02267: Software Development of Web Services

Week 2

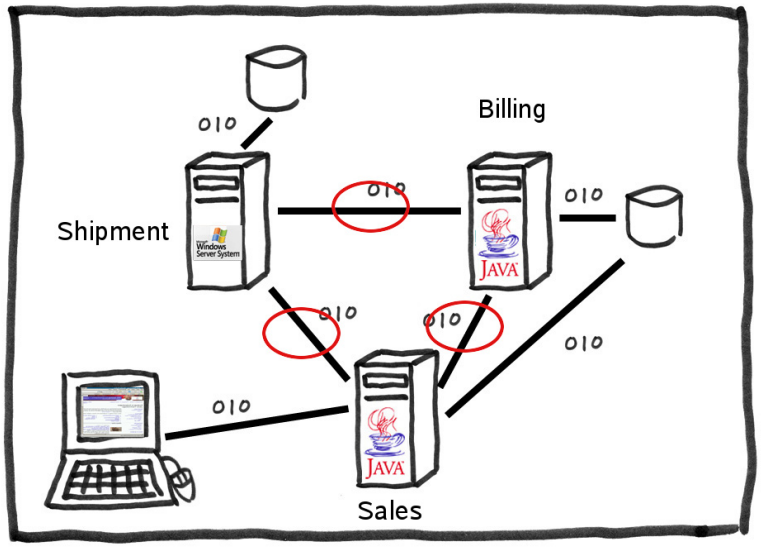
Hubert Baumeister
huba@dtu.dk

Department of Applied Mathematics and Computer Science
Technical University of Denmark

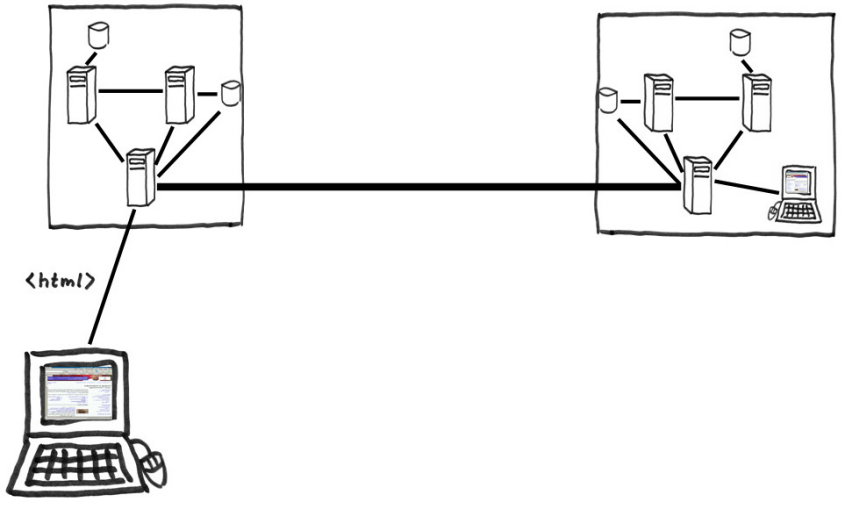
Fall 2015

Recap

Distributed IT



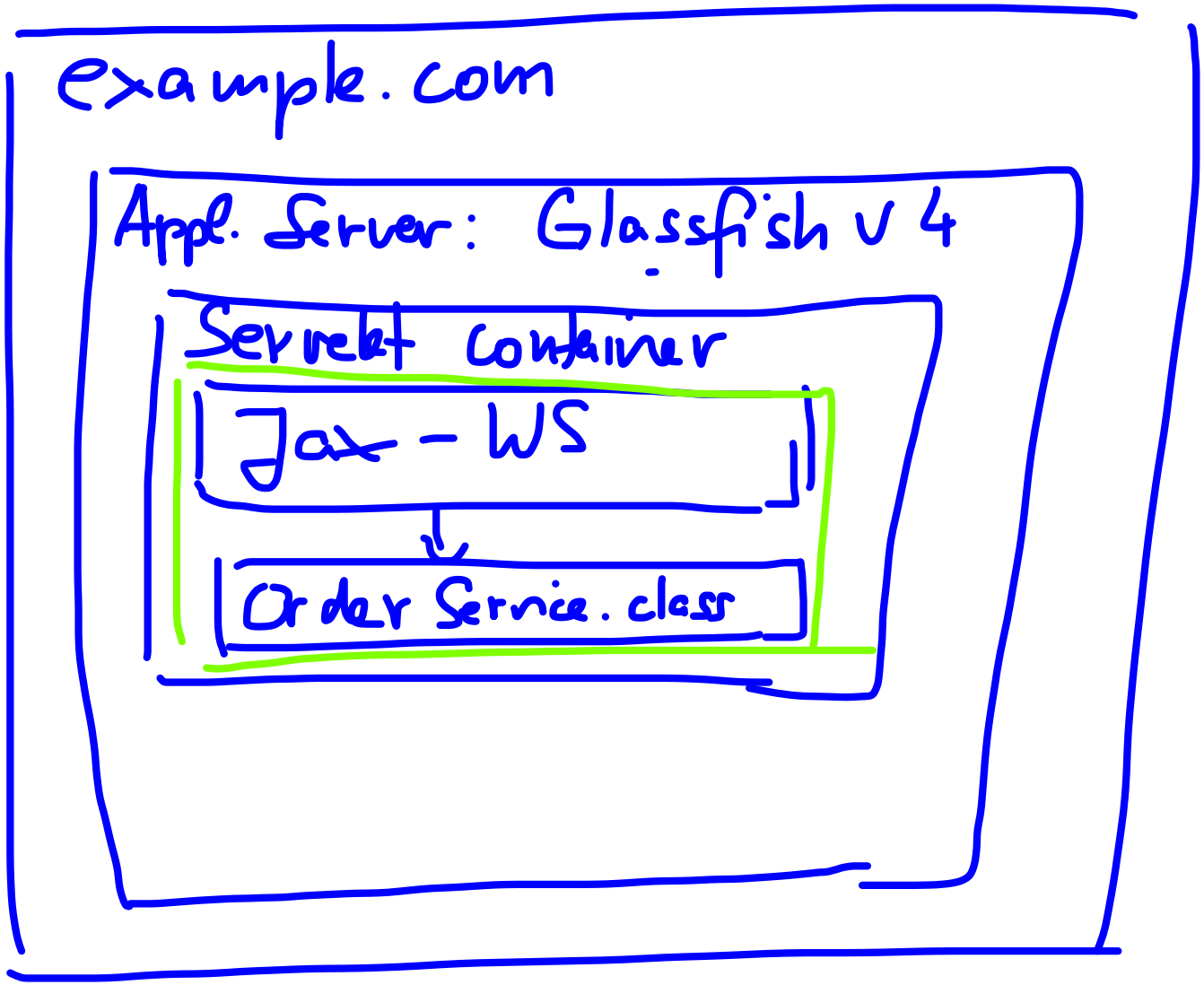
Gobal Computing



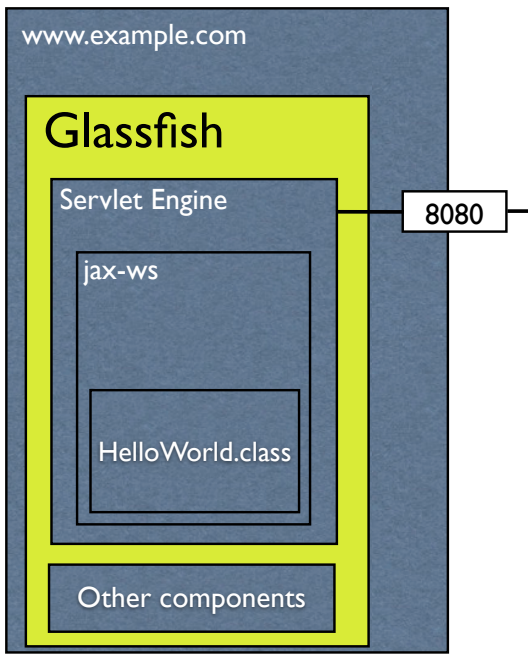
- Service Oriented Architecture (SOA)
- Web Services = SOA + Use of open Web Standards
- 2 types: SOAP and REST based Web services

Contents

Basic architecture



Basic architecture



- Web services are usually implemented as a Web application
  - Use HTTP to tunnel firewalls
  - → Thus we need a Web server for the deployment of Web services
  - → Our choice is GlassFish as Web server
- JAX-WS translates SOAP messages to method calls in the Echo class
- Note the use of port 8080 instead of port 80 (the standard for HTTP)
  - only the administrator can listen on port 80

The Order Web Service

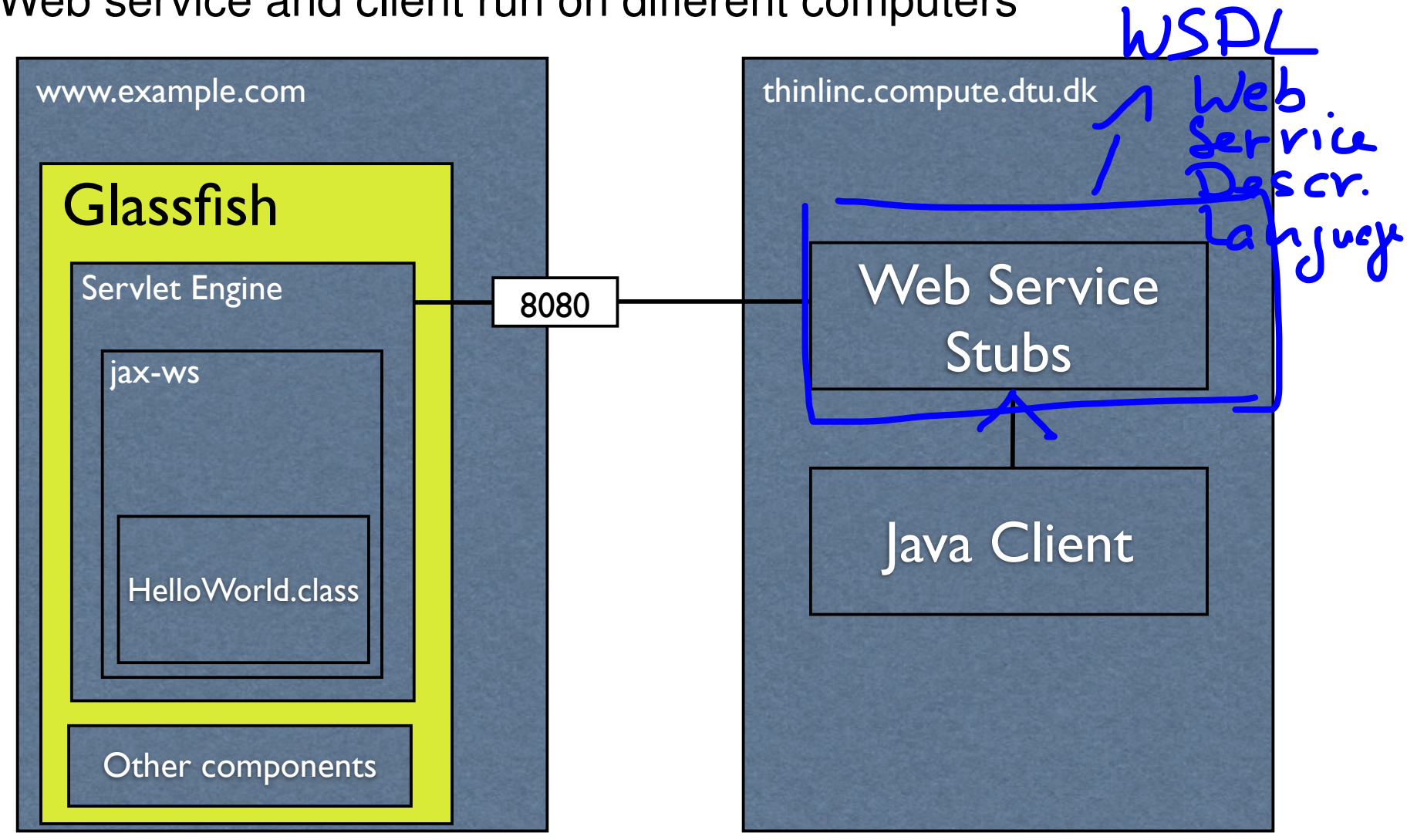
- Create a Java class and use the annotation @javax.jws.WebService to mark it as an implementation class for Web services
- By default public methods of the class are offered as Web services

```
package dk.dtu.ws;

@javax.jws.WebService
public class OrderService {
    public String receiveOrder(String customer,
                               String product,
                               int amount) {
        return String.format("Confirmed %d %s by %s",
                             amount,product,customer);
    }
}
```

Annotation documentation: `http://docs.oracle.com/cd/E13222_01/wls/docs92/webserv/annotations.html`

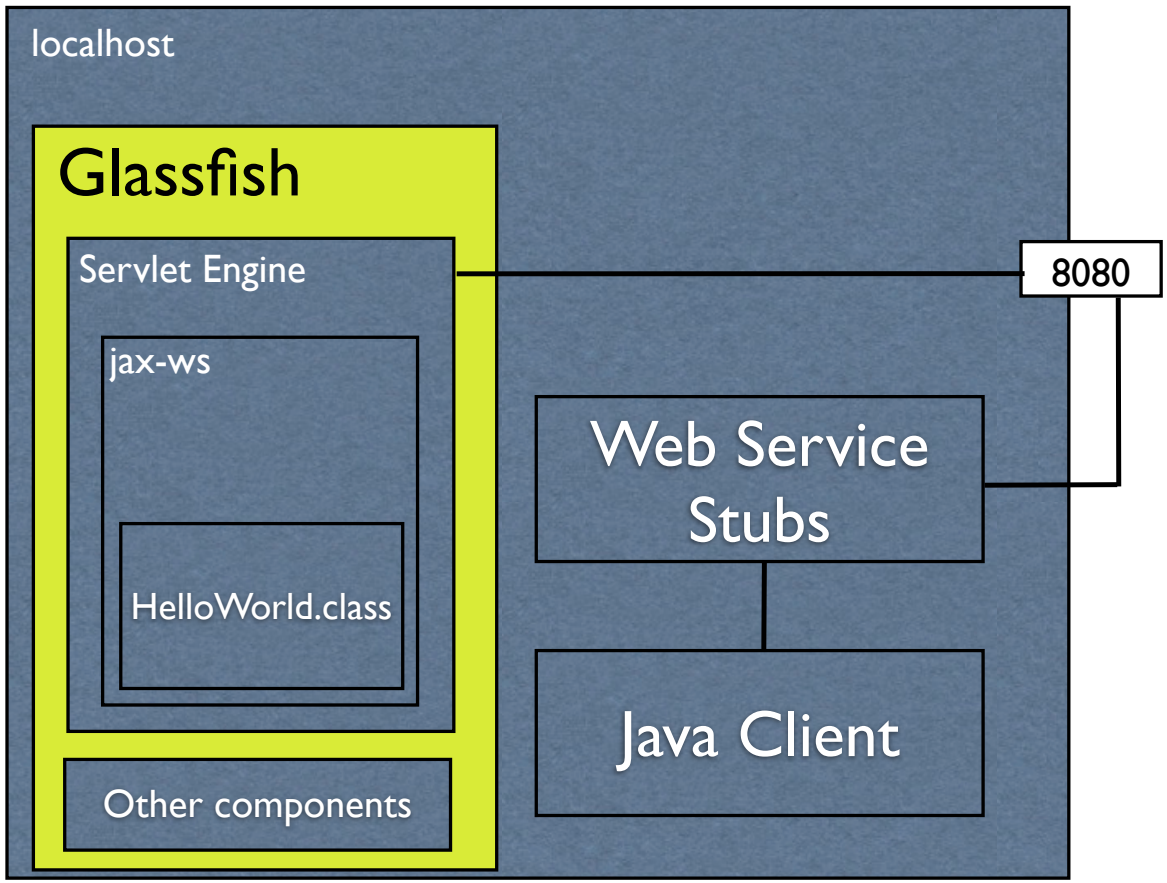
Calling Web services

Web service and client run on different computers



Calling Web services (localhost)

Web service and client run on the same computer



Writing a Web service client

- A Web service client is any program using a Web service, e.g. a Java application
- How to access the Web services
  1. send a HTTP POST request with the request as a SOAP message to the server
  2. better: use some Java stubs doing that for you

  → The program `wsimport` generates these stubs (called directly by the NetBeans wizard)
- However, wsimport needs a description of the Web services offered by the Web server
  - → use the WSDL (Web Service Description Language) document generated by the Web server
    - This URL of this document can be obtained by looking at the Web services section at `http://localhost:4848`

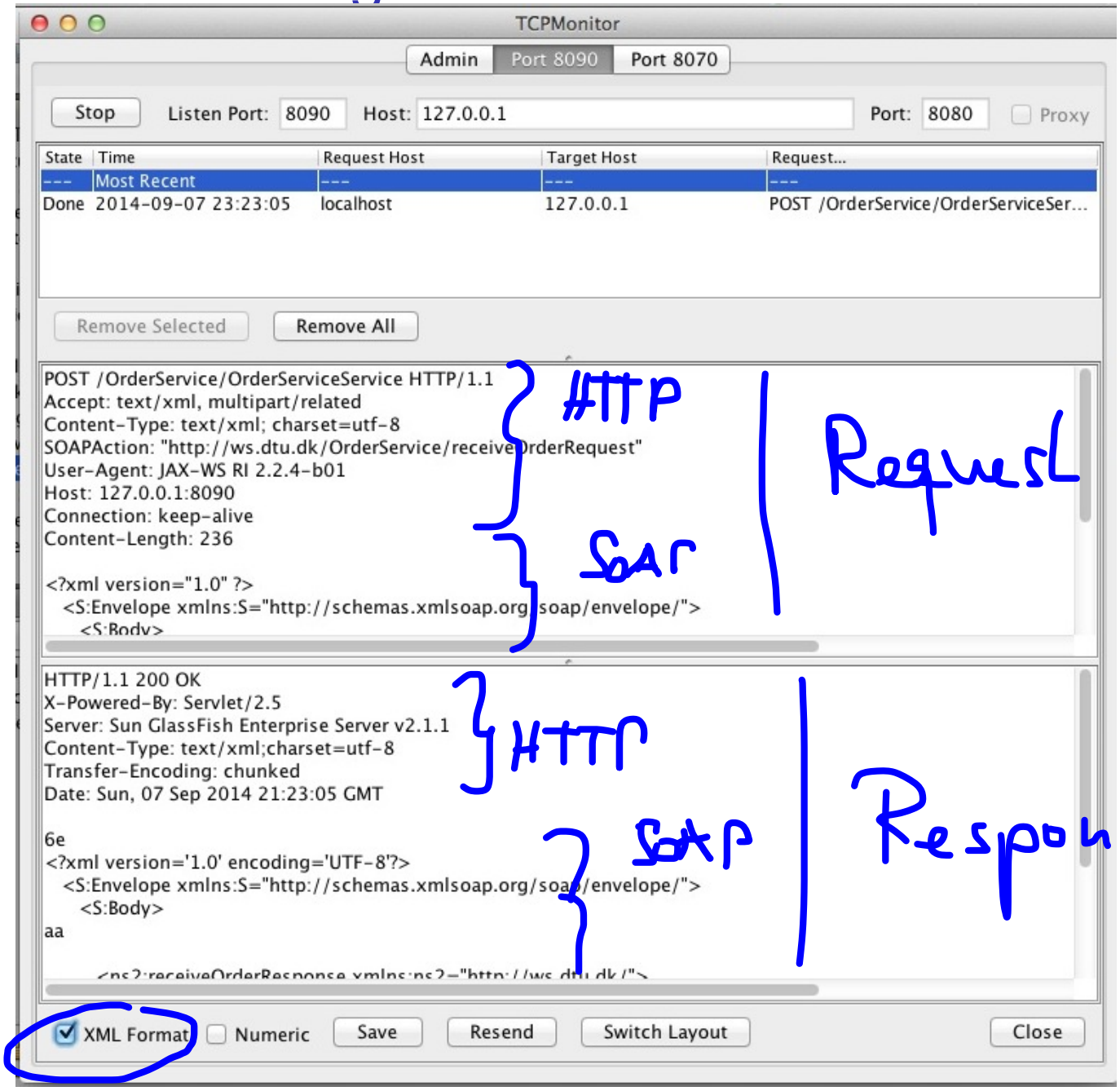
OrderService Client

```
package dk.dtu.ws;
// Client stub factory
import dk.dtu.ws.OrderServiceService;
// Service interface
import dk.dtu.ws.OrderService;

// Annotation for test methods
import org.junit.Test;
// Assertions like assertEquals
import static org.junit.Assert.*;

public class OrderServiceClient {

 OrderService orderServiceStub = null;

 public OrderServiceClient() {
   OrderServiceService factory = new OrderServiceService();
   orderServiceStub = factory.getOrderServicePort();
 }

 @Test
 public void test() {
   String r = orderServiceStub.receiveOrder("Hubert","pencil",10);
   assertEquals("Confirmed 10 pencil by Hubert",r);
 }
}
```

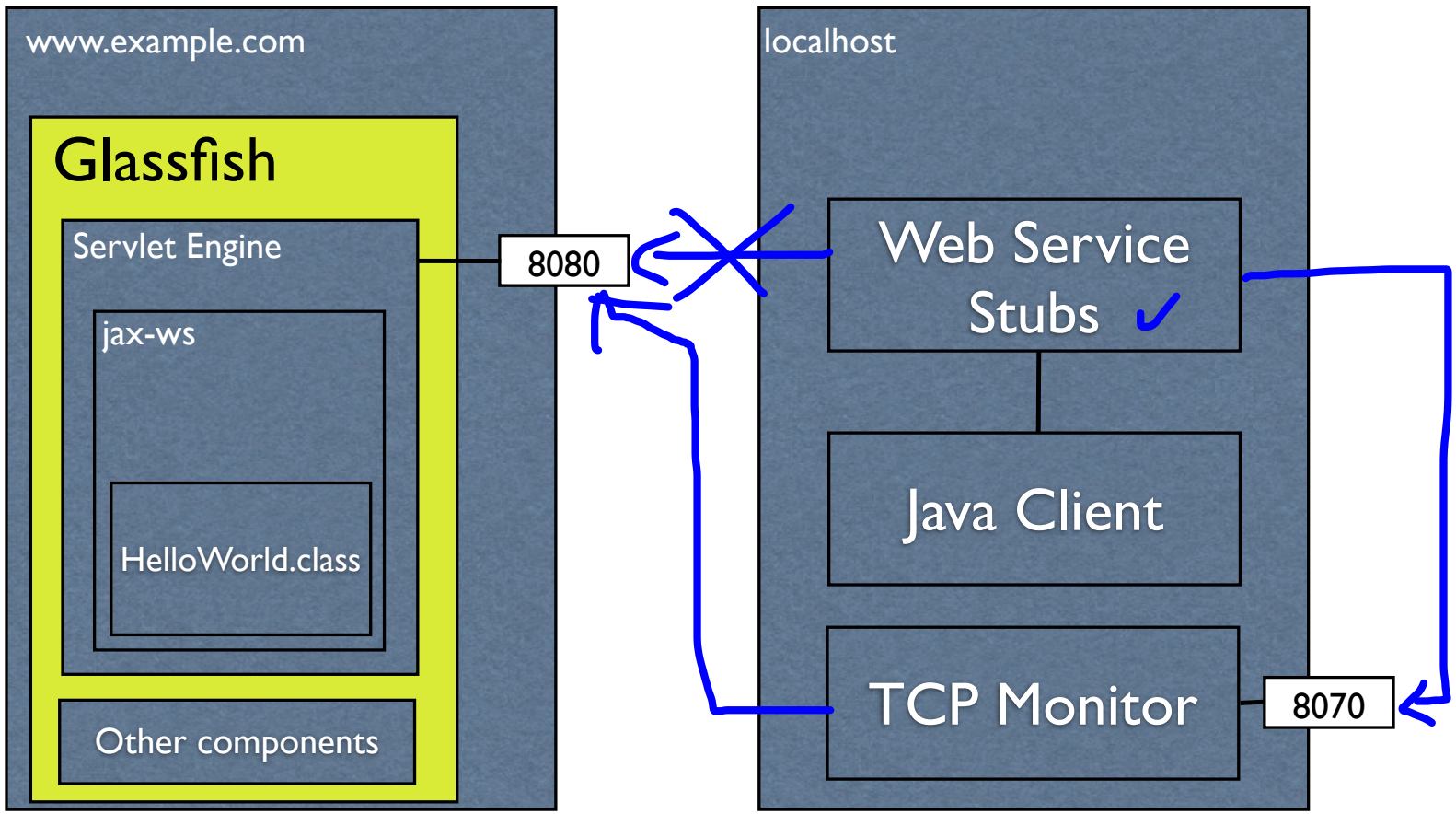
wsimport

Contents

TCPMon: Monitoring Web service communication



Monitoring the traffic between the Web server and the client



Monitoring Web services

- A TCP monitor forwards all TCP packets from one port (8070) to another (e.g. 80 or 8080) at another host (can be localhost)
- Note, client now needs to connect to TCP monitor port 8070
  - → a new Web service client stub needs to be generated:
  - a) Remove the old stub
  - b) Download the WSDL of the Web service
  - c) In the WSDL exchange

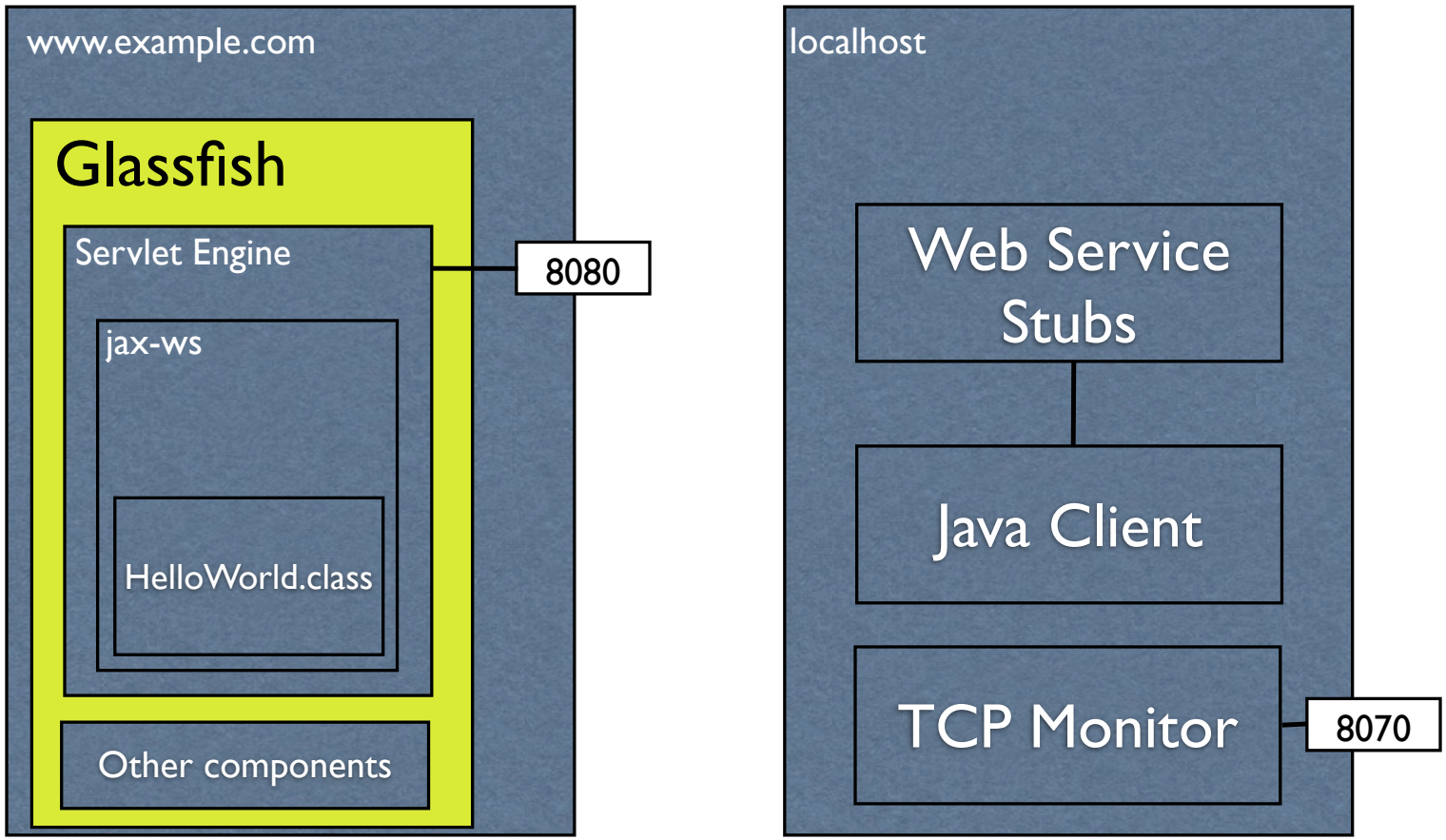
```
<soap:address
 location="http://serv.ex.com/OrderService/OrderServiceService"/>
```

by

```
<soap:address
 location="http://localhost:8070/OrderService/OrderServiceService"/>
```

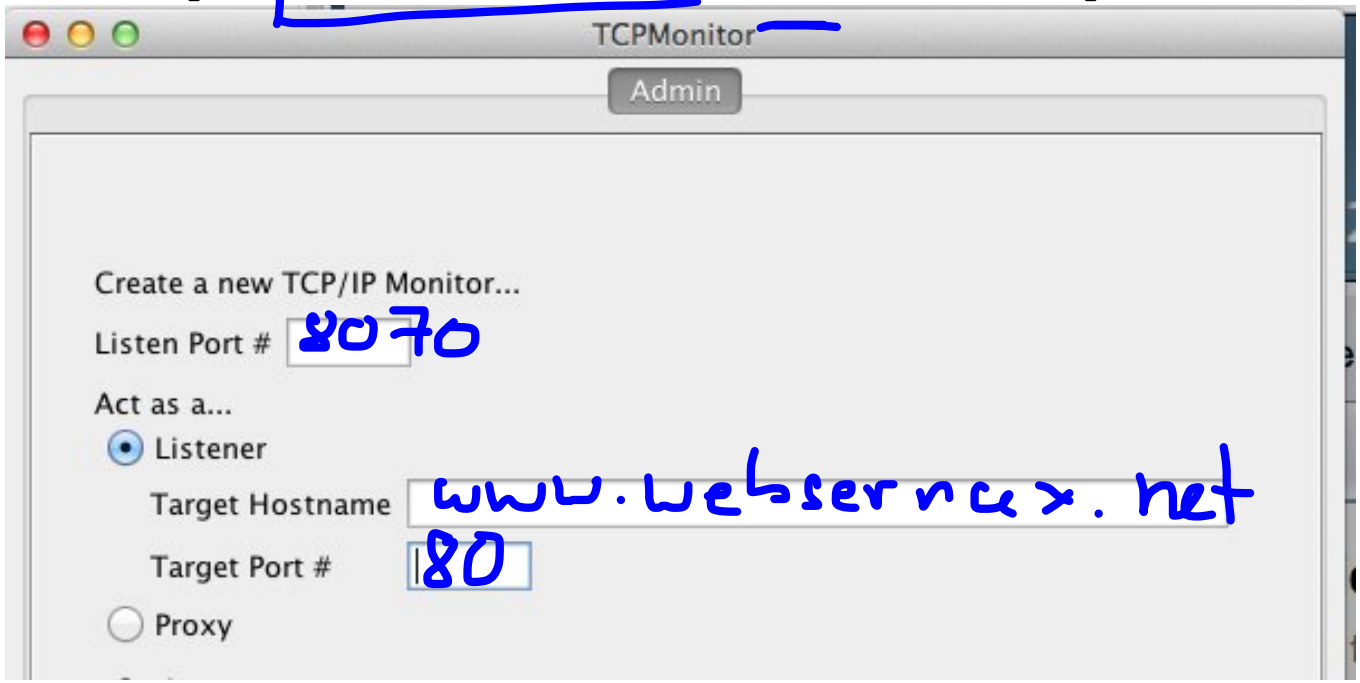
  - d) Regenerate the client stubs from the changed WSDL

Configuring tcpmon (I)



Configuring tcpmon (II)

Service URL: `http://www.webservicex.net/ConvertTemperature.asmx`



In WSDL file for client generation

```
<soap:address
location="http://localhost:8070/ConvertTemperature.asmx"

/>
```

Contents

eXtended Markup Language (XML)

- XML is one of the basic technologies of Web services
- XML is used to describe data as well as Web services, properties of Web service, programs, protocols, ...
  - SOAP, WSDL, WS-Policy, BPEL, WS-Coordination, ...

Markup languages

Special markers in the text are used to describe the structure of the text

Example: HTML

HTML markup

```
<h4>News</h4>
<ul>
   <li>Added instructions on <a href="index1.html">how
   <li>Added instructions to how to install the softwar
</ul>
<p>

<h4>Time and Location</h4>
```

Display in the browser

News

- Added instructions on how to use the TCP monitor
- Added instructions to how to install the software an and Web service client (31.08.12)

Time and Location

History of XML

- Allow description of single markup languages
  - E.g. XHTML for Web pages
- SGML (since 1986 ISO standard); preceeds the Web
- HTML (1989; inspired by SGML; however, is not XML conform)
- XML (first version in 1997); Simplified version of SGML
  - Version 1.0 fourth edition (`http://www.w3.org/TR/REC-xml/`) (Version 1.1 is in principle the same as v1.0 but has less restrictions regarding character representations.)
- Uses
  - Electronic Publishing
  - Representing structured data in a standard way
  - Programming language representation
    - BPEL, Ant scripts, ...
  - ...

Structure of XML documents

Recipe

```
<?xml version="1.0" encoding="UTF-8"?>
<recipe name="bread" prep_time="5 mins" cook_time="3 hours">
  <title>Basic bread</title>
  <ingredient amount="3" unit="cups">Flour</ingredient>
  <ingredient amount="0.25" unit="ounce">Yeast</ingredient>
  <ingredient amount="1.5" unit="cups" state="warm">Water</ingredient>
  <ingredient amount="1" unit="teaspoon">Salt</ingredient>
  <instructions>
    <step>Mix ingredients, and knead thoroughly.</step>
    <step>Leave for one hour in warm room.</step>
    <step>Knead again, and then bake in the oven.</step>
  </instructions>
</recipe>
```

Structure of XML documents I

- Declaration (optional)

  ```
  <?xml version="1.0" encoding="UTF-8"?>
  ```

- Elements
  - Start tag: `<step>`, end tag: `</step>`
  - With contents (can be text and / or other elements)

    ```
    <name>with <strong>contents</strong></name>
    ```

  - Without contents (but can have attributes)

    ```
    <step/>
    ```

  - XML documents must have one root element
  - Elements have to be nested properly

Structure of XML documents II

- Elements can have attributes

```
<recipe name="bread" prep_time="5 mins">}
```

- Comment

```
<!-- Some comment in XML files -->
```

- Entities
  - Represents a named body of data, usually text, such as an unusual character
  - E.g: < and > for < and > (needed as < and > are reserved for elements)

Well-formed and valid XML documents

- Well-formed XML documents
  - Obeys the structure described on the previous slides; For example, the following is not well-formed

    ```
    <xyz><abc></xyz></abc>
    ```
- Valid XML documents (e.g. XML documents describing Recipes, SOAP, WSDL, XHTML . . .)
  - Restricts the use of elements / attribues etc. in XML documents. For example, the following is not a valid XHTML document

    ```
    <html><html></html></html>
    ```
- Valid XML documents are described (among others) using
  - DTD (Document Type Definition)
    - Is not an XML document
    - Not as expressive as XML Schema
  - XML Schema
    - Is again an XML document (the structure of which is again described by an XML Schema document :-)
    - More expressive as DTD

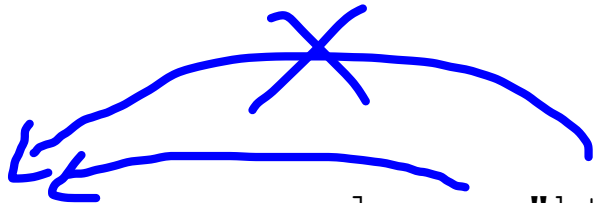
XML Namespaces: Problem

```
<company>
   <name>IBM</name>
   <address>...</address>
</company>

<person>
   <name>
      <first>Helge</first>
      <last>Helmersen</last>
   </name>
   <address>...</address>
</person>
```

XML Namespaces: Problem

```
<company>
   <name>IBM</name>
   <ceo>
      <person>
         <name>
            <first>Helge</first>
            <last>Helmersen</last>
         </name>
         <address>...</address>
      </person>
   </ceo>
   <address>...</address>
</company>
```

Namespaces to the rescue

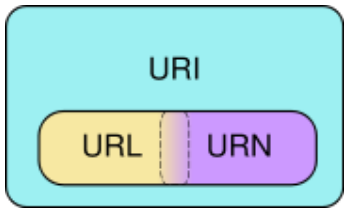


```
<c:company xmlns:c="http://dk.dtu.ws.company">
   <c:name>IBM</c:name>
   <c:ceo>
      <person xmlns="http://dk.dtu.ws.person">
         <name>
            <first>Helge</first>
            <last>Helmersen</last>
         </name>
         <b:address xmlns:b="http://dk.dtu.ws.address">
            ...
         </b:address>
      </person>
   </c:ceo>
   <a:address xmlns:a="http://dk.dtu.ws.address">
      ...
   </a:address>
</c:company>
```

XML Namespaces

- namespace declaration as an attribute with a start tag: `xmlns:nsprefix="URI"`
  - The URI globally identifies the namespace
  - The prefix `nsprefix` locally identifies the namespace within one XML document and can be freely chosen (even empty)
  - Tags and attributes belonging to that namespace get the prefix `nsprefix`, e.g., `c:company` or `env:Envelope`
  - The start tag can already use the local name of the name space
    `<env:Envelop xmlns:env="..." ...>`
- Namespaces may be accompanied by a XML Schema definition to define what is allowed in that namespace and what not
- More information
  `http://www.w3.org/TR/REC-xml-names/`

URI vs URL vs URN



- Identification: By name and/or by location
- URI: Universal Resource Identifier
- URL: Universal Resource Locator
  - Locates a resource
- URN: Universal Resource Name
  - Names a resource
- Possible: a name is equally a locator
  - `http://schemas.xmlsoap.org/soap/envelope/` names the SOAP 1.0 namespace
  - but also locates the XML Schema definition of SOAP 1.0
- General form
  - protocol:protocol specific
  - protocol can be http, ftp, urn, mailto, . . .

Contents

SOAP

- Defines SOAP-based Web services
- Message exchange standard based on XML
- Can be used over different protocols (HTTP, SMTP, ...)
- SOAP 1.1 (2000) Namespace URI:
  `http://schemas.xmlsoap.org/soap/envelope/`
- SOAP 1.2: Namespace URI:
  `http://www.w3.org/2003/05/soap-envelope`
- Standard documents: `http://www.w3.org/TR/soap/`

Structure of a SOAP message

```
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Header>
     <To xmlns="http://www.w3.org/2005/08/addressing">
        http://127.0.0.1:8070/Calculator/CalculatorWSService</To>
     <ReplyTo xmlns="http://www.w3.org/2005/08/addressing">
       <Address>
          http://www.w3.org/2005/08/addressing/anonymous
       </Address>
     </ReplyTo>
     <MessageID xmlns="http://www.w3.org/2005/08/addressing">
        uuid:1410abbe-58db-44eb-bb02-a5da4992d473
     </MessageID>
   </S:Header>
   <S:Body>
     <ns2:add xmlns:ns2="http://calculator.ws.imm/">
        <arg0>3</arg0>
        <arg1>4</arg1>
     </ns2:add>
   </S:Body>
</S:Envelope>
```

optional

mandatory

appl. specific

initiate the WS service

Structure of a SOAP Message

`<Envelope encodingStyle="uri">`

- `<Header encodingStyle="uri">`?
  - EncodingStyle
    - E.g. `http://www.w3.org/2003/05/soap-encoding`
    - Can be also in a child element of `<Header>`, or `<Body>`
  - `<ns1:... >` $*$ (SOAP header block; application specific)
- `<Body>`
  - `<ns2:... >` $*$ (Contents of a SOAP message; application specific)
  - or `<Fault>`

Response

```
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
 <S:Header>
  <To xmlns="http://www.w3.org/2005/08/addressing">
    http://www.w3.org/2005/08/addressing/anonymous</To>
  <RelatesTo xmlns="http://www.w3.org/2005/08/addressing">
     uuid:1410abbe-58db-44eb-bb02-a5da4992d473
  </RelatesTo>
 </S:Header>
 <S:Body>
  <ns2:addResponse xmlns:ns2="http://calculator.ws.imm/">
   <return>7</return>
  </ns2:addResponse>
 </S:Body>
</S:Envelope>
```

Contents

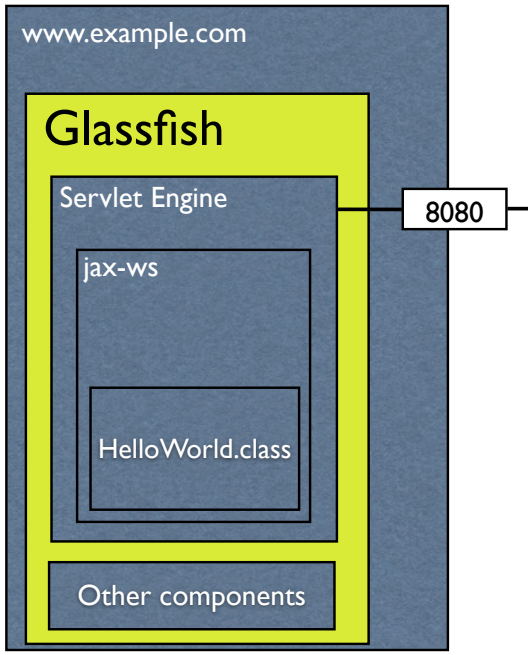
WS Architecture

- SOAP messages require a transport protocol for transmission (e.g. HTTP, SMTP, ...)

→ Commonly used HTTP

  - SOAP messages are transported by the HTTP protocol via POST requests

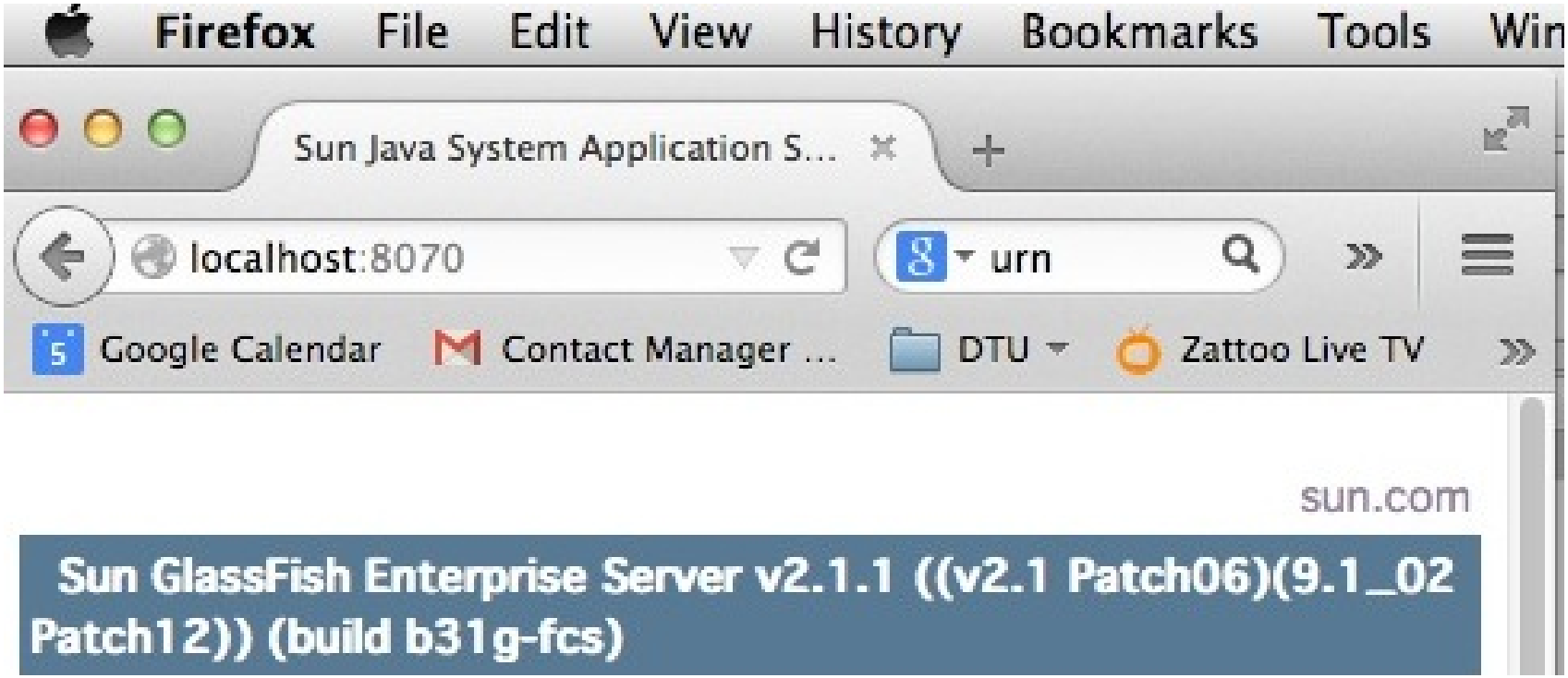
- Distinction between message layer (SOAP) and transport layer (HTTP, SMTP, ...)



HTTP

- HTTP = Hypertext Transfer Protocol
- The basic protocol for the Web
- Is used to retrieve Web sites from Web servers
  - Static Web pages
  - On the fly generated Web pages: CGI bins / Servlets in Java / ...
- HTTP/1.1 is defined in RFC 2616 (`http://www.w3.org/Protocols/`)

HTTP Get Request




```
GET / HTTP/1.1
Host: 127.0.0.1:8070
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.1.1) Gecko/
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

HTTP Answer

```
HTTP/1.1 200 OK
X-Powered-By: Servlet/2.5
Server: Sun GlassFish Enterprise Server v2.1
ETag: W/"4827-1251914047000"
Last-Modified: Wed, 02 Sep 2009 17:54:07 GMT
Content-Type: text/html
Content-Length: 4827
Date: Fri, 04 Sep 2009 00:04:56 GMT

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
<html lang="en">
<head>
...
</body>
</html>
```

HTTP Access Methods

- Most common access methods:
  - GET: Get information, such as a Web page from the server
  - POST: Submit information (e.g. forms) to the Web server and wait for an answer
  - PUT: Replace the resource identified by the request URI with the enclosed information
  - DELETE: Delete the resource identified by the request URI
  - ...
- POST: used by Web services to send a SOAP message
- GET, POST, PUT, DELETE
  - $\rightarrow$ REST (Representational State Transfer) architectural principle
  - used with RESTful Web services

SOAP over HTTP

- HTTP requests must be POST request
- content-type must be text/xml
- HTTP header must have a SOAPAction field SOAPAction: ”URI”. This identifies the action/service and allows firewalls to detect and handle SOAP messages. Ex:
  - SOAPAction ”http://electrocommerce.org/abc#MyMessage”
  - SOAPAction ”axis/EchoString.jws”
  - SOAPAction ”” means that the URI of the HTTP request is the SOAPAction URI

SOAP over HTTP example: Web Service Request

```
POST /Echo/EchoService HTTP/1.1
SOAPAction: ""
Accept: text/xml, multipart/related, text/html, image/jpg, image/jpeg,
Content-Type: text/xml; charset=utf-8
User-Agent: Java/1.6.0_07
Host: 127.0.0.1:8070
Connection: keep-alive
Content-Length: 182

<?xml version="1.0" ?>
   <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
      <S:Body>
         <ns2:echo xmlns:ns2="http://week01/">
            <arg0>Hello</arg0>
         </ns2:echo>
      </S:Body>
   </S:Envelope>
```

SOAP over HTTP example: Answer

```
HTTP/1.1 200 OK
X-Powered-By: Servlet/2.5
Server: Sun GlassFish Enterprise Server v2.1
Content-Type: text/xml;charset="utf-8"
Transfer-Encoding: chunked
Date: Fri, 04 Sep 2009 00:11:03 GMT
ca
<?xml version="1.0" ?>
   <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
      <S:Body>
         <ns2:echoResponse xmlns:ns2="http://week01/">
            <return>Hello</return>
         </ns2:echoResponse>
      </S:Body>
   </S:Envelope>
0
```

Summary

- Web service architecture
- Monitoring Web services with TCPmon
- XML + XML Namespaces
- Basic SOAP
- HTTP + SOAP