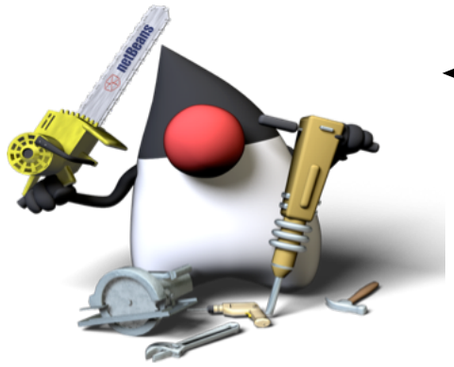


GET the list of all employees

```
@GET
@Produces("application/json")
public List<EmployeeDTO> getAllEmployees() {
  List<Employee> employees = employeesManager.findAllEmployees();
  List<EmployeeDTO> result = new ArrayList<>();

  for (Employee employee : employees) {
    result.add(toDTO(employee));
  }

  return result;
}
```

Second layer of inversion of Control. If the HTTP request has a GET method, then this method is invoked.

Content negotiation and automatic marshalling. If the request has a “Accept” header with a value of “application/json”, then this method is called and JSON will be sent back to the client. Moreover, JAX-RS (with the help of JAXB) is responsible for converting the List of EmployeeDTO to JSON (it could do the same for XML). Isn’t that cool?



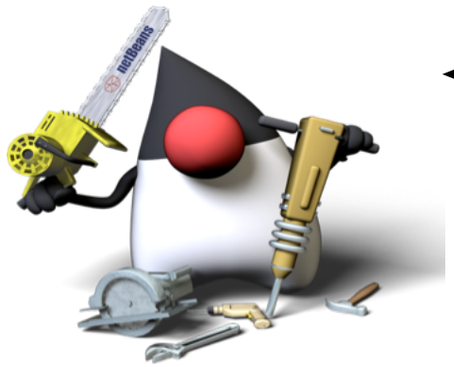
GET the details of one employee

```
@Path("/{id}")
@GET
@Produces("application/json")
public EmployeeDTO getEmployeeDetails(@PathParam("id") long id) {
   Employee employee = employeesManager.findEmployeeById(id);
   return toDTO(employee);
}
```

Retrieval of a variable in the URL path. It the URL in the request is /employees/128, then id = 128.

Second and third layers of inversion of Control. If the URL in the request starts with "employees" and is followed by an "id" element AND if the HTTP request has a GET method, then this method is invoked.

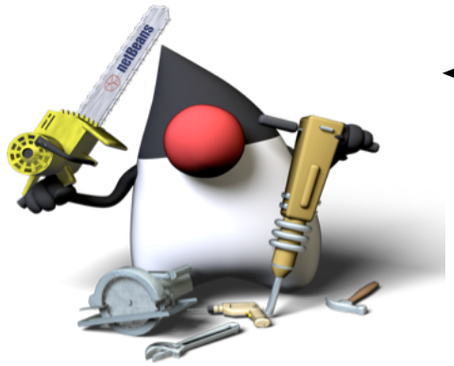
Content negotiation and automatic marshalling. Same as before, but now we only return a single DTO instance.



Create a new employee with a POST

```
@POST
@Consumes("application/json")
public long createEmployee(EmployeeDTO dto) {
   Employee newEmployee = new Employee();
   long id = employeesManager.createEmployee(toEmployee(dto, newEmployee));
   return id;
}
```

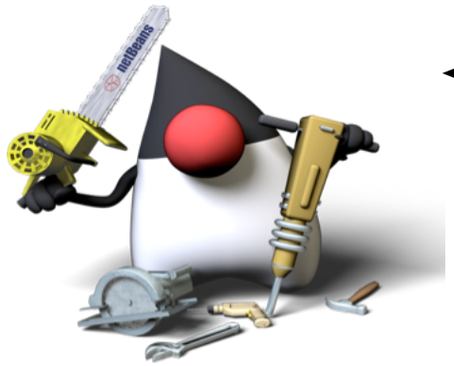
Content negotiation and automatic marshalling. This works in both directions. In this case, we receive (consume) a JSON payload. JAX-RS gives us a ready-to-use dto in the method and takes care of the unmarshalling.



Update an existing employee with a PUT
Delete one with a DELETE

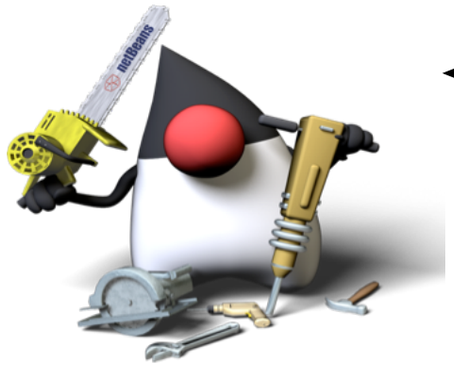
```
@Path("/{id}")
@PUT
@Consumes("application/json")
public void updateEmployee(@PathParam("id") long id, EmployeeDTO dto) {
  Employee existing = employeesManager.findEmployeeById(id);
  employeesManager.updateEmployee(toEmployee(dto, existing));
}
```

```
@Path("/{id}")
@DELETE
public void deleteEmployee(@PathParam("id") long id) {
  employeesManager.deleteEmployee(id);
}
```



We need to convert DTO instances to model instances, and vice versa.

```
private Employee toEmployee(EmployeeDTO dto, Employee original) {
  original.setFirstName(dto.getFirstName());
  original.setLastName(dto.getLastName());
  original.setEmail(dto.getEmail());
  return original;
}

private EmployeeDTO toDTO(Employee employee) {
  EmployeeDTO dto = new EmployeeDTO();
  dto.setId(employee.getId());
  dto.setFirstName(employee.getFirstName());
  dto.setLastName(employee.getLastName());
  dto.setEmail(employee.getEmail());
  return dto;
}
```



Last but not least, we need to configure the JAX-RS framework

Inversion of Control. HTTP requests are processed by an “invisible” front controller provided by JAX-RS. The front-controller registers itself on this URL prefix.

To invoke the methods of our class, we need to use the following URLs: **/DemoRest/api/employees/**

```
package ch.heigvd.amt.demo.rest.api;

import java.util.Set;
import javax.ws.rs.core.Application;

@javax.ws.rs.ApplicationPath("api")
public class ApplicationConfig extends Application {

  @Override
  public Set<Class<?>> getClasses() {
    Set<Class<?>> resources = new java.util.HashSet<>();
    addRestResourceClasses(resources);
    return resources;
  }

  private void addRestResourceClasses(Set<Class<?>> resources) {
    resources.add(ch.heigvd.amt.demo.rest.api.EmployeeResource.class);
  }
}
```