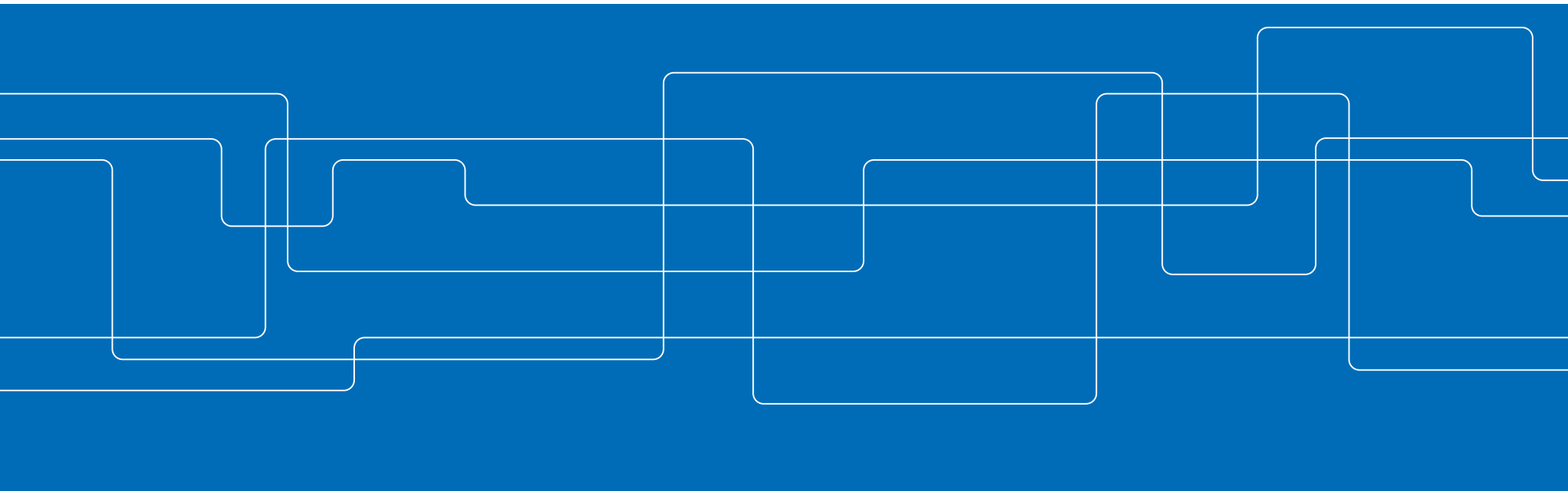
KTH ROYAL INSTITUTE
OF TECHNOLOGY

ID2208 Programming Web Services

Homework 1 - XML Processing

Kim Hammar (kimham@kth.se)
Cosar Ghandeharioon (cosarg@kth.se)
Mihhail Matskin (misha@kth.se)

January 23, 2018



Outline

Administration

Formalities

- Two members per group in all homeworks and project.
- If any general problem or question, use canvas discussion forum
- All deliverables will be through canvas.

Administration

Bonus system

- Three homeworks. Timely delivery and approval of all homeworks gives **5 bonus points**.
- One project. Timely delivery and approval of project gives **5 bonus points**.
- In total 10 bonus points for exam.
- Must pass homeworks+project to pass course.

Administration

Important Dates

Table: Important Dates

Release Date	Due Date	Deliverable
2018-01-23	2018-01-29	Homework 1
2018-01-30	2018-02-05	Homework 2
2018-02-06	2018-02-12	Homework 3
2018-02-13	2018-02-27	Project

Homework 1 - XML Processing[1]

Introduction

- Aim: learn tooling for XML processing and gain deeper understanding of XML as a data format.

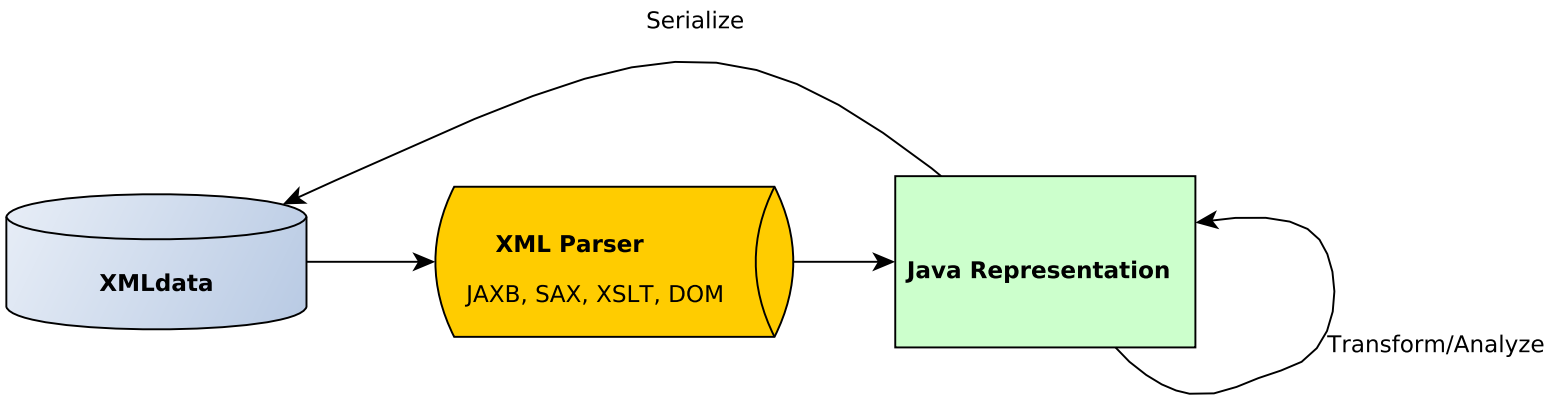


Figure: XML Processing Pipeline

[1]Original lab content prepared by Dr. Shahab Mokarizadeh

Homework 1 - XML Processing

Problem Description 1/4

Build application for employment service company. Users of application: job-seekers. Users upload on registration:

- **Degree and transcript records**: from university web service
- **Employment records**: from an employment office webservice
- **personal information**: provided by user.

And companies that upload:

- **Company information**

Homework 1 - XML Processing

Problem Description 2/4

All of the data is in XML format.

Your application should take the XML information, read it into memory, and process it to build a job-seeker profile.

When done, the job-seeker profile is saved to disk in XML format.

Homework 1 - XML Processing

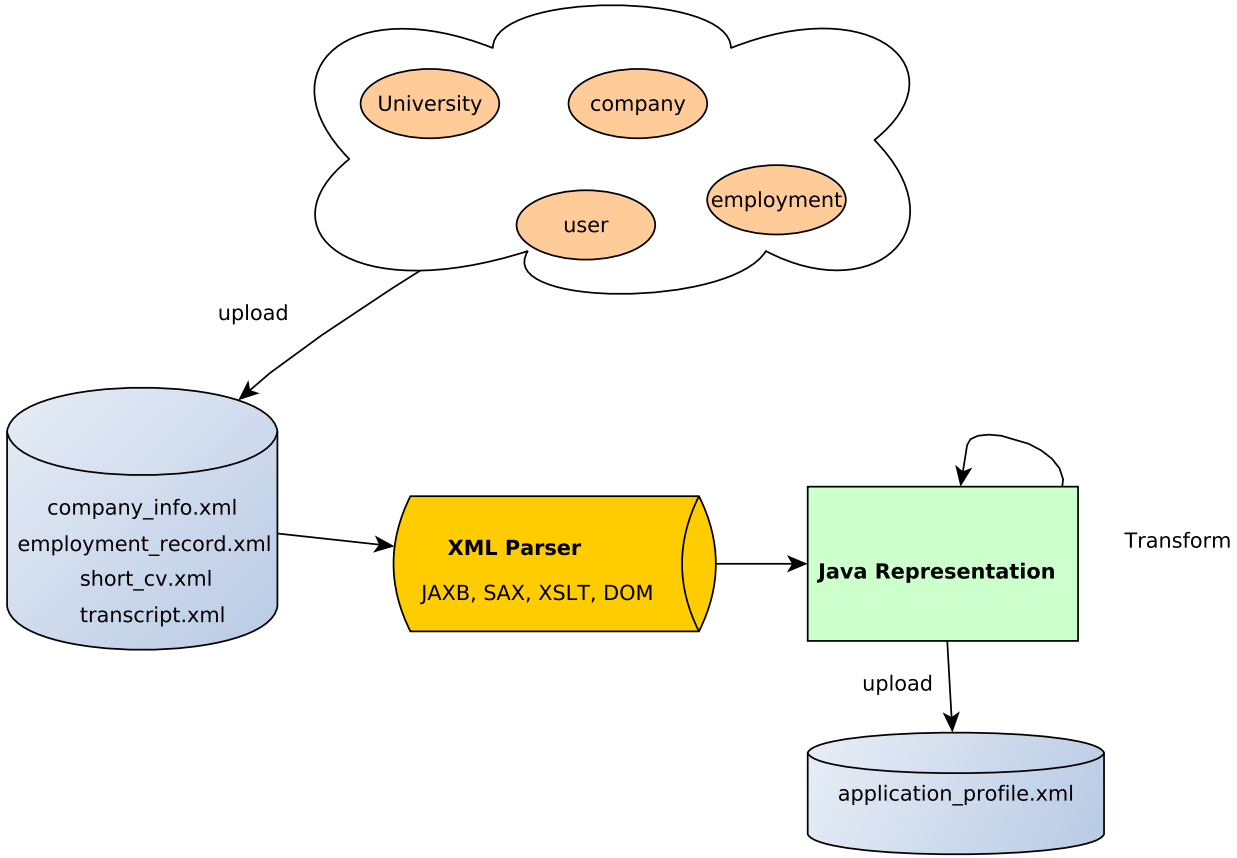
Problem Description 3/4

A profile of job seeker is made of:

- CV,
- relevant academic degree(s),
- previous working experiences,
- information about companies where the applicant worked for before,
- motivation letter,
- places desired to work,
- type of job (permanent, part time, contract,...) ,
- references and other relevant qualifications (e.g. driving license)

Homework 1 - XML Processing

Problem Description 4/4



Homework 1 - XML Processing

Task 1 1/2

- Given the information about what an application profile contains, design XML schema (XSD) for each XML document:
  - `transcript.xsd`,
  - `employment_record.xsd`,
  - `company_info.xsd`,
  - `short_cv.xsd`,
  - `application_profile.xsd`.

The individual documents can contain more fields than what is required to create the application-profile, but it is not required.

Homework 1 - XML Processing

Task 1 2/2

- Given your schemas, create a few sample documents that are valid according to your schemas. These documents will be used later for XML processing (task 2).

Homework 1 - XML Processing

Task 2 1/2

Write a program with the following functionality

1. **Parse** your sample XML documents into java objects
2. **Combine** the parsed documents into an ApplicationProfile
3. **Serialize** the profile back into XML
4. Try all of the following libraries/parsing **techniques**:
   - Document Object Model (DOM)
   - Simple API for XML (SAX)
   - Extensible Stylesheet Language Transformations (XSLT)
   - JAXB

Homework 1 - XML Processing

Task 2 2/2

- Example: parse `transcript.xml` with DOM, parse `employment_record.xml` with SAX etc.
- It is OK to focus on one library but you should try all of them
- **Final requirement:** In addition to fields in your sample documents, add a field **GPA** to the profile.
- GPA should not be part of `transcript.xml`, it should be calculated in your application that processes the XML.

Homework 1 - XML Processing

Deliverables

- **Textual report** explaining what you did.
- The **XML schemas** (5 xsd files)
- The populated **sample XML documents** (4, `transcript.xml`, `employment_record`...)
- The **source code** of your XML processing project, **including XSLT file.**
- The generated **application-profile in xml** format.
- You will demonstrate that your code works in a **presentation** (will be announced in canvas).

Homework 1 - XML Processing

Tips

- Use meaningful names to XML tags
- Use namespaces
- Use complex and simple types in your schemas
- Use xml restrictions in your schemas

XML Processing Primer

Introduction

XML is a **textual format**, requires **parsing into memory**.

Techniques for parsing XML (you will try all of them!):

- Document Object Model (DOM)
- Simple API for XML (SAX)
- Extensible Stylesheet Language Transformations (XSLT)
- JAXB

Good tutorials on the internet, use it if you need.

Any prog-lang that has the framework support is OK.

XML Processing Primer

XML Schemas

Useful examples in lecture slides and in the course book.

Use an editor which can highlight well and performs syntax checking of your XML.

```
 <xsd:element name="FirstName">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:minLength value="1"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
```

XML Processing Primer

DOM

DOM-parsing: parse the XML data into a DOM tree and use an API to interact with the tree (whole tree is loaded into memory).

```
NodeList nodes = doc.getElementsByTagName("Company");
int len = nodes.getLength()
for (int i = 0; i < len i++) {
   Element companyElement = (Element) companyNodes.item(i);
   .
   .
   .
//Extract the info you need from the element to create application profile
```

XML Processing Primer

SAX 1/2

What if DOM tree is too large to fit into memory?

SAX parses XML gradually and generate events when parsing, e.g *“start of element”, “end of element”* etc.

More difficult to program but not as memory hungry. You program by implementing event-handlers.

Create handler:

```
private class MainHandler extends DefaultHandler {
  ..
  ..
```

XML Processing Primer

SAX 2/2

Override event handlers that you need:

```
/* Called at the beginning of an element.*/
@Override
public void startElement(String namespaceURI, String localName, String qName, Attributes atts) throws
        SAXException {
    if (qName.equalsIgnoreCase("FirstName")) {
            employeeFirstName = true;
        }
    if(....)
    ..
    ..

/* Called when character data is encountered. */
@Override
public void characters(char ch[], int start, int length) throws SAXException {
String data = new String(ch, start, length);
    if (employeeFirstName) {
            employee.setFirstName(data);
            employeeFirstName = false;
        }
        if(....)
        .
```

XML Processing Primer

JAXB 1/2

- JAXB: Map java classes $\longleftrightarrow$ XML.
- Automatic marshall/unmarshall
- IDEs: create JAXB Pojos from XML schema
- XJC CLI: create JAXB Pojos from XML schema
- Once you have the java classes the parsing is easy.

Create Java Class From XSD:

```
> xjc application_profile.xsd .
parsing a schema...
compiling a schema...
application_profile/hw1/id2208/se/kth/limmen/ApplicationProfile.java
application_profile/hw1/id2208/se/kth/limmen/ObjectFactory.java
application_profile/hw1/id2208/se/kth/limmen/package-info.java
> whereis xjc
xjc: /usr/bin/xjc /usr/share/man/man1/xjc.1.gz
```

XML Processing Primer

JAXB 2/2

Unmarshalling:

```
String DOCUMENT = "xml/documents/transcript.xml";
transcriptDocument = new File(DOCUMENT);
jaxbContext = JAXBContext.newInstance(Transcript.class);
unmarshaller = jaxbContext.createUnmarshaller();
unmarshaller.setSchema(transcriptSchema);
return (Transcript) unmarshaller.unmarshal(transcriptDocument);
```

Marshalling

```
jaxbContext = JAXBContext.newInstance(ApplicationProfile.class);
marshaller = jaxbContext.createMarshaller();
marshaller.marshal(applicationProfile, applicationProfileDocument);
```

To get the right output, might have to tune the marshaller: `marshaller.setProperty(....)`.

XML Processing Primer

XSLT 1/3

- **XSLT**: write stylesheets describing XML processing
- **XSLT processor:** XSLT stylesheet + XML document $\rightarrow$ transformed XML
- **XSLT**: uses XPath to find information in an XML document.
- Think of XML document as a tree and XPath as expressions to match things in the tree.

Snippet from target document:

```
<PersonalInformation>
    <FirstName>John</FirstName>
    <LastName>Doe</LastName>
    <City>Stockholm</City>
    <CivicRegistrationNumber>910406−1337</CivicRegistrationNumber>
    <Email>johndoe@kth.se</Email>
</PersonalInformation>
```

XML Processing Primer

XSLT 2/3

Below is some XSLT code to select a subset of the elements of the target XML document to be used in the output document.

Snippet from stylesheet:

```
<xsl:template match="/cv:ShortCV">
        <xsl:element name="Person">
            <xsl:element name="FirstName">
                <xsl:value-of select="cv:PersonalInformation/cv:FirstName"/>
            </xsl:element>
            <xsl:element name="LastName">
                <xsl:value-of select="cv:PersonalInformation/cv:LastName"/>
            </xsl:element>
            <xsl:element name="CivicRegistrationNumber">
                <xsl:value-of select="cv:PersonalInformation/cv:CivicRegistrationNumber"/>
            </xsl:element>
        </xsl:element>
```

XML Processing Primer

XSLT 3/3

Snippet from output document:

```
<Person>
    <FirstName>John</FirstName>
    <LastName>Doe</LastName>
    <CivicRegistrationNumber>910406−1337</CivicRegistrationNumber>
</Person>
```

Thank You and Good Luck!

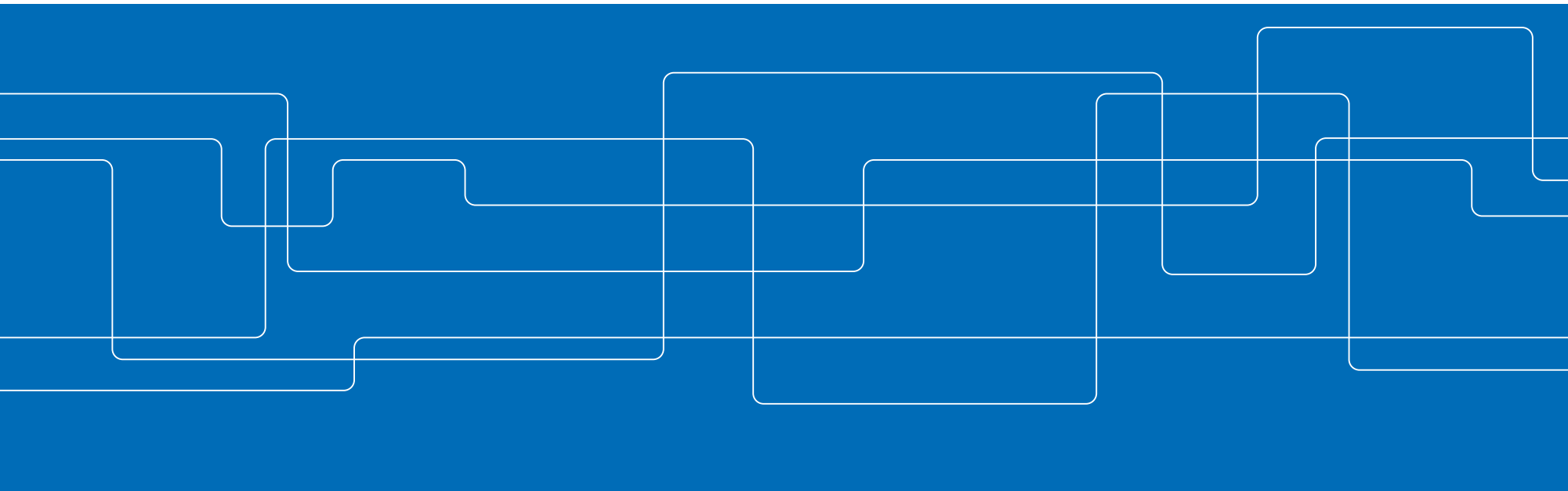
KTH ROYAL INSTITUTE
OF TECHNOLOGY

ID2208 Programming Web Services

Homework 2 - SOAP & WSDL

Kim Hammar (kimham@kth.se) & Mihhail Matskin (misha@kth.se)

January 30, 2018



Outline

Homework 2 - SOAP & WSDL [1]

Introduction

Goals of this lab:

- Design and Develop **XML Web Services**
- Develop Web **Service Client**
- **SOAP** processing

[1] The tasks of this lab were designed by Hooman Peiro Sajjad and is based on a tutorial published by IBM and some e Oracle documents pointed out in the reference.

Homework 2 - SOAP & WSDL

Problem Description 1/3

Design and implement: flight ticket reservation web service with the **functionality**:

1. Authorization of clients. Service require some valid token to get access.

```
@WebMethod
    public String login(String username, String pw) throws AuthorizationException {
...
```

Homework 2 - SOAP & WSDL

Problem Description 2/3

2. Provide itineraries given departure and destination city. Combine many flights if no direct flight.

```
@WebMethod
    public ArrayList<Itinerary> getItineraries(String departureCity, String destinationCity, String
            token) throws AuthorizationException {
```

3. Check availability of tickets and finding their price for a given itinerary and given date.

```
@WebMethod
    public ArrayList<Ticket> getAvailableTickets(Date date, Itinerary itinerary, String token) throws
            AuthorizationException {
```

Homework 2 - SOAP & WSDL

Problem Description 3/3

4. Output the price of available itineraries

```
@WebMethod
  public ArrayList<PriceEntry> getPriceList(String token) throws AuthorizationException {
```

5. Book tickets for requested itinerary.

```
@WebMethod
    public Receipt bookTickets(int creditCardNumber, ArrayList<Ticket> tickets, String token)
            throws AuthorizationException {
```

6. Issue tickets. Only booked tickets can be issued.

```
@WebMethod
    public ArrayList<PurchasedTicket> issueTickets(Receipt receipt, String token) throws
            AuthorizationException {
```

Homework 2 - SOAP & WSDL

Tasks 1/2

- Implement half of the services listed above in the **top-down fashion**.
- Top-down: WSDL $\rightarrow$ Java (or other lang).
- Do automatic generation with the help of tools.
- Implement the other half of the services in **bottom-up fashion**.
- Bottom-up: Java $\rightarrow$ WSDL.

Homework 2 - SOAP & WSDL

Tasks 2/2

- Develop a **test-client** for the web service that tests all of the services above.
- Explain **in the report** how you would extend the SOAP messages of your service with headers to manage some of the functionality of the service.
- **Hint:** Think about authentication.

Homework 2 - SOAP & WSDL

Deliverables

- **Textual report** explaining what you did
- The **Source code**, **WSDLs** and **Schema** of the implemented Web services.
- The **XML** of constructed, sent and received **SOAP messages** communicated among services. (Some sample messages is enough).
- A short description about your system design.
- **Executable** version of your system
- Show your fully functional system in a 10-15 minutes **presentation**.

Java WebServices Primer

Some Links

- Tutorial for creating a JAX-WS web service in Netbeans [Net18]
- Tutorial for creating JAX-WS web service by IBM [BH18]
- Apache Tomcat Application Server [Fou18]
- Glassfish Application Server [Ora18b]
- Many more tutorials on the web, take a look!

Java WebServices Primer

JAX-WS intro 1/2

- You can use any language or framework that supports the bottom-up/top-down techniques.
- **JAX-WS**: framework for creating XML-based webservices in Java.
- Framework design: create **WAR files** to be deployed on **application servers.**
- Alternative: use some **lightweight java server** for deployment.

Java WebServices Primer

JAX-WS intro 2/2

JAX-WS runtime hides all the low-level stuff (serialization, threading etc) for you.

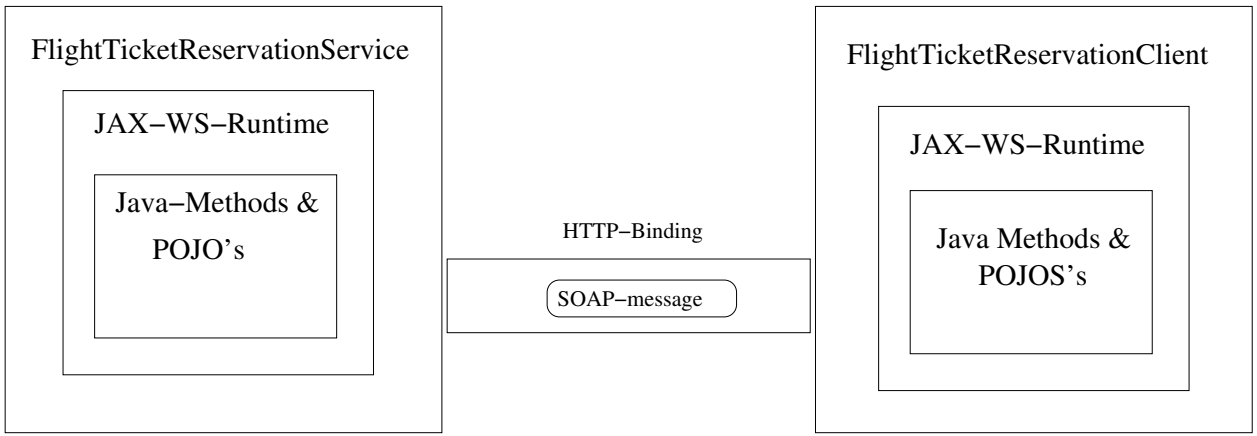


Figure: Architecture

Java WebServices Primer

JAX-WS annotations

JAX WS uses an annotation based programming model.

```
@WebService
public class Hello {
    private String message = new String("Hello, ");

    public void Hello() {
    }

    @WebMethod
    public String sayHello(String name) {
        return message + name + ".";
    }
}
```

Java WebServices Primer

JAX-WS marshalling

- JAX-WS uses JAXB under the hood for marshalling and unmarshalling objects.
- Powerful programming pattern: return java obejcts from webmethods.
- Make sure the objects you return are annotated with JAXB annotations (remember HW1).

```
@XmlRootElement(name = "Ticket")
public class Ticket {
....
...
      @XmlElement(name = "Date")
      public Date getDate() {
            return date;
      }
}
```

Java WebServices Primer

Deployment 1/4

- JAX-WS comes with a lightweight webserver
- You use javax.xml.ws.Endpoint [Ora18a] to publish a simple web service.

```
//implementor should be a annotated @WebService class
Object implementor = new FlightTicketReservationService();
String address = "http://localhost:9000/kth.se.id2208.bottom_up.FlightTicketReservationServiceTopDown";
Endpoint.publish(address, implementor);
```

Java WebServices Primer

Deployment 2/4

- Alternative deployment: application server + .war file.
- Below is the steps to do it with command line.

1. Create war file using maven plugin [Pro18]

```
mvn install
..
..
[INFO] Webapp assembled in [92 msecs]
[INFO] Building war:
        /media/limmen/HDD/workspace/id2208/WebServicesScenarios/hw3/target/hw3.war
```

2. Copy war file to my TOMCAT installation (first removing previous deployed war)

```
rm -rf ~/programs/apache-tomcat-7.0.82/webapps/ROOT*
cp ~/workspace/id2208/WebServicesScenarios/hw3/target/hw3.war
        ~/programs/apache-tomcat-7.0.82/webapps/ROOT.war
```

Java WebServices Primer

Deployment 3/4

3. Start tomcat

```
./catalina.sh start
```

4. Test service with curl (your service in this homework will not use JSON but SOAP)

```
curl -H "Content-Type: application/json" -X POST -d
        '{"username":"kim","password":"id2208"}' http://localhost:8080/rest/login
ID2208_AUTH_TOKEN
```

Java WebServices Primer

Deployment 4/4

You can also set this up in your IDE and skip the whole command-line!

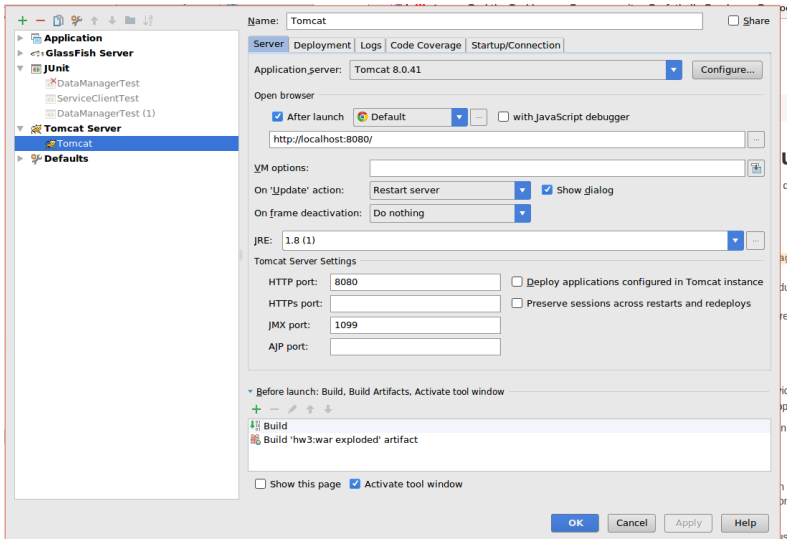


Figure: IntelliJ Tomcat configuration setup

Java WebServices Primer

Display SOAP messages

Use **VM argument**
-Dcom.sun.xml.ws.transport.http.HttpAdapter.dump=true
to **display SOAP messages** to stdout when the webservice receives and sends responses. Below is an example log.

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Body>
<ns2:login xmlns:ns2="http://flight_reservation"><arg0>kim</arg0><arg1>id2208</arg1></ns2:login>
</S:Body>
</S:Envelope>
```

Any method to print the SOAP messages is OK to use.

Java WebServices Primer

Top-Down Design 1/2

There are a lot of examples in your coursebook and in the lecture slides.

Example snippet of WSDL

```
<operation name="Login">
    <soap:operation soapAction="Login"/>
    <input>
        <soap:body use="literal"/>
    </input>
    <output>
        <soap:body use="literal"/>
    </output>
    <fault name="AuthorizationException">
        <soap:fault name="AuthorizationException" use="literal"/>
    </fault>
</operation>
```

Java WebServices Primer

Top-Down Design 2/2

You also have to design XML schemas for the messages you use in your WSDL.

Example Schema for a Login-Invocation message

```
<xsd:element name="Login">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Username" type="xsd:string" minOccurs="0"/>
            <xsd:element name="Password" type="xsd:string" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

Java WebServices Primer

Top-Down generation 1/2

The tools **wsgen** [Ora18c] and **wsimport** [Ora18d] can be used to generate WSDL file given a web service and vice verse. Likely your IDE will have built in support for this also. wsgen and wsimport are part of the JDK.

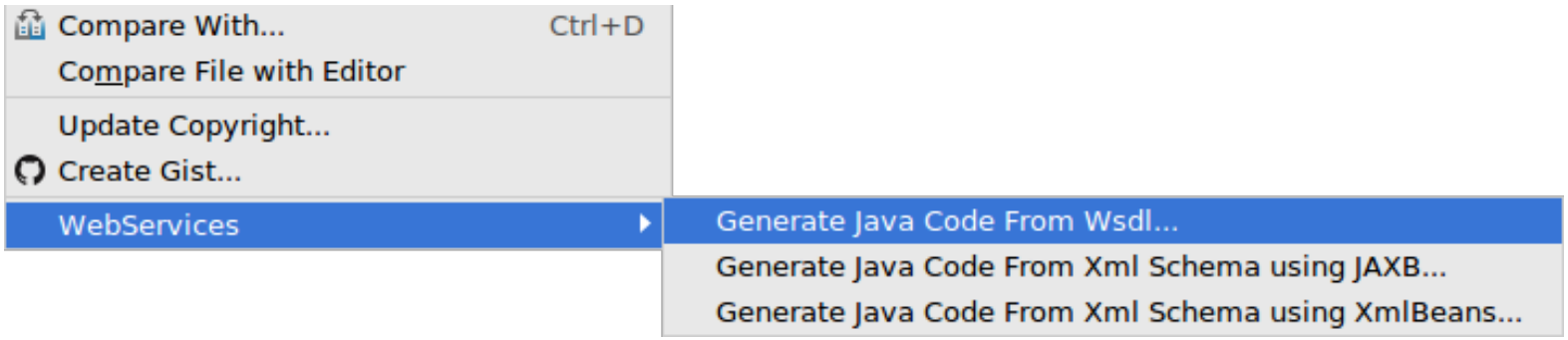


Figure: IntelliJ wsgen + wsimport

Java WebServices Primer

Top-Down generation 2/2

Below is an example of using wsimport on the commandline to generate the java code from your WSDL (use -keep to save source and not just compiled files).

```
kim@limmen ~/w/t/wsdl> ls
totalt 8
-rw-rw-r-- 1 kim kim 6398 jan 6 19:33 FlightTicketReservationService.wsdl
kim@limmen ~/w/t/wsdl> wsimport -keep -verbose FlightTicketReservationService.wsdl
parsing WSDL...
Generating code...
flightticketreservationservice/top_down/kth/se/id2208/AuthorizationException.java
flightticketreservationservice/top_down/kth/se/id2208/AuthorizationException_Exception.java
flightticketreservationservice/top_down/kth/se/id2208/FlightTicketReservationPortType.java
flightticketreservationservice/top_down/kth/se/id2208/FlightTicketReservationService.java
flightticketreservationservice/top_down/kth/se/id2208/GetItineraries.java
flightticketreservationservice/top_down/kth/se/id2208/GetItinerariesResponse.java
flightticketreservationservice/top_down/kth/se/id2208/GetPriceList.java
flightticketreservationservice/top_down/kth/se/id2208/GetPriceListResponse.java
flightticketreservationservice/top_down/kth/se/id2208/ItineraryType.java
flightticketreservationservice/top_down/kth/se/id2208/Login.java
flightticketreservationservice/top_down/kth/se/id2208/LoginResponse.java
flightticketreservationservice/top_down/kth/se/id2208/ObjectFactory.java
```

Java WebServices Primer

Bottom-up generation 1/2

Use the tools on the command line or your IDE.

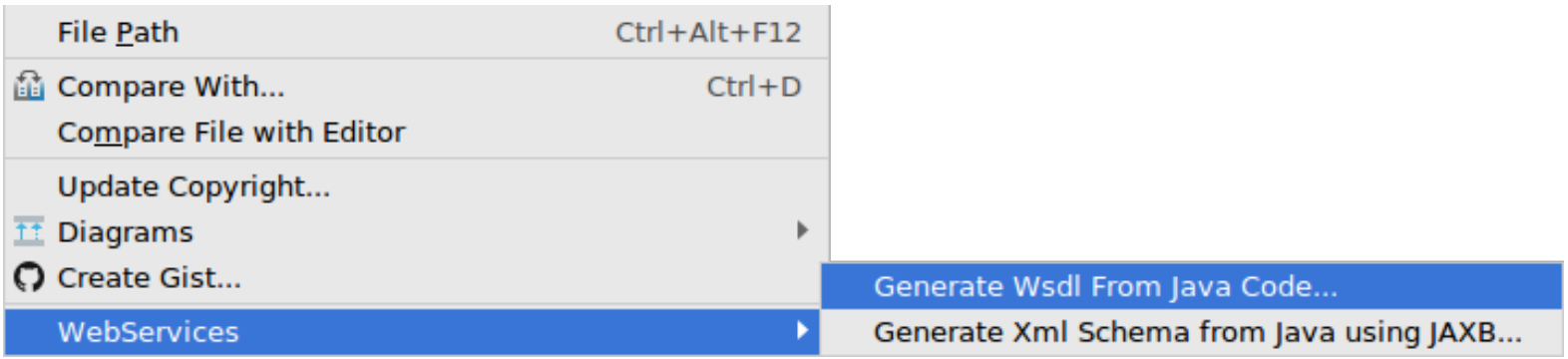


Figure: IntelliJ wsgen + wsimport

Java WebServices Primer

Bottom-up generation 2/2

Below is an example of using wsgen on the commandline to generate the WSDL, Schema, and all JAX-WS portable artefacts (JAXB annotated classes).

```
kim@limmen ~/w/t/j/hw2> wsgen -verbose -keep -cp target/classes/
        kth.se.id2208.bottom_up.FlightTicketReservationService -wsdl
FlightTicketReservationServiceTopDown_schema1.xsd
FlightTicketReservationServiceTopDown.wsdl
kth/se/id2208/bottom_up/jaxws/AuthorizationExceptionBean.java
kth/se/id2208/bottom_up/jaxws/BookTickets.java
kth/se/id2208/bottom_up/jaxws/BookTicketsResponse.java
kth/se/id2208/bottom_up/jaxws/GetAvailableTickets.java
kth/se/id2208/bottom_up/jaxws/GetAvailableTicketsResponse.java
kth/se/id2208/bottom_up/jaxws/GetItineraries.java
kth/se/id2208/bottom_up/jaxws/GetItinerariesResponse.java
kth/se/id2208/bottom_up/jaxws/GetPriceList.java
kth/se/id2208/bottom_up/jaxws/GetPriceListResponse.java
kth/se/id2208/bottom_up/jaxws/IssueTickets.java
kth/se/id2208/bottom_up/jaxws/IssueTicketsResponse.java
kth/se/id2208/bottom_up/jaxws/Login.java
```

Java WebServices Primer

WebService Client

- The tools **wsgen** [Ora18c] and **wsimport** [Ora18d] can be used to generate clients from WSDL as well. Your IDE might support it natively.
- The client will typically be generated with a bunch of regular java methods that you can invoke for testing, e.g:

```
FlightTicketReservationPortType service = new
        FlightTicketReservationService().getFlightTicketReservationPortTypePort();
String AUTH_TOKEN = service.login("kim", "id2208");
System.out.println("Successfully logged in as user 'kim', AUTH_TOKEN received:" + AUTH_TOKEN);
System.out.print("Looking up price−list of all itineraries...");
ArrayList<PriceEntry> priceList = (ArrayList) service.getPriceList(AUTH_TOKEN);
System.out.println("SUCCESS! Price list:");
printPriceList(priceList);
```

Thank You and Good Luck!

References I

Naveen Balani and Rajeev Hathi, *Design and develop jax-ws 2.0 web services*, `https://www6.software.ibm.com/developerworks/education/ws-jax/ws-jax-a4.pdf`, 2018.

The Apache Software Foundation, *Apache tomcat*, `http://tomcat.apache.org/`, 2018.

References II


NetBeans, *Getting started with jax-ws web services*, `https://netbeans.org/kb/docs/websvc/jax-ws.html`, 2018.


Oracle, *Endpoint*, `https://docs.oracle.com/javase/7/docs/api/javax/xml/ws/Endpoint.html`, 2018.

References III


_____, *Glassfish*, https://javaee.github.io/glassfish/, 2018.


_____, *wsgen*, https://docs.oracle.com/javase/6/docs/technotes/tools/share/wsgen.html, 2018.

References IV

_____, *wsimport*, https://docs.oracle.com/javase/6/docs/technotes/tools/share/wsimport.html, 2018.

Apache Maven Project, *Apache maven war plugin*, https://maven.apache.org/plugins/maven-war-plugin/, 2018.

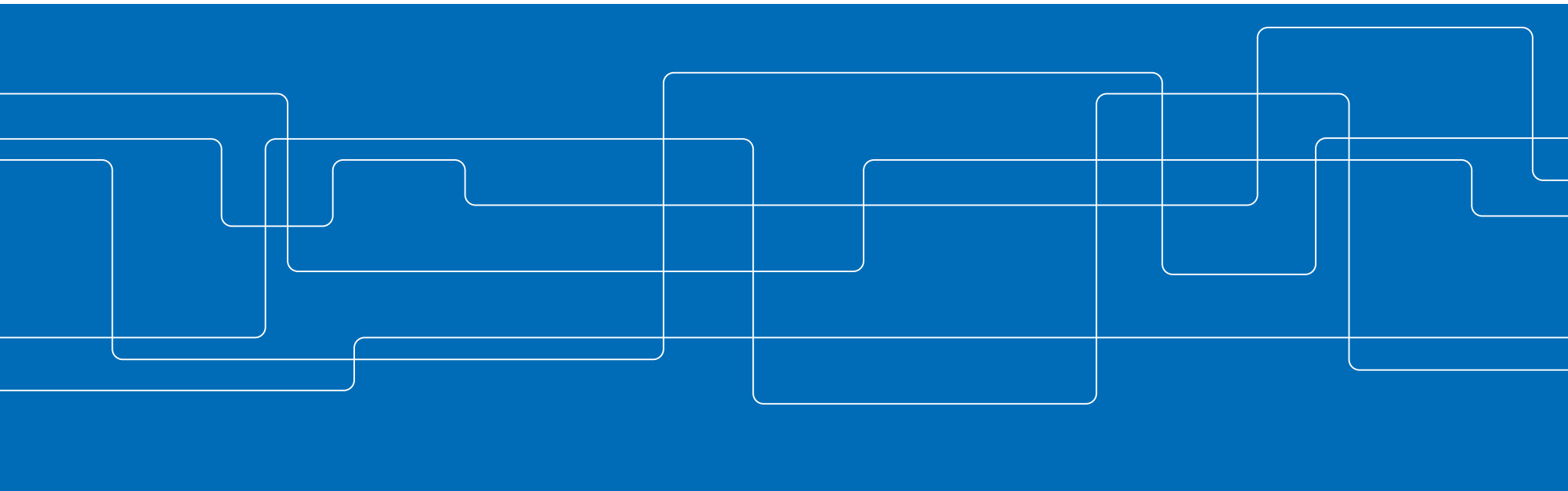
KTH ROYAL INSTITUTE
OF TECHNOLOGY

ID2208 Programming Web Services

Homework 2 - SOAP & WSDL

Kim Hammar (kimham@kth.se) & Mihhail Matskin (misha@kth.se)

Frebruary 6, 2018



Outline

Homework 3 - RESTful Web Service [1]

Introduction

Goals of this lab:

- Design and Develop **RESTful Web Services**
- Developing **Web Service Client**
- **JSON/XML** Processing

[1] The tasks of this lab were designed by Hooman Peiro Sajjad and is based on the following resources: Lecture notes by John Cowan and tutorial by Huang et al [YMHwDFW09]

Homework 3 - RESTful Web Service

Problem Description

- Same problem use-case as last lab.
- In this lab use RESTful web services with XML or JSON.

Table: The most common HTTP methods (you can use others as well)

HTTP Method	Usage
GET	get a resource (e.g get itinearies)
POST	create resource (e.g create user session by login or bookticket)
PUT	update a resource (e.g add a flight)
DELETE	delete a resource (e.g remove ticket)

Homework 3 - RESTful Web Service

Tasks 1/3

- Implement all webservices and functionality from HW2 as RESTful web services.

- Develop a client to test all of your RESTful resources with HTTP operations (should be atleast all of the following methods: GET/POST/PUT/DELETE).

  Tip: implement the client as a set of automatic unit-tests that asserts that the response from each endpoint is correct in terms of response-code and content.

Homework 3 - RESTful Web Service

Tasks 2/3

Think about the design of your webservice. Designing a RESTful resources includes careful consideration of the following.

- what RESTful **resources** to use?
- what **URLs** to use?
- what **mediatype** to use? Can user control mediatype with its request?
- what **HTTP methods** to use?
- what **HTTP response** codes to use?

Homework 3 - RESTful Web Service

Tasks 3/3

Think about: REST vs XML based Webservices

- REST is less powerful in terms of business-to-business integration.
- Integration can be improved slightly by following RESTful design standards (e.g dont use GET method for creating resources).
- REST is less complex.
- See more in lecture slides.

Homework 3 - RESTful Web Service

Deliverables

- **Textual report** explaining what you did.
- **Source code** for your project.

Java RESTful WebServices Primer

Some Links

- Tutorial on building a RESTful Web service using Jersey and Apache Tomcat [YMHwDFW09].
- Tutorial on using Jersey Client to consume a RESTful web service [Pod09].
- Jersey Test Framework [fM18].
- Apache Tomcat Application Server [Fou18].
- Glassfish Application Server [Ora18a].
- Many more tutorials on the web, take a look!

Java RESTful WebServices Primer

Jersey intro

You can use any library or programming lang you like for building the RESTful web service but we recommend **Jersey** [Ora18b].

Jersey is **based on annotations** just like JAX-WS. In addition Jersey can be deployed to web servers in similar fashion as JAX-WS applications.

Jersey allows you to write RESTful web services on a **high-level**, the **runtime** will handle low-level details.

Java RESTful WebServices Primer

Jersey Annotations 1/2

Annotate your classes to create RESTful resources and annotate methods to create RESTful operations on the resources.

RESTful resource

```
@Path("/itineraries")
public class Itineraries {
```

Java RESTful WebServices Primer

Jersey Annotations 2/2

RESTful operation

```
@GET
@Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
public ArrayList<Itinerary> getItineraries(@QueryParam("departmentCity") String departmentCity,
                                           @QueryParam("destinationCity") String destinationCity,
                                           @QueryParam("token") String token) {
```

Operations can return multiple mediatypes, Jersey runtime will check the mediatype of the HTTP request to decide which one to return.

Java RESTful WebServices Primer

Jersey Marshalling

Jersey automates java POJO $\longleftrightarrow$ JSON, for XML format use annotations just like in HW2.

```
@XmlRootElement(name = "Ticket")
public class Ticket {
....
...
      @XmlElement(name = "Date")
      public Date getDate() {
            return date;
      }
}
```

Java WebServices Primer

Deployment 1/4

- Common practice is to use application servers, such as tomcat [Fou18] or glassfish [Ora18a].
- If you find lightweight servers, feel free to use. I have not tried them.

Java WebServices Primer

Deployment 2/4

Below is the steps to create war file and deploy it to tomcat using the command line.

1. Create war file using maven plugin [Pro18]

```
mvn install
..
..
[INFO] Webapp assembled in [92 msecs]
[INFO] Building war:
        /media/limmen/HDD/workspace/id2208/WebServicesScenarios/hw3/target/hw3.war
```

2. Copy war file to my TOMCAT installation (first removing previous deployed war)

```
rm -rf ~/programs/apache-tomcat-7.0.82/webapps/ROOT*
cp ~/workspace/id2208/WebServicesScenarios/hw3/target/hw3.war
        ~/programs/apache-tomcat-7.0.82/webapps/ROOT.war
```

Java WebServices Primer

Deployment 3/4

3. Start tomcat

   ./catalina.sh start

4. Test service with curl

   curl −H "Content−Type: application/json" −X POST −d
   '{"username":"kim","password":"id2208"}' http://localhost:8080/rest/login
   ID2208_AUTH_TOKEN

Java WebServices Primer

Deployment 4/4

You can also set this up in your IDE and skip the whole command-line!

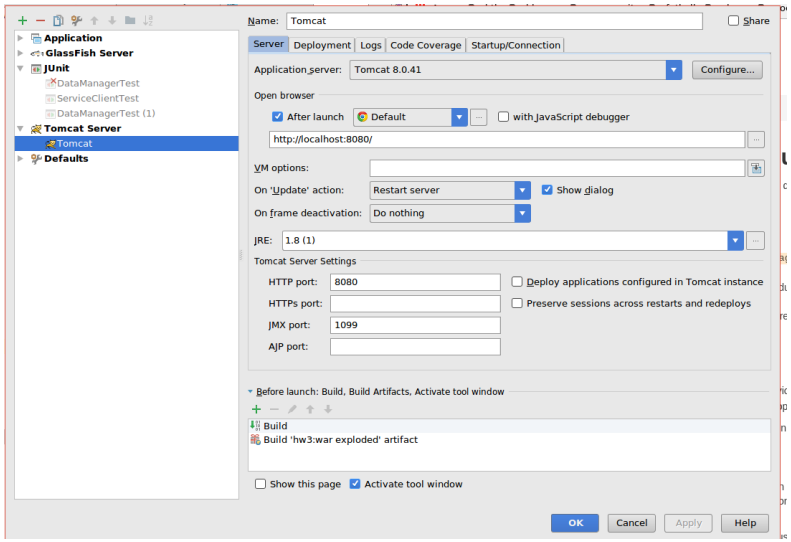


Figure: IntelliJ Tomcat configuration setup

Java WebServices Primer

Test client 1/2

- Jersey provides a test framework [fM18].
- Jersey also has a Client API [Pod09].
- You are free to use any type of client you want for testing.

Create a Client with Jersey Client API:

```
clientConfig = new DefaultClientConfig();
        clientConfig.getFeatures().put(JSONConfiguration.FEATURE_POJO_MAPPING, Boolean.TRUE);
        client = Client.create(clientConfig);
        webResource = client.resource("http://localhost:8080/rest");
```

Java WebServices Primer

Test client 2/2

Use Jersey Client to consume RESTful webservice

```
@Test
 public void itinerariesTest() {
      ClientResponse clientResponse = webResource.path("/itineraries").queryParam("token",
              SECRET_TOKEN).accept(MediaType.APPLICATION_XML).get(ClientResponse.class);
      Assert.assertEquals(200, clientResponse.getStatus());
      String response = webResource.path("/itineraries").queryParam("token",
              SECRET_TOKEN).accept(MediaType.APPLICATION_XML).get(String.class);
      Assert.assertEquals("<?xml version=\"1.0\" encoding=\"UTF−8\"
              standalone=\"yes\"?><itineraries><Itinerary><Flights><DepartmentCity>Stockholm</DepartmentCity><
              response);
      response = webResource.path("/itineraries").queryParam("token",
              SECRET_TOKEN).accept(MediaType.APPLICATION_JSON).get(String.class);
      Assert.assertEquals("[{\"Flights\":[{\"DepartmentCity\":\"Stockholm\",\"DestinationCity\":\"Paris\"}...]}]",
              response);
      ArrayList<Itinerary> itineraries = (ArrayList) webResource.path("/itineraries").queryParam("token",
              SECRET_TOKEN).accept(MediaType.APPLICATION_XML).get(new
              GenericType<List<Itinerary>>() {});
      Assert.assertEquals(7, itineraries.size());
```

Thank You and Good Luck!

References I

frodriguez MvnRepository, *Jersey test framework core*, https://mvnrepository.com/artifact/com.sun.jersey.jersey-test-framework/jersey-test-framework-core, 2018.

The Apache Software Foundation, *Apache tomcat*, http://tomcat.apache.org/, 2018.

References II


Oracle, *Glassfish*, https://javaee.github.io/glassfish/, 2018.


_____, *Jersey - restful web services in java*, https://jersey.github.io/, 2018.

References III

Jakub Podlesak, *Consuming restful web services with the jers y client api*, `https://blogs.oracle.com/enterprisetechtips/consuming-restful-web-services-with-the-jers` 2009.

References IV

Apache Maven Project, *Apache maven war plugin*, `https://maven.apache.org/plugins/maven-war-plugin/`, 2018.

References V

Qing Guo Yi Ming Huang with Dong Fei Wu, *Build a restful web service using jersey and apache tomcat*, `https://www.ibm.com/developerworks/web/library/wa-aj-tomcat/index.html`, 2009.

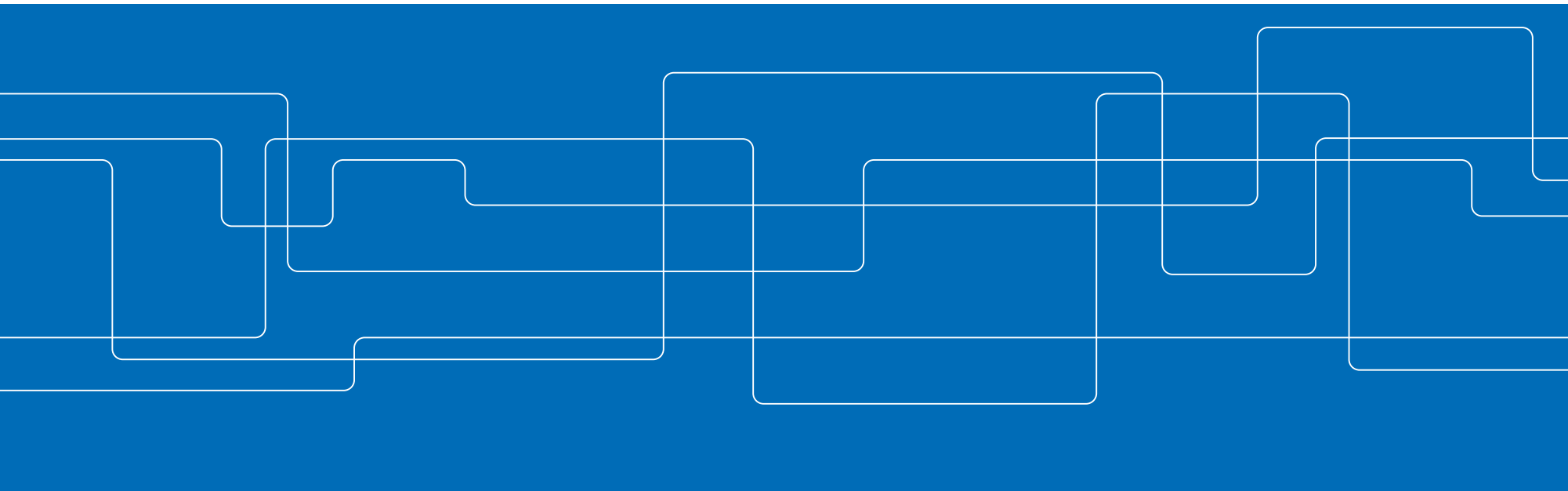
KTH ROYAL INSTITUTE
OF TECHNOLOGY

ID2208 Programming Web Services

Project 2018 - Semantic Web & Linked Open Data (LOD)

Kim Hammar (kimham@kth.se)
Cosar Ghandeharioon (cosarg@kth.se)
Mihhail Matskin (misha@kth.se)

February 13, 2018



Outline

Semantic Web & LOD Introduction

Topic Introduction 1/3

Why the Semantic Web?

> *To make the web more accessible to computers [AH08]*

Prior to the semantic web:

- Computer can index keywords
- Computer can tell syntactic difference between hyperlink and paragraph
- Most understanding is left to humans
- Structured data published with unstructured HTML

Semantic Web & LOD Introduction

Topic Introduction 2/3

Idea of the Semantic Web:

- Publish the data using **standardized data model** (RDF).
- **Link data** together with RDF triples.
- Add more **machine understandable semantics** (OWL).
- **Link semantics** between datasets (OWL Linking).
- Allow **semantic queries** to read the data (SPARQL).
- Reuse **XML only as a serialization format**.

Semantic Web & LOD Introduction

Topic Introduction 3/3

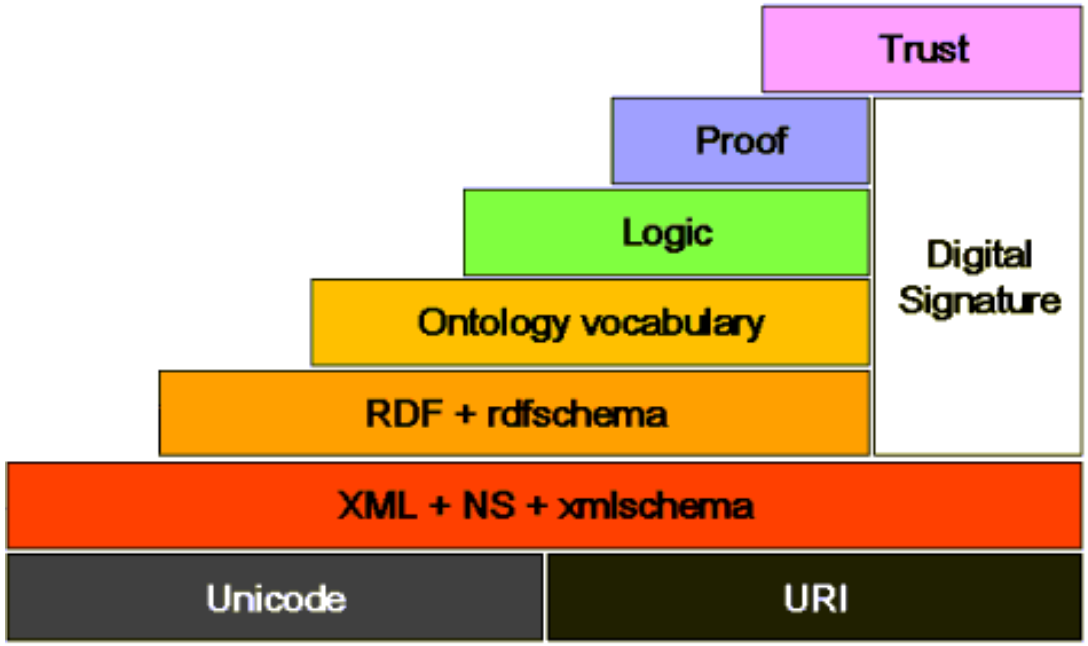


Figure: Semantic Web Technology Stack [W3C18]

Semantic Web & LOD Introduction

Why vanilla XML is only sufficient as a serialization format 1/2

```
    <PersonalInformation>
....
    </PersonalInformation>
```

What is PersonalInformation?

- Is it a Concept (Class)?
- Is it an Object of another class?
- Does it refer to the swedish word or the english word?
  $\Longrightarrow$ (**different meaning!**)

Semantic Web & LOD Introduction

Why XML is only sufficient as a serialization format 2/2

Added Semantics:

```
<rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22−rdf−syntax−ns#"
      xmlns:fl="http://www.limmen.kth.se/id2208/ontologies/persons#" >
   <rdf:Description rdf:about="http://www.limmen.kth.se/id2208/rdf/person#JohnDoe">
      <fl:FirstName>John</fl:FirstName>
.....
      <rdf:type rdf:resource="http://www.limmen.kth.se/id2208/rdf/person#Person"/>
   </rdf:Description>
</rdf:RDF>
```

- Add semantic $\Longrightarrow$ Link to an ontology
- Semantic annotation $\Longrightarrow$ Allows machine to look up meaning

Semantic Web & LOD Introduction

Linked Open Data (LOD) principles

- **Use URIs** to uniquely identify things (data entities).
- **Use HTTP URLs**, corresponding to these URIs $\Longrightarrow$ information can be retrieved.
- Provide meta-data using **open standards** such as RDF.
- **Include links** to related URIs $\Longrightarrow$ agents can discover more things.

Semantic Web & LOD Introduction

Shared Global Data Space 1/2

- A Semantic Web link **is typed** $\Longrightarrow$ Agent can look-up semantic.
- Different than hyperlink: **link concepts**, not documents.
- Typed link enables to **merge data from different domains** into a single graph.
- Huge web graph with links $\Longrightarrow$ agent can dereference the links to treat it as a **shared global data space**.

Semantic Web & LOD Introduction

Shared Global Data Space 2/2

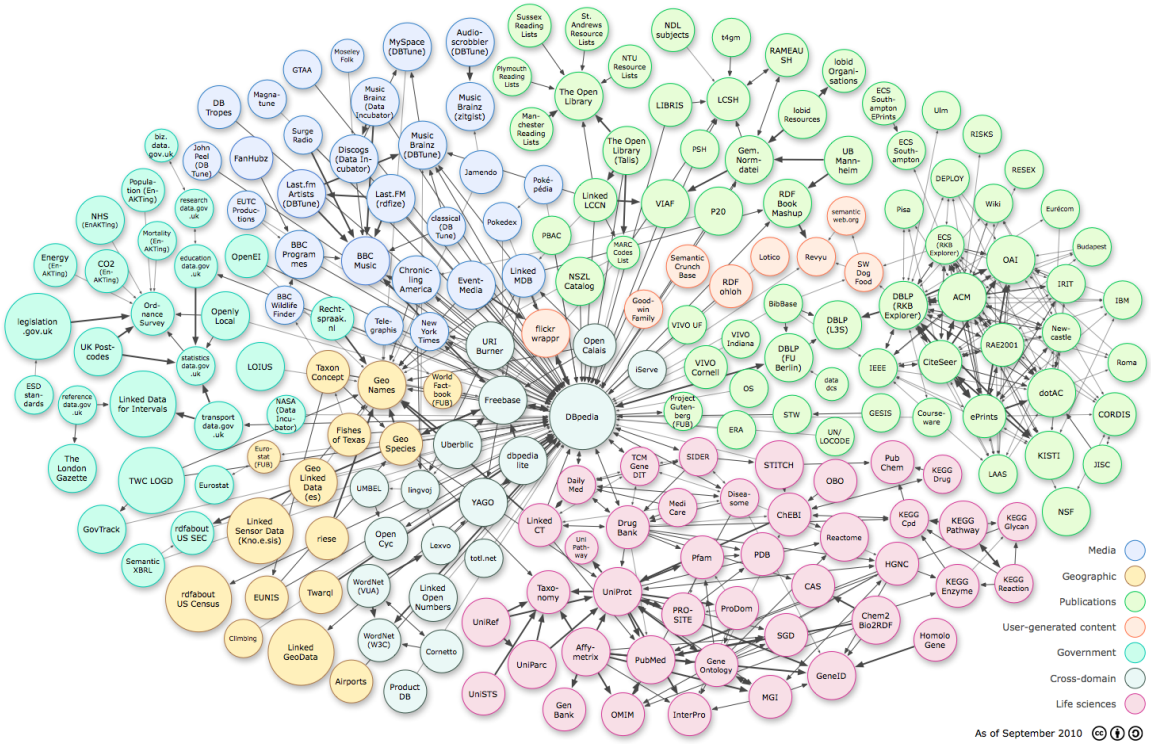


Figure: LOD Cloud November 2010 [HB11]

Semantic Web & LOD Introduction

Evolution of the web

1. Web 1.0: HTML pages with links
2. Web 2.0: HTML but also open APIs and Web services
3. Web 3.0: HTML + APIs but also LOD and semantics

Web 3.0 Goals:

- Flexible data browsing
- Accessible for software agents
- global data can grow in a *distributed fashion*.
- Link everything $\Longrightarrow$ whole web as a shared database.
- intelligent search

Semantic Web & LOD Introduction

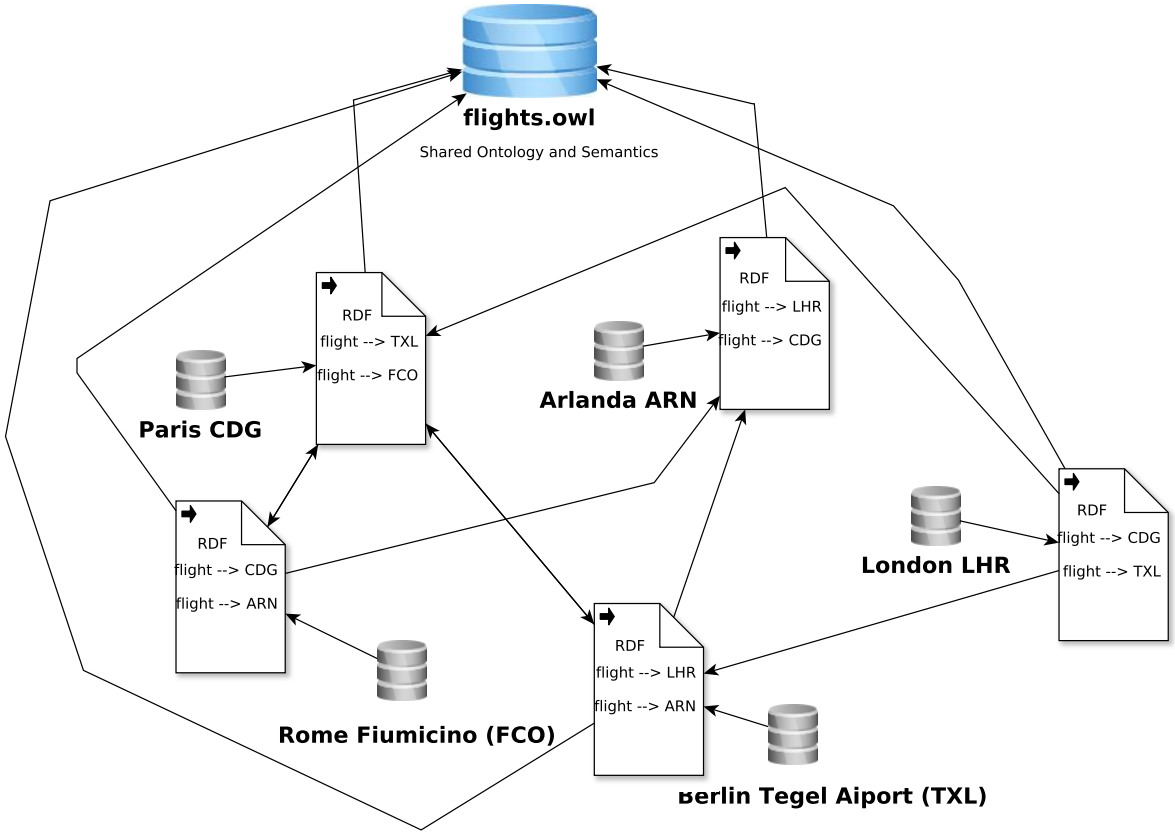


Figure: LOD Aiport Data

Semantic Web & LOD Introduction

Business Value?

- Obvious value for consumers
- **Not obvious for providers** - Maybe: simple HTML service can not be used by computer agents as easily, service might exclude possible clients
- Public data providers should be pioneers $\Longrightarrow$ Governments, wikipedia[1], medical etc.
- **Community Effort:** Linking Open Data (LOD) project anno 2007 [2]

[1] http://wiki.dbpedia.org/

[2] https://www.w3.org/wiki/SweoIG/TaskForces/CommunityPro
LinkingOpenData#Project_Description

Project 2018 - Semantic Web & LOD

Introduction

Project goals

- Learn the concepts of Semantic Web and LOD
- Get familiar with OWL and RDF
- Learn how to consume and make use of semantic web data

Project 2018 - Semantic Web & LOD

Problem Description

Design/Implement Semantic airport web service

1. One airport = one service/endpoint
2. Airports publish static RDF of their flights
3. Link RDF between airports and to external data
4. Airports use a shared ontology (typed links!)
5. Client/Agent fetch itineraries by following links

Project 2018 - Semantic Web & LOD

Tasks 2/2

- **Design ontology**: `flights.owl`
- Implement min **3 airport services/endpoints**
- Each airport service should have a URI where **RDF data of their departure flights** can be downloaded.
- Implement **Client/Agent** that provides *findItineraries*$(A_1, A_2) \rightarrow (I_1, I_2, \ldots, I_n)$ where A_i is a URI's of an airport and I_i is information about an itinerary.

Project 2018 - Semantic Web & LOD

Tasks 2/2

- Concepts/Classes: Flight, Airport, Airline...
- Flights are linked to airports
- Add some RDF metadata for each airport
- In one of your RDF documents, link to public data from DBPedia [DBP18]. E.g flight link to dbpedia entry for destination city
- Itinerary information: flights, airports, length, city..

Project 2018 - Semantic Web & LOD

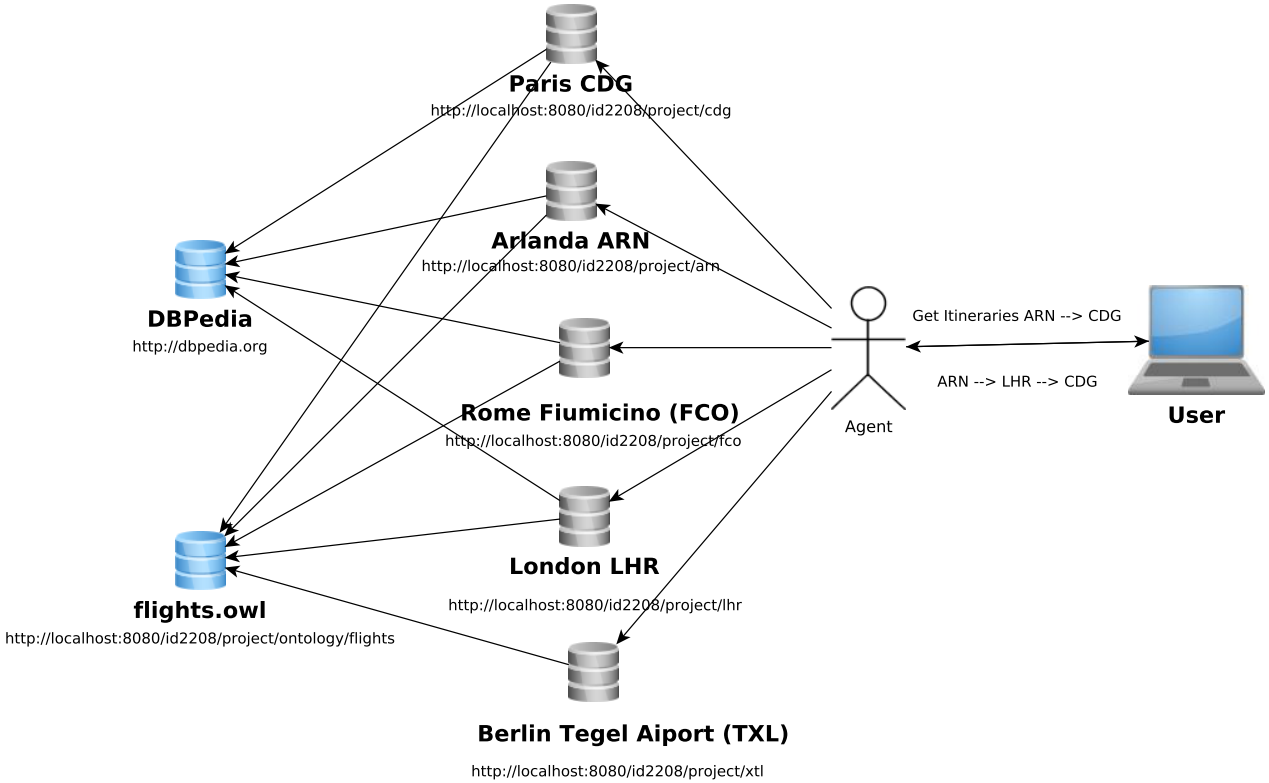


Figure: Agent follows links between airports to fetch itineraries and external data

Project 2018 - Semantic Web & LOD

Overview

Example output of agent for flights Arlanda → CDG

Finding all shortest−path itineraries from:
http://localhost:8080/kim/id2208/project/rdf/arlanda#ArlandaAirport to:
http://localhost:8080/kim/id2208/project/rdf/cdg#CDGAirport
##ITINERARY##
−−FLIGHT−−
FlightId: 2 | with airline: http://dbpedia.org/data/resource/Transavia, Transavia Airlines C.V., trading as Transavia and formerly branded as transavia.com, is a Dutch low−cost airline and ...
From Airport: http://localhost:8080/kim/id2208/project/rdf/arlanda#ArlandaAirport which is close to city: http://dbpedia.org/data/Stockholm.rdf, Stockholm is the capital of Sweden and the most populous city in the Nordic countries; 925,934 people live in the municipality....
To Airport: http://localhost:8080/kim/id2208/project/rdf/heathrow#HeathrowAirport
−−FLIGHT−−
FlightId: 3 | with airline: http://dbpedia.org/data/resource/Air_Peru, Air Peru International was a planned Peruvian airline to be based in Lima, Peru. It planned to operate ...
From Airport: http://localhost:8080/kim/id2208/project/rdf/heathrow#HeathrowAirport which is close to city: http://dbpedia.org/data/London.rdf, London is the capital and most populous city of England and the United Kingdom...
To Airport: http://localhost:8080/kim/id2208/project/rdf/cdg#CDGAirport
Closest City to final destination: http://dbpedia.org/data/Paris.rdf, Paris is the capital and the most populous city of France...

Project 2018 - Semantic Web & LOD

Deliverables

1. **Ontology** FLIGHTS.OWL[3].
2. **RDF data**, minimum 1 flight per airport, 1 itinerary of length 3.
3. **Source code** for the airport services (can be one service with 4 endpoints) and client
4. **Report** describing what you did
5. **Presentation** - demonstrate code and answer questions

[3] The required data will result in a quite small ontology and RDF-files, we encourage you to add more data if you want!

Semantic Web Tooling Primer

Some Links 1/2

- DBPedia [DBP18]. Browse some ontologies and RDF data to get inspiration.
- Semantic Web Primer (book) [AH08].
- Linked Data Book [HB11] (free online).
- **Programming the Semantic Web tutorial (ID2208) With Source Code Examples** [Ham18].
- Pizza.owl (Example ontology) [PD18].

Semantic Web Tooling Primer

Some Links 2/2

- Apache Jena - Java framework for Semantic Web [Fou18b]
- Apache Tomcat [Fou18a], Glassfish Application Server [Ora18a], Jersey [Ora18b]
- A Practical Guide To Building OWL Ontologies (Available free PDF) [HKR+04].
- Protege (tool for building ontologies, recommended) [Pro18].

Semantic Web Tooling Primer

URI and URLs

- URI of your resources will be your service URL (e.g `http://localhost:8080/kim/id2208/project/rdf/arlanda#ArlandaAirport`)
- Remember: We want to link concepts $\Longrightarrow$ need to link to **parts of documents**
- Hash URI strategy[4]: Fragment part and Document URL part, separated by **#**
- **#** is not part of the HTTP request, it is just symbolic

[4] An alternative to hash strategy is 303 HTTP code strategy

Semantic Web Tooling Primer

Ontologies 1/3

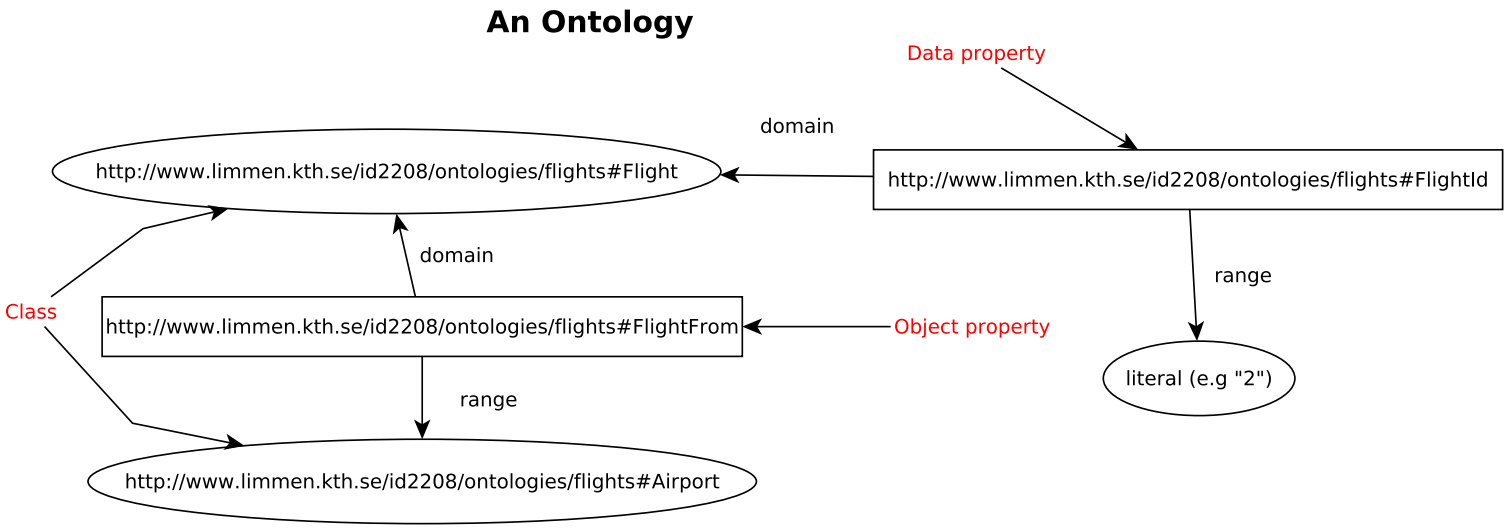


Figure: Graphical Representation of a simple ontology

Semantic Web Tooling Primer

Ontologies 2/3

Ontology describes a domain, a taxonomy[5]. Example:

- **Class hierarchies** (Child subclass of Person)
- **Data properties** (associate classes $\rightarrow$ data)
- **Object properties** (associate class $\rightarrow$ class)
- **Meta-data**
- **Linking** with other ontologies
- **Assertions** (owl:sameAs)

[5] See tutorial [HKR+04]. OWL is powerful, you will only need subset for this project

Semantic Web Tooling Primer

Ontologies 3/3

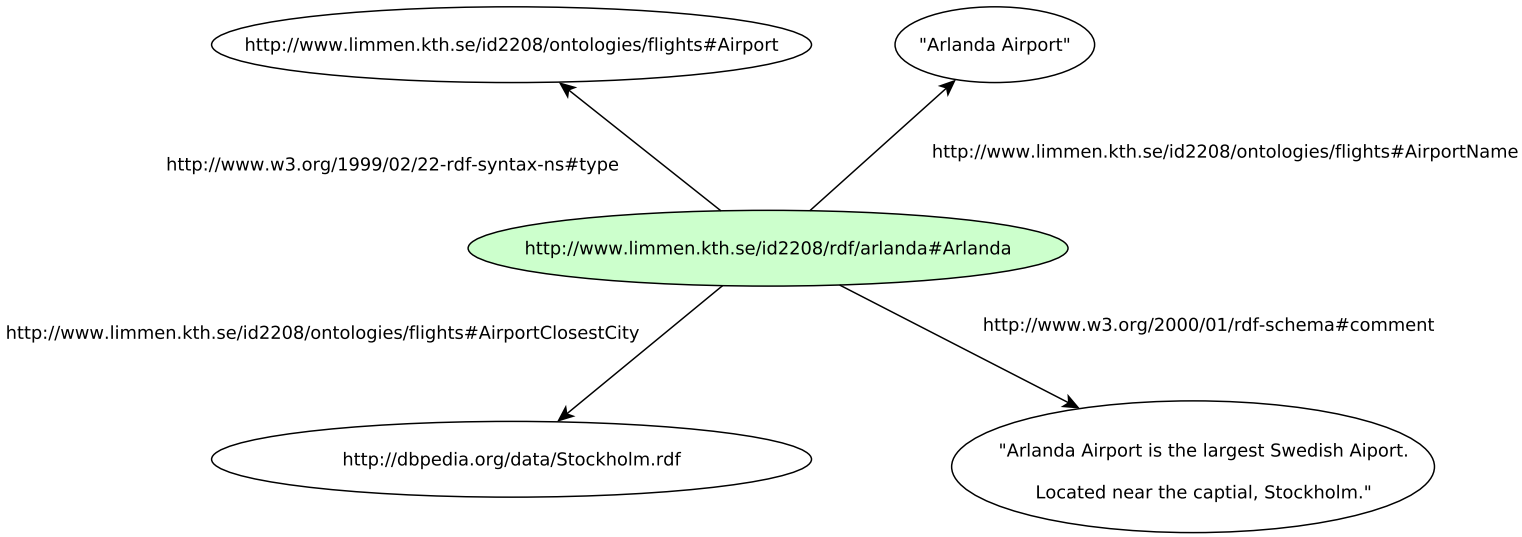
OWL can be serialized in different formats, e.g RDF/XML.

```
<owl:DatatypeProperty rdf:about="http://www.limmen.kth.se/id2208/ontologies/flights#FlightLength">
    <rdfs:subPropertyOf
            rdf:resource="http://www.limmen.kth.se/id2208/ontologies/flights#FlightDataProperties"/>
    <rdfs:domain>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.w3.org/2002/07/owl#topObjectProperty"/>
            <owl:someValuesFrom
                    rdf:resource="http://www.limmen.kth.se/id2208/ontologies/flights#Flight"/>
        </owl:Restriction>
    </rdfs:domain>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#double"/>
</owl:DatatypeProperty>
```

Semantic Web Tooling Primer

RDF

RDF is the basic data model, describe resource with triples. Notice below the linking to an ontology and to an external dataset (dbpedia).



Semantic Web Tooling Primer

RDF

Many serializations of RDF & OWL, one of them is RDF/XML.

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22−rdf−syntax−ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf−schema#"
    xmlns:fl="http://www.limmen.kth.se/id2208/ontologies/flights#" >
  <rdf:Description rdf:about="http://www.limmen.kth.se/id2208/rdf/arlanda#Arlanda">
    <fl:AirportName>Arlanda Airport</fl:AirportName>
    <fl:AirportClosestCity>http://dbpedia.org/resource/Stockholm</fl:AirportClosestCity>
    <rdfs:comment>Arlanda Airport is the largest Swedish Aiport, located in the capital,
          Stockholm</rdfs:comment>
    <rdf:type rdf:resource="http://www.limmen.kth.se/id2208/ontologies/flights#Airport"/>
    </rdf:Description>
    ...
    </rdf:RDF>
```

The `Description` tag defines a resource with the “about” attribute. The `RDF` tag is the root tag.

Semantic Web Tooling Primer

SPARQL

SPARQL is a query language for RDF data $\approx$ SQL, don't need it for this project if you don't want [6].

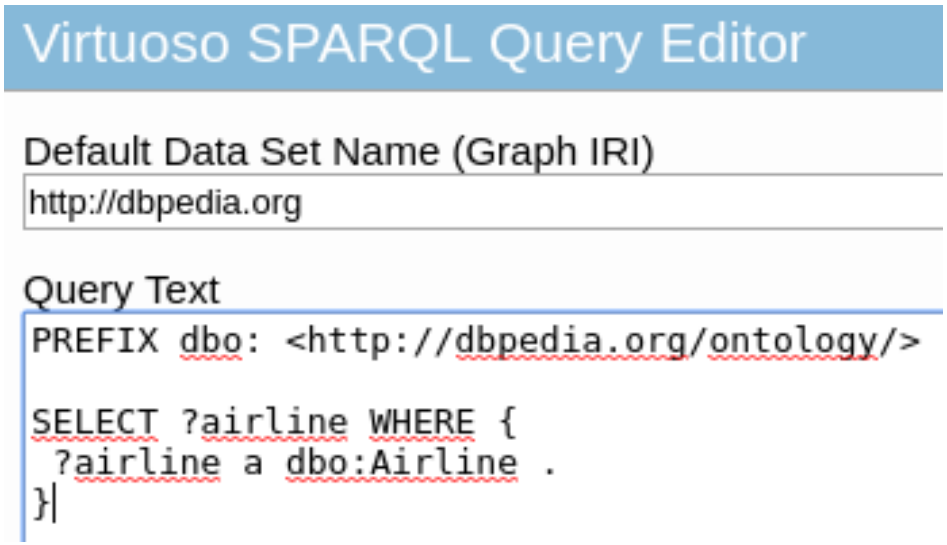


Figure: SPARQL query to fetch a list of Airlines form DBPedia from `https://dbpedia.org/sparql`

[6] In real-world application your airports would have SPARQL endpoints instead of static RDF endpoints

Semantic Web Tooling Primer

Logic

Logic interpretation of ontology: Knowledge base with terminology definitions and assertions. **OWL is based on description logic**[7]

```
<owl:Class rdf:about="Father">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:complementOf>
      <owl:Class rdf:about="Woman"/>
    </owl:complementOf>
    <owl:Class rdf:about="Parent"/>
  </owl:intersectionOf>
</owl:Class>
```

Description logic equivalent to the OWL snippet:

$$Father \equiv \neg Woman \cap Parent$$

[7] Description logic is a subset of first-order logic. Description logic makes the open-world assumption.

Semantic Web Tooling Primer

Protege

Recommended tool for creating ontologies through GUI: Protege[Pro18]

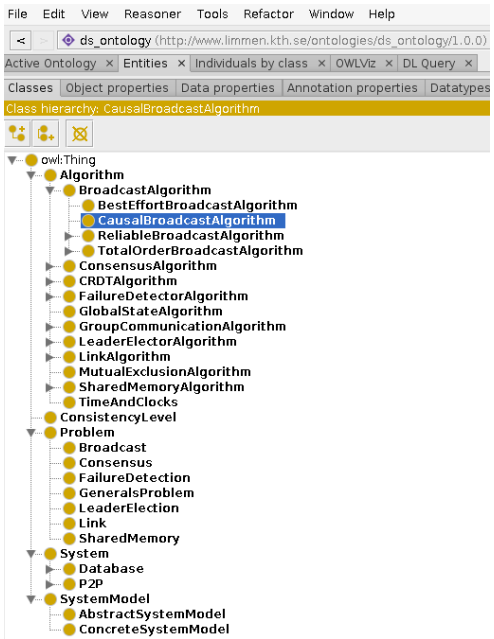


Figure: Protege

Semantic Web Tooling Primer

Apache Jena 1/4

Apache Jena: Processing RDF/OWL in java

Select RDFNodes of particular RDF-type from a model using Jena:

```
ArrayList<RDFNode> flights = new ArrayList();
Selector flightsSelector = new SimpleSelector(null, RDF.type,
        ontModel.getOntClass(FlightsOntology.Flight));
StmtIterator stmtIterator = model.listStatements(flightsSelector);
while (stmtIterator.hasNext()) {
   flights.add(stmtIterator.nextStatement().getSubject());
}
```

Semantic Web Tooling Primer

Apache Jena 2/5

Generate RDF document using Jena

```
Resource airport = rdfModel.createResource(ns+airportName);
airport.addProperty(RDF.type, ontModel.getOntClass(FlightsOntology.Airport));
airport.addProperty(RDFS.comment, airportComment);
airport.addProperty(ontModel.getProperty(FlightsOntology.AirportClosestCity), airportClosestCity);
airport.addProperty(ontModel.getProperty(FlightsOntology.AirPortName), airportName);
model.write(System.out, "RDF/XML");
```

Note that FlightsOntology in the code snippet is just a class holding static String constants for the URI's in the ontology.

Semantic Web Tooling Primer

Apache Jena 3/5

Load Ontology using Jena

```
public static OntModel readOntology(String path) {
        OntDocumentManager ontDocumentManager = new OntDocumentManager();
        OntModelSpec ontModelSpec = new OntModelSpec(OntModelSpec.OWL_MEM);
        ontModelSpec.setDocumentManager(ontDocumentManager);
        OntModel ontModel = ModelFactory.createOntologyModel(ontModelSpec, null);
        try {
            ontModel.read(new ByteArrayInputStream(DataUtils.readResource(path,
                    Charsets.UTF_8).getBytes()), "RDF/XML");
        } catch (IOException e) {
            throw new IllegalArgumentException("File: " + path + " not found");
        };
        return ontModel;
    }
```

Semantic Web Tooling Primer

Apache Jena 4/5

SPARQL query with Jena

```
public static ArrayList<String> fetchAirlines() {
    String queryString = "PREFIX dbo: <http://dbpedia.org/ontology/> \n" +
            "SELECT ?airline WHERE {\n" +
            " ?airline a dbo:Airline .\n" +
            "}";
    Query query = QueryFactory.create(queryString);
    String service = "http://dbpedia.org/sparql";
    QueryEngineHTTP serviceRequest = QueryExecutionFactory.createServiceRequest(service,
            query);
    ResultSet results = serviceRequest.execSelect();
    ArrayList<String> airlines = new ArrayList();
    while (results.hasNext()) {
        QuerySolution querySolution = results.nextSolution();
        airlines.add(DBPediaResourceToRdf(querySolution.getResource("airline").toString()));
    }
    return airlines;
}
```

Semantic Web Tooling Primer

Apache Jena 5/5

Load resource from URL into Jena

```
Model cityModel = ModelFactory.createDefaultModel();
cityModel.read("http://dbpedia.org/data/Stockholm.rdf");
NodeIterator nodeIterator = cityModel.listObjectsOfProperty(RDFS.comment);
Literal comment = null;
while (nodeIterator.hasNext() && comment == null) {
   Literal c = nodeIterator.nextNode().asLiteral();
   if (c.getLanguage().equals("en"))
   comment = c;
}
return comment;
```

The code above returns the English RDF-comment from `http://dbpedia.org/data/Stockholm.rdf`. **Note:** Jena might not be able to load URL's if they point to HTML format, e.g `http://dbpedia.org/page/Stockholm`, so make sure you use the /data API

Semantic Web Tooling Primer

Deployment

For deployment you can use any setup you like that can serve serve static RDF file, your endpoints can be very basic, see below.

```
@GET
@Produces("application/rdf+xml")
public String arlanda() {
   StringWriter stringWriter = new StringWriter();
   dataMngr.getRdfModel().write(stringWriter, "RDF/XML");
   return stringWriter.toString();
}
```

Tip: Reuse server-code from HW2 or HW3 Example endpoint:

```
 curl http://localhost:8080/kim/id2208/project/rdf/arlanda#ArlandaAirport
<rdf:RDF
...
   <fl:Flight rdf:about="http://localhost:8080/kim/id2208/project/rdf/arlanda#FL1">
...
```

Future Work

If you are interested how this application can be extended

- Add **SPARQL endpoints**
- Add **HTML format for humans** to read the data, and return correct format based on a content-negotiation
- Use **triple store** instead of in-memory representation of data
- Use a semantic **reasoner** to reason about the data
- **Semantic XML-based WS**: SAWSDL & OWL-S

DEMO

(If there is interest and we have time)

Thank You and Good Luck!

References I

Grigoris Antoniou and Frank van Harmelen, *A semantic web primer, 2nd edition (cooperative information systems)*, 2 ed., The MIT Press, 2008.

DBPedia, *Dbpedia*, `http://wiki.dbpedia.org/`, 2018.

References II


The Apache Software Foundation, *Apache tomcat*, `http://tomcat.apache.org/`, 2018.


_____, *A free and open source java framework for building semantic web and linked data applications.*, `https://jena.apache.org/`, 2018.

References III

Kim Hammar, *Programming the semantic web - a tutorial*, `TODO`, 2018.

Tom Heath and Christian Bizer, *Linked data: Evolving the web into a global data space*, 1st ed., Morgan & Claypool, 2011.

References IV

Matthew Horridge, Holger Knublauch, Alan Rector, Robert Stevens, and Chris Wroe, *A practical guide to building owl ontologies using the prot'eg'e-owl plugin and co-ode tools*, 08 2004.

Oracle, *Glassfish*, `https://javaee.github.io/glassfish/`, 2018.

References V


_____, *Jersey - restful web services in java*, https://jersey.github.io/, 2018.


Protege and Nick Drummond, *The semantic web made easy*, https://protege.stanford.edu/ontologies/pizza/pizza.owl, 2018.

References VI

Protege, *A free, open-source ontology editor and framework for building intelligent systems*, `https://protege.stanford.edu/`, 2018.

W3C, *The semantic web made easy*, `https://www.w3.org/RDF/Metalog/docs/sw-easy`, 2018.