11.10 Web Services (Overview)

- History: RPC (Remote Procedure Call)
  - call a specific procedure at a specific server
    (client stub→marshalling→message→unmarshalling→ server stub→ server).
- History: OMG Standard (Object Management Group) CORBA (1989, “Common Object Request Broker Architecture”; cf. Slides 38 ff.):
  - Middleware, usually applied in an Intranet,
  - central ORB bus where services can connect,
  - service registry (predecessor of WSDL and UDDI ideas),
  - description of service interfaces in object-oriented style
    (IDL - interface description language, similar to C++ declarations),
  - exchanging objects between services via OIF (Object Interchange Format),

  ⇒ RPC abstraction (call abstract functionality) by the ORB as a broker.
- XML-RPC and SOAP+WSDL+UDDI are XML-based variants of RPC+Corba.
- SOA (“Service-Oriented Architecture”).

HTTP: Hypertext Transfer Protocol (Overview)

- HTTP 0.9 (1991), HTTP 1.0 (1995), HTTP 1.1 (1996).
- Application Layer Protocol [OSI Level 7], based on a (reliable) transport protocol (usually TCP “Transmission Control Protocol” [ISI/OSI Level 4] that belongs to the “Internet Protocol Suite” (IP))
  [see Telematics lecture].
- Request-Response Protocol: open connection, send something, receive response (both can be streamed), close connection.
- well-known from Web browsing and HTML:
  send (HTTP GET) URL, get URL (=resource) contents
  ⇒ this is already a (very basic) Web Service
  also: send HTTP POST URL+Data (Web Forms) get answer
  ⇒ this is also a (still basic) Web Service; “Hidden Web”
- common protocol used for communication with and between Web Services ...

Infrastructure Architecture

Web Server

- hosts different things; amongst them
  - "simple" HTML pages, binaries (pdfs, pictures, movies, ...)
  - Web Services, i.e. software artifacts that implement some functionality.
- Example: Apache Web Server.
- not the topic of this lecture ($\rightarrow$ technical infrastructure).

(Java) Servlet

- a piece of software that should be made available as a Web Service,
- implements the methods of the Servlet interface
  (Java: `javax.servlet.http.Servlet`, subclasses `GenericServlet`, `HttpServlet`)

Web (Service|Servlet) Container

- a piece of software that extends a Web Server with infrastructure to provide the runtime environment to run servlets as Web Services,
- hosts one or more Web Services that extend the container's base URL

Web Servlet Container [incorrect: Web Service Container]

- Servlets are the pieces of software that are used to *provide* services.
- The servlets' code must be accessible to the Web Servlet Container, usually located in a specific directory,
- WSC controls the lifecycle of the servlets: (`init()`, `destroy()`)
- maps the incoming communication from ports via the URLs to the appropriate servlet invocation.
  Container: method `service(`*`httpContents`*`)`, mapped to Servlets' `doGet(`*`httpContents`*`)` `doPost(`*`httpContents`*`)`, (`doPut(`*`httpContents`*`)`), (`doDelete(`*`httpContents`*`)`).
- Example: Apache tomcat.
- standalone tomcat: one port (default 8080), one base URL;
- tomcat might be run in a Web Server (Apache), then, multiple base URLs can be mapped to the same tomcat.
- URL tails do not necessary belong to the same/different Servlets (see next slides)!

$\Rightarrow$ URL tails are just abstract names
(even the internal organization/implementation might change over time)

ABSTRACTION LEVELS

Goal: abstract from internal software/programming structure of the projects against the externally visible URLs.

- a Web Service Container contains several "projects" (eclipse terminology) or "applications":
  - from the programmer's view, a "project" is an (e.g., eclipse) project,
    as a package it is a single .war file,
    at the end, it is a subdirectory in the container.
    Each project has an (internal) name (its directory name in the container), e.g. `xquery-demo` or `servletdemo`.
- Each project consists of one or more servlets:
  - each servlet has an (internal) name (relative to its directory name in the container), e.g. the `servletdemo` project contains three different servlets (just due to its programming as a "silly example", nothing about efficiency)
    (nobody from the outside will see what are the actual names of these servlets)
  - each servlet's code is a class that extends javax.servlet.http.HttpServlet;

Abstraction Levels: URL mapping

HTTP connections received by the servlet container are internally forwarded to the servlets.

- the Web Service Container has a base url;
  `http://www.semwebtech.org`.
  (actually, this is the base URL of an Apache that maps most things to a tomcat)
- Service URLs: `http://www.semwebtech.org/xquery-demo`,
  `http://www.semwebtech.org/servletdemo`,
  `http://www.semwebtech.org/services/2016/xml2sql` etc.
- the Web Service Container maps *relative paths* to projects (by tomcat's server.xml):
  `/xquery-demo` to `xquery-demo`, and `/servletdemo` to `servletdemo`, and
  `/services/2016/xml2sql` to `xmlconverter`.
- each project's configuration (in its web.xml) maps URL path tails to servlet ids, and servlet ids to servlet classes, e.g. for the servletdemo project
  `/sum` to `sum-servlet` to `org.semwebtech.servletdemo.SumServlet`,
  `/format`, `/all` and `/reset` to `format-servlet` to `org.s.s.FormatServlet`,
  `/makecalls` to `makecalls-servlet` to `org.s.s.MakeCallsServlet`, and `index.html` is the front page served for "/".

⇒ internal software organization independent from externally visible URLs

TOMCAT BASIC INSTALLATION

- See course Web page for detailed instructions with servlet examples.
- Web Servlet Container with simple Web Server: Download and install Apache Tomcat
  - can *optionally, but not necessarily* be combined with the Apache Webserver,
  - can be installed in the CIP Pool
- set environment variable (catalina is tomcat's Web Service Container)

```
export CATALINA_HOME=~/apache-tomcat-x.x.x
```

- configure server: edit

```
$CATALINA_HOME/conf/server.xml:
<!-- Define a non-SSL HTTP/1.1 Connector on port 8080 -->
<Connector port="8080" .../>
```

- start/stop tomcat:

```
$CATALINA_HOME/bin/startup.sh
$CATALINA_HOME/bin/shutdown.sh
```

- logging goes to

```
$CATALINA_HOME/logs/catalina.out
```


Tomcat: Servlet Deployment

- upon startup, tomcat deploys all servlets that are available in

```
$CATALINA_HOME/webapps
```

(considering path mappings etc. in `$CATALINA_HOME/conf/server.xml`)

Two alternatives how to make servlets available there:

- create a `myproject.war` file (web archive, similar to jar) and copy it into `$CATALINA_HOME/webapps`.
  (e.g. via build.xml targets "dist" and "deploy")
  (tomcat will unpack and deploy it upon startup)
  When replacing an old war file, delete the old unpacked stuff also.
- create a directory `myproject`, copy everything that is in the `WebRoot` directory there.
  (e.g. build.xml target "deploy"; cf. Demo-Servlet)

Tomcat's conf/server.xml

The URL paths to the projects can be defined to differ from the defaults (path name = webapps-directory name)

This is done in the `<Host>` element:

```
<Host>
  <Context path="/services/2016/xml2sql" reloadable="false" docBase="xmlconverter"/>
  :
</Host>
```

- if the project name is the same as the path (e.g. xquery-demo and servlet-demo), the entry can be omitted
  (usually, software projects do not have the same name, but distributed .war archive files can be renamed accordingly).
- reloadable: automatically reloads the servlet if the code is changed (e.g. a new .war archive).
  Should be done only during development.
- the path attribute is key. There can be multiple paths that are mapped to the same docBase.

General Servlet (Eclipse) Project Directory Structure

MyProject: project directory (anywhere outside tomcat)

MyProject/build.xml: the ant file for compiling and deploying – see later.

MyProject/src: the .java (and other) sources

MyProject/WebRoot: roughly, all this content is copied to the Servlet Container.
Plain HTML pages like index.html can be placed here.

MyProject/WebRoot/WEB-INF: the whole content of MyProject/WebRoot except WEB-INF is visible later (e.g., HTML pages can be placed here); the contents of WEB-INF is *used* by the Servlet Container.

MyProject/WebRoot/WEB-INF/web.xml: web application configuration,

MyProject/WebRoot/WEB-INF/classes: compiled binary stuff,

MyProject/WebRoot/WEB-INF/lib: used jars (except javax.servlet.jar – tomcat has own classes for servlets, this would create conflicts),

MyProject/lib: jars that are needed for building, but should not be copied to the Servlet Container (put javax.servlet.jar here),

build path: all jars in MyProject/lib + MyProject/WebRoot/WEB-INF/lib

SERVLET-DEMO EXAMPLE

Basic demonstration of servlet programming [servletdemo.zip on course Web page]

- The basic functionality is simple:
  a form where the user enters two numbers, and the servlet computes the sum (SumServlet),
  [HTML form with simple HTTP GET from servlet, simple answer]
- The same (added to the same form): the result is presented in an HTML table (FormatServlet),
  [HTML page as an answer]
- The same again (added to the same form): the numbers are taken, submitted to the SumServlet, and all three are submitted to the FormatServlet and a HTML page is created as answer (MakeCallsServlet).
  [HTML form with simple HTTP POST to servlet, inter-Servlet HTTP POST]
- The Demo collects all formatted tables and can output them.
  [persistent information, multiple GETs in the same servlet]
- it can be reset.

THE PROJECT'S WEB.XML (EXCERPT)

```
<web-app>
  <!-- Define servlet names and associate them with classfiles -->
  <servlet>
    <servlet-name>makecalls-servlet</servlet-name>
    <servlet-class>org.semwebtech.servletdemo.MakeCallsServlet</servlet-class>
    <init-param>
      <param-name>myURL</param-name>
      <param-value>http://localhost:8080/servletdemo/</param-value>
    </init-param>
  </servlet>
  <servlet> ... </servlet>
  <!-- define mapping of path tails to servlets -->
  <servlet-mapping>
    <servlet-name>makecalls-servlet</servlet-name>
    <url-pattern>/makecalls</url-pattern>
  </servlet-mapping>
  <servlet-mapping> ... </servlet-mapping>
  <!-- optionally: define default html page -->
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
</web-app>
```

Comments: web.xml

- <servlet>:
  - a short, abstract name (unique)
  - which java class
  - optional: init parameters that can be read in the `init(ServletConfig cfg)` method with `cfg.getInitParameter(`*`param-name`*`)`;
    - allows some adaptation of "foreign" servlets by only editing the web.xml, without recompiling Java code (e.g. if a .war contains only binaries).
    - if servlets (like MakeCallsServlet) need to call other servlets or use files, they can be told about their actual URLs.
    - directories where files can be found locally can be specified:

```
<init-param>
  <param-name>examplesDir</param-name>
  <param-value>/home/may/teaching/ssd/XQuery/</param-value>
</init-param>
```

- <servlet-mapping>:
  - url-pattern: key, things like `/*` allowed,
  - multiple patterns can be mapped to the same servlet.

COMMUNICATION WITH SERVLETS: HTTP METHODS GET AND POST

The servlets (virtually) run continuously in the Servlet Container and wait for incoming calls ...

HTTP GET and POST: request-response paradigm

HTTP GET should be used only if invocation does *not* change

- Request consists only of URL+parameters:

```
http://www.example.org/mondial?type=city&code=D&name=Berlin&province=Berlin
```

HTTP POST should be used if it has side effects or changes the state of the Web Service

- Request URL consists only of the plain URL,
- parameters (e.g. queries using forms) or any other information is sent via a stream

$\Rightarrow$ often also queries use POST

Response: always as a stream.

- other HTTP methods PUT (resource), DELETE (resource) are used in REST (Representational State Transfer) "architectures"
  (e.g. the eXist XML database and document management system uses REST)

Content of the Response

- if the service is invoked via the browser (forms; e.g. the XQuery-Demo), the response contents is the HTML code that is shown as "Web page" to the user.
- The “page” that is shown initially:
  - static index.html in the WebRoot directory (servletdemo), or
  - *answer* dynamically generated by the servlet on the first GET request (HTTP GET http://www.semwebtech.org/xquery-demo).
- if the service is invoked by another Web Service, the answer contains data (this course: in XML form).

Simple GET: “Content” of the Request

- A simple GET (from filling a Web form) carries the parameters as extension to the URL:
  http://www.semwebtech.org/xquery-demo?query-text=//country[name='Germany']
  https://univz.uni-goettingen.de/qisserver/?search=3&raum.dtxt=2.101
  (simplified)

HTML Forms: invoking Web Services via Browser

The following elements (and several others) can be used in HTML pages:

`<form action="./sum" method="get">`	will call *thisURL*/sum, HTTP GET
`<input type="text" name="a"/>`	form field to type in parameter “a”
`<input type="text" name="b"/>`	form field to type in parameter “b”
`<input type="submit"/>`	click here to submit HTTP GET ...
`</form>`	with the parameters a and b

- the call URL is e.g. `http://localhost:8080/servletdemo/sum?a=4&b=5`
- an HTML page can contain multiple separate forms, to the same or different URLs/“actions”,
- cf. servletdemo/WebRoot/index.html

SERVLET PROGRAMMING

- event-driven (cf. SAX): on incoming HTTP connections, the servlet container calls the servlets' doGet() (=react-on-get) and doPost() (=react-on-post) methods.

```
public class MyServlet extends HttpServlet
{ public void init(ServletConfig cfg) throws ServletException
    {  // initialization ...
       // read web.xml init params by  cfg.getInitParameter(...);
    }
  protected void doGet(HttpServletRequest request,
            HttpServletResponse response) throws ServletException  { ... }
  protected void doPost(HttpServletRequest request,
            HttpServletResponse response) throws ServletException  { ... }
}
```

- doGet() and doPost() both read the HttpServletRequest and write the HttpServletResponse object,
- the HttpServletRequest differs for GET (simpler) and POST (including a stream),
- the HttpServletRequest always provides a stream.

Recall:

- the distribution of connection URLs to projects is done according to tomcat's server.xml,
- the distribution inside of the project to servlets is done according to the project's web.xml,
- multiple URLs can be mapped to the same method (doGet/doPost) of the same servlet (Demo: FormatServlet)

$\Rightarrow$ must be analyzed in doGet() and doPost().

- Request.getPathInfo(): contains the tail of the URL path *after* the mapping by web.xml (non-null if `<url-pattern>/*</url-pattern>`)
- Request.getServletPath(): contains the tail of the URL path that is exploited for mapping according to the web.xml.

```
doGet/doPost(HttpServletRequest req, HttpServletResponse resp) throws ...
{ String path = req.getPathInfo();
  if (path == null) path = req.getServletPath();
  if (path.startsWith("/reset")) { ... }
  else if (path.startsWith("/format")) { ... }
  else if (path.startsWith("/all")) { ... }
}
```

Servlet Programming: Read Parameters and Contents

- GET and POST Requests can have parameters; POST can also have contents
- in doGet() and doPost() for accessing parameters:

```
java.util.Map<java.lang.String, java.lang.String[]> mymap =
                  req.getParameterMap();
String strA = req.getParameter("a");
```

(always Strings!)

- in doPost() for reading contents:

`ServletInputStream in = req.getInputStream();`
retrieves the body of the (POST) request (as binary data) using a ServletInputStream, where any Reader (e.g. a StAX XMLStreamReader) can be put on
(usually, set reader's encoding to UTF-8).
`java.io.BufferedReader r = req.getReader();` retrieves the body of the (POST) request as character data (according to character encoding decl of the body) using a BufferedReader.
For instance, one can create a JDOM from the contents:

```
BufferedReader in = req.getReader();
SAXBuilder builder = new SAXBuilder();
Document doc = builder.build(in);
Element root = doc.getRootElement();
```


Servlet Programming: Write into a Response

- doGet() and doPost() provide the HttpServletResponse object of the HTTP connection,
- it consists mainly of a stream,
- The requesting service (Browser, Web Service) has a Reader waiting on the stream (see next slide).
- `PrintWriter out = resp.getWriter();`
  yields a Writer to the response – send character text (or XML events).
- `ServletOutputStream os = resp.getOutputStream();`
  yields an output stream that can directly fed with write(), print(), println() or can be connected to another stream. Don't forget os.flush() and os.close().

Invoking a new HTTP Connection (to a Web Service)

(servletdemo: MakeCallsServlet)

- (Http)UrlConnection object is created by invoking the openConnection method on a URL;
- below: urlstr is a string, in the GET case already with parameters.

```
HttpURLConnection.setFollowRedirects(true);   // static
HttpURLConnection con = (HttpURLConnection) new URL(urlstr).openConnection();
con.setRequestMethod("GET or POST");
  con.setDoInput(true);     // can be omitted - default is true
  con.setDoOutput(true);    // default is false(!), for "get" it's OK
  con.setRequestProperty("Connection", "keep-alive"); // is answer takes longer
  con.setRequestProperty("Content-type", ...);
  con.setRequestProperty("Accept", "text/xml");
con.connect();
-- use con.getOutputStream() to write contents of the request
con.getOutputStream().close();
-- use con.getInputStream() to read contents of the response
-- BufferedReader in =
     new BufferedReader(new InputStreamReader(con.getInputStream(), "UTF-8"));
con.getInputStream().close();
```


Code: HTTP GET

- Parameters given with the URL:

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
public class HttpGetSimple {
  public static void main(String[] args) { try {
     BufferedReader br = null;
     URL inputURL = new URL("http://www.semwebtech.org/xquery-demo/" +
          "?action=query&query-text=//country[1]");
     HttpURLConnection con = (HttpURLConnection) inputURL.openConnection();
     con.setRequestMethod("GET");
     con.connect();
     String s = "";   StringBuffer res= new StringBuffer();
     br = new BufferedReader(new InputStreamReader(con.getInputStream(), "UTF-8"));
     while ((s = br.readLine()) != null) { res.append(s+ "\n"); }
     br.close();
     System.out.println(res);
  } catch (Exception e) { e.printStackTrace(); } }}
```

[Filename: java/HttpGetSimple.java]

Code: HTTP POST – Parameters in the Request

- https://docs.oracle.com/javase/tutorial/networking/urls/readingWriting.html

```
import java.io.BufferedReader;       import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.HttpURLConnection;   import java.net.URL;
public class HttpPostSimple {
  public static void main(String[] args) { try {
     BufferedReader br = null;
     URL inputURL = new URL("http://www.semwebtech.org/xquery-demo/");
     String params = "action=query&query-text=//country[1]";
     HttpURLConnection con = (HttpURLConnection) inputURL.openConnection();
     con.setRequestMethod("POST");
     con.setDoOutput(true);    // default is false(!)
     con.connect();
       OutputStreamWriter wr = new OutputStreamWriter(con.getOutputStream());
       wr.write(params);
       wr.flush();  wr.close();
     String s = "";   StringBuffer res= new StringBuffer();
     br = new BufferedReader(new InputStreamReader(con.getInputStream(), "UTF-8"));
     while ((s = br.readLine()) != null) { res.append(s+ "\n"); }
     br.close();  System.out.println(res);
  } catch (Exception e) { e.printStackTrace(); } }}
```

[Filename: java/HttpPostSimple.java]

HTTP Access in the Data Management Area

- HTTP GET and POST are important means to access “Deep Web” data via queries against forms, and “Linked Open Data” (LOD) (RDF data, [Semantic Web lecture]).

Alternative: [not tested]

- Connection getContent() method:
  returns an Object whose type is determined by the the content-type header field of the response. Uses a ContentHandler to convert data based on its MIME type to the appropriate class of Java Object.
- maybe useful for binary types?
- or even URL.getContent() as a shortcut for openConnection().getContent();
  String foo = (String) url.getContent();
  seems to be useful for plain GET on HTML pages;
- for XML content, using the stream seems to be more useful
  ($\rightarrow$ SAXBuilder$\rightarrow$ JDOM, or $\rightarrow$ StAX)

Notes on Handling Character Encodings

- default for WebServices is ISO-8859-1 (covers german umlauts, swedish etc.)
- then, for HTML forms, set also
  `<form method="get/post" accept-charset="ISO-8859-1">`
- UTF-8 also covers chinese, persian, etc. (localnames in Mondial)
- Web Service side:
  - if HTTP GET is used, request character encoding can only be set *globally* (Apache tomcat: URIEncoding attribute of the <Connector port="..."> element in server.xml to UTF-8).
  - HTTP POST: request.setCharacterEncoding("UTF-8") before reading parameters or contents (e.g. DBIS XQuery and SQL Web Interfaces);
  - use also response.setCharacterEncoding("UTF-8")

DATA EXCHANGE: AN INTEGRATED XML PERSPECTIVE

- HTTP connections are Unicode.
- exchanging XML via HTTP basically works on its serialization
  - explicitly working with Reader→String/StringBuffer and String/StringBuffer→Writer is possible, but often not necessary;
  - in:
    - let a SAXBuilder build a JDOM,
    - put SAX or an StAX XMLEventReader on the InputStream,
    - put a JAXB Unmarshaller on the InputStream,
    - put the Digester on the InputStream,
    - cf. Examples where these were put on the FileInputStream for mondial.xml.
  - out:
    - serialize XML by putting an XMLEventWriter on the OutputStream,
    - let JAXB write into it, ...

A Note on Multithreading

- servlets can be instantiated by the container permanently or on-demand.
- if multiple requests for the same servlet come in, the servlet container can run multiple threads on the same instance of a servlet.
  - be careful with instance variables,
  - implement mutual exclusion if necessary
- the servlet container can also create (and remove) additional instances of a servlet.

PHP IN TOMCAT

- Tomcat is Java-based,
- Embedded PHP in HTML files or pure PHP is not executed by default.
- Name HTML files that include embedded PHP (cf. Slide 186) *filename*.php,
- there are several implementations of PHP in Java,
- e.g. see `https://stackoverflow.com/questions/779246/run-a-php-app-using-tomcat/779319` and `http://www.studytrails.com/blog/php-on-a-java-app-server-apache-tomcat-using-quercus/`