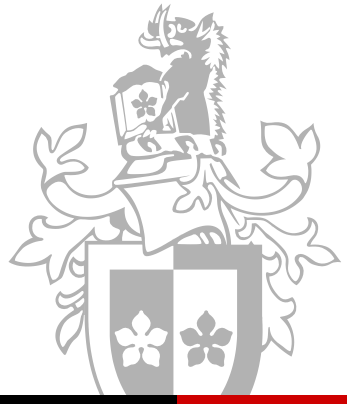
COS30041 Creating Secure and Scalable Software

Lecture 09a Traditional (“Big”) Web Services



SWINBURNE UNIVERSITY OF TECHNOLOGY

Learning Objectives

- After studying the lecture material, you will be able to
  - Understand and describe what Traditional ("Big") Web Services are
  - Understand and describe the advantages and disadvantages of "Big" Web Services
  - Understand the issues involved in programming the "Big" Web Services
  - Program the "Big" Web Services via JAX-WS
  - Program client applications that call the "Big" Web services
  - Understand and describe the similarities and differences between CORBA and "Big" Web Services

Pre-requisite

- Object Oriented Programming
- Some experiences on XML would be an advantage

Outline

- Traditional (“Big”) Web Services
- CORBA versus “Big” Web Services
- Programming “Big” Web Services
  - Server side component
  - Client program

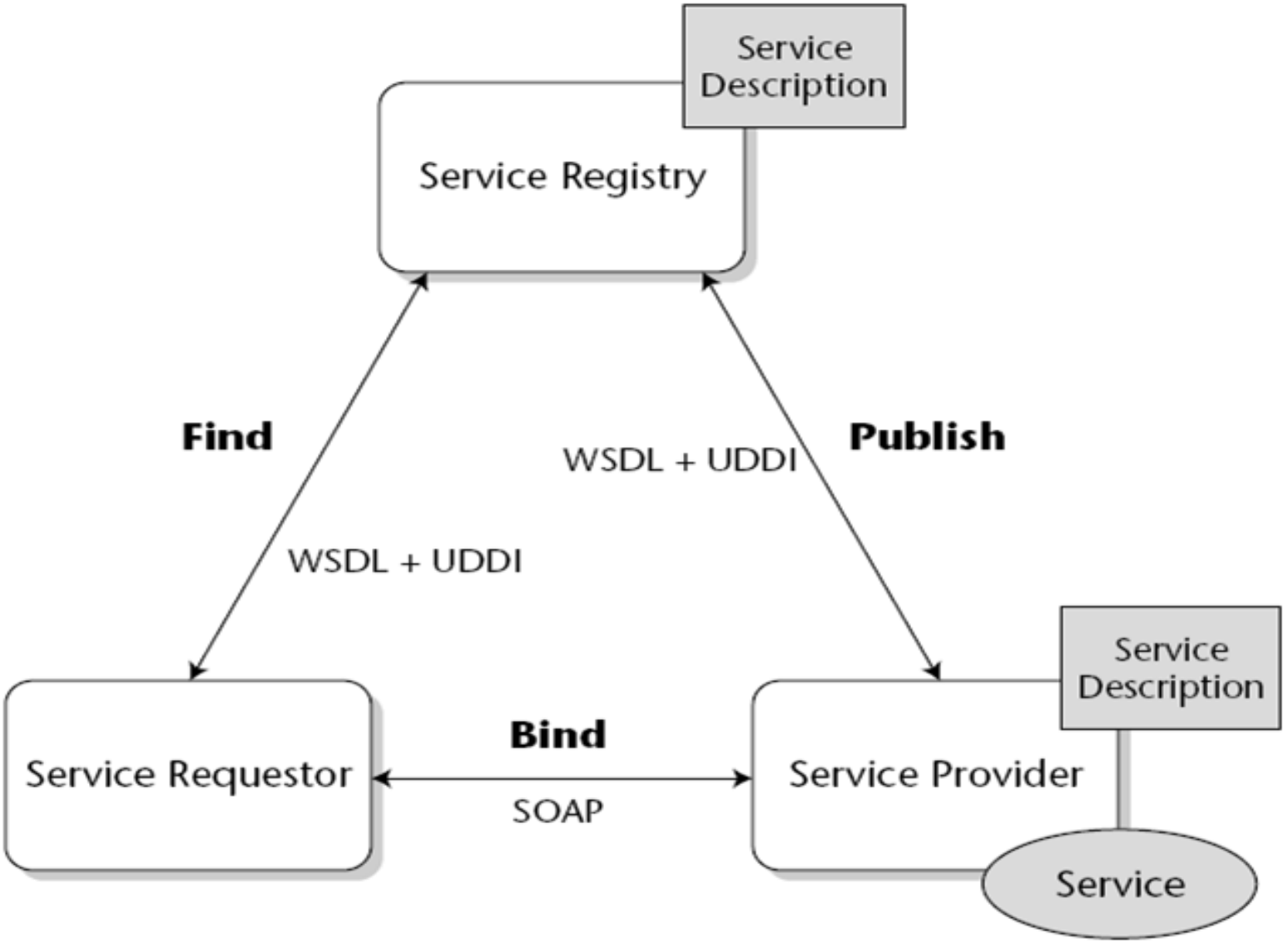
Roadmap

- **Traditional ("Big") Web Services**
- CORBA versus "Big" Web Services
- Programming "Big" Web Services
  - Server side component
  - Client program

Traditional Web Services

- “Big” Web Services
- A way to build and integrate large-scale systems within and between companies by sending XML messages to well-defined, modular interfaces
- Different software components are communicated via XML messages
- Using WSDL (Web Services Description Language)
- Using UDDI (Uniform Description, Discovery, and Integration)
- Using SOAP (Simple Object Access Protocol)

Web Services (cont'd)



From Fig. 5.1 of [MEJB3,p.116]

Web Servcies (cont'd)

- Service provider
  - Software component that provide the web services
- Service requestor
  - Software component that request for certain web services
- Service registry
  - A place for service provider to publish its web services via the abstract service definitions
  - A place for service requester to search for the services that it wants

Web Services Description Language, WSDL

- The language used to describe the web services
- Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="HelloWorldWS" targetNamespace="urn:examples">
<types/>
   <message name="HelloInterface_hello"/>
   <message name="HelloInterface_helloResponse">
      <part name="result" type="xsd:string"/>
   </message>
   <portType name="HelloInterface">
      <operation name="hello">
         <input message="tns:HelloInterface_hello"/>
         <output message="tns:HelloInterface_helloResponse"/>
      </operation>
   </portType>
   <binding name="HelloInterfaceBinding" type="tns:HelloInterface">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/
http" style="rpc"/>
      <operation name="hello">
         <soap:operation soapAction=""/>
```

Universal Description, Discovery, and Integration, UDDI

- A standard for the service registry to communicate with
  - The service requestor for searching the services required
  - The service provider for the services being provided
- If developers know where the web services are, they do not need to use UDDI to search the required web services
- Dynamic service lookups in UDDI registry is still unsure for the time being

Simple Object Access Protocol, SOAP

- The service requestor and the service provider are communicated using the SOAP
- Example: (from service requestor)

```
POST /HelloBean HTTP/1.1
Content-Type: text/xml; charset="utf-8"
Content-Length: 398
SOAPAction: ""
Host: falcon:8080

<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
   <env:Body>
      <ans1:hello xmlns:ans1="urn:examples"/>
   </env:Body>
</env:Envelope>
```

Simple Object Access Protocol, SOAP (cont'd)

- Example: (from service provider)

```
HTTP/1.1 200 OK
SOAPAction: ""
Content-Type: text/xml;charset=utf-8
Transfer-Encoding: chunked

<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:ns0="urn:examples">
   <env:Body>
      <ns0:helloResponse>
         <result xsi:type="xsd:string">Hello, World!</result>
      </ns0:helloResponse>
   </env:Body>
</env:Envelope>
```

Roadmap

- Web Services
- **CORBA versus “Big” Web Services**
- Programming “Big” Web Services
  - Server side component
  - Client side component

Comparing CORBA and “Big” Web Services

	CORBA	“Big” WS
Cross platform	Yes	
Language	Neutral	
Interface language	IDL	WSDL
Communication protocol	IIOP	SOAP (/HTTP)

Advantages of “Big” Web Services vs CORBA

- Encourage
  - modularity through standardized interfaces
  - flexibility through loose coupling
  - extensibility through using XML
- Language neutral
- SOAP (vs CORBA’s IIOP) is lightweight in terms of
  - simple to use
  - does not require many assumptions about the behaviour of clients and servers

Drawbacks of “Big” Web Services vs CORBA

- Scalability of Web Services has not been properly addressed
- SOAP has large overheads
  - as compared with communication protocol in binary form, say, IIOP in CORBA
  - when parsing larger XML messages

Roadmap

- Web Services
- CORBA versus “Big” Web Services
- **Programming “Big” Web Services**
  - Server side component
  - Client program

Programming “Big” Web Services

- No DTO anymore
- Communication is via SOAP / HTTP (text-based)
- Only values are passed to and from server via XML
- Primitive data types are good (no extra work)
  - ☐ boolean, char, double, float, int
- Objects needs to be “converted” to and from XML [programmer to do]
  - ☐ String is an exception
- [Client] Objects →XML >>SOAP>> XML →Object [Server]

Roadmap

- Web Services
- CORBA versus “Big” Web Services
- Programming “Big” Web Services
  - **Server side component**
  - Client program

Prog. “Big” WS using JAX-WS

- Things to note
  - Where should the web services modules sit? Web / EIS Tier
    - Is that obvious?
  - How do client and server communicate?
    - SOAP on top of HTTP (basically text)
  - Any DTO to pass between server and client?
    - Is that obvious?

Prog. “Big” WS using JAX-WS – Server side

- For Web Services Java Class
  - @WebService(serviceName = “XxxxWS”) – declare a web service class
    - @WebService(serviceName = “MyuserFacadeWS”)
      public class MyuserFacadeWS
- For Web Service Methods in Web Service Java Class
  - @WebMethod(operationName = “yyyyZzzz”) – declare a method
  - @WebParam(name = “vvvv”) – declare a parameter from HTTP request
    - @WebMethod(operationName = “getRecordsByAddress”)
      public List<MyuserDTO> getRecordsByAddress(
      @WebParam(name = “address”) String address)

Example (server side) – MyuserFacadeWS

```java
/**
 *
 * @author elau
 */
@WebService(serviceName = "MyuserFacadeWS")
public class MyuserFacadeWS {

    // a reference to Stateless EJB, so no actual coding here
    // Alternatively, put the code here, no need to refer to the EJB
    @EJB
    private MyuserFacadeRemote ejbRef;

    @WebMethod(operationName = "createRecord")
    public boolean createRecord(@WebParam(name = "myuserDTO") MyuserDTO myuserDTO) {
        return ejbRef.createRecord(myuserDTO);
    }

    @WebMethod(operationName = "getRecord")
    public MyuserDTO getRecord(@WebParam(name = "userid") String userid) {
        return ejbRef.getRecord(userid);
    }
```

Prog. “Big” WS using JAX-WS – Parameter Passing

- Where do such parameters (primitive data or Java object) sit?
- Primitive data types – Easy
  - No need to do extra work
- Java Object / User-defined Class [Type] – More work
  - Convert “Object” to XML [“printing values of instance variables in XML format”]
  - Convert XML to “Object” [“reading variable values in XML format, create an object and call the getters, if necessary”]
  - JAXB does all these for you NOW

How does JAXB work?

- @XmlRootElement – define the name of the Java Object
  - @XmlRootElement(name = “MyuserDTO”)

  Note: createRecord(@WebParam(name = “**m**yuserDTO”) MyuserDTO myDTO)
- @XmlAccessorType – define the accessor type of the Java Object
  - @XmlAccessorType(XmlAccessType.FIELD)
- @XmlElement(required = true) – specify a field / an instance variable is a must in the XML doc
  - @XmlElement(required = true)
    private String userid;

Example (Passing Object, JAXB) – MyuserDTO

```java
/**
 *
 * @author elau
 */
@XmlRootElement(name = "MyuserDTO")
@XmlAccessorType(XmlAccessType.FIELD)
public class MyuserDTO implements Serializable {

    @XmlElement(required = true)
    private String userid;
    @XmlElement(required = true)
    private String name;
    @XmlElement(required = true)
    private String password;
    @XmlElement(required = true)
    private String email;
    @XmlElement(required = true)
    private String phone;
    @XmlElement(required = true)
    private String address;
    @XmlElement(required = true)
    private String secQn;
    @XmlElement(required = true)
    private String secAns;

    public MyuserDTO () {
    }

    public MyuserDTO(String userid, String name, String password,
```

Roadmap

- Web Services
- CORBA versus “Big” Web Services
- Programming “Big” Web Services
  - Server side component
  - **Client program**

Programming “Big” WS client using JAX-WS

- Within the same “enterprise”, no need to use UDDI to discover the WSDL
- Use @WebServiceRef(wsdlLocation = “http://…”) to get a reference to the web service object
  - @WebServiceRef(wsdlLocation = “http://localhost:…”)
    private static MyuserFacadeWS_Service myService;
- Use “Service.getXxxxPort()” to get the actual port of the web service object
  - MyuserFacadeWS port = myService.getMyuserFacadeWSPort();
- Call the web service method via the “port” object
  - port.createRecord(myuserDTO);

Example – WSDL of MyuserFacadeWS

```
69      <service name="MyuserFacadeWS">
70          <port name="MyuserFacadeWSPort" binding="tns:MyuserFacadeWSPortBinding">
71              <soap:address location="http://localhost:8080/ED-JEE-DTO-war/MyuserFacadeWS"/>
72          </port>
73      </service>
39      <binding name="MyuserFacadeWSPortBinding" type="tns:MyuserFacadeWS">
40          <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
41          <operation name="getRecordsByAddress">
42              <soap:operation soapAction=""/>
43              <input>
44                  <soap:body use="literal"/>
45              </input>
46              <output>
47                  <soap:body use="literal"/>
48              </output>
49          </operation>
50          <operation name="createRecord">
51              <soap:operation soapAction=""/>
52              <input>
53                  <soap:body use="literal"/>
54              </input>
55              <output>
56                  <soap:body use="literal"/>
57              </output>
58          </operation>
59          <operation name="getRecord">
60              <soap:operation soapAction=""/>
61              <input>
62                  <soap:body use="literal"/>
63              </input>
64              <output>
```

Example – WSDL of MyuserFacadeWS (cont'd)

```
25    <portType name="MyuserFacadeWS">
26        <operation name="getRecordsByAddress">
27            <input wsam:Action="http://ws/MyuserFacadeWS/getRecordsByAddressRequest" message="tns:getRecordsByAddress"/>
28            <output wsam:Action="http://ws/MyuserFacadeWS/getRecordsByAddressResponse" message="tns:getRecordsByAddressResponse"/>
29        </operation>
30        <operation name="createRecord">
31            <input wsam:Action="http://ws/MyuserFacadeWS/createRecordRequest" message="tns:createRecord"/>
32            <output wsam:Action="http://ws/MyuserFacadeWS/createRecordResponse" message="tns:createRecordResponse"/>
33        </operation>
34        <operation name="getRecord">
35            <input wsam:Action="http://ws/MyuserFacadeWS/getRecordRequest" message="tns:getRecord"/>
36            <output wsam:Action="http://ws/MyuserFacadeWS/getRecordResponse" message="tns:getRecordResponse"/>
37        </operation>
38    </portType>
```

```
7     <message name="getRecordsByAddress">
8         <part name="parameters" element="tns:getRecordsByAddress"/>
9     </message>
10    <message name="getRecordsByAddressResponse">
11        <part name="parameters" element="tns:getRecordsByAddressResponse"/>
12    </message>
13    <message name="createRecord">
14        <part name="parameters" element="tns:createRecord"/>
15    </message>
16    <message name="createRecordResponse">
17        <part name="parameters" element="tns:createRecordResponse"/>
18    </message>
19    <message name="getRecord">
20        <part name="parameters" element="tns:getRecord"/>
21    </message>
22    <message name="getRecordResponse">
23        <part name="parameters" element="tns:getRecordResponse"/>
24    </message>
```

Example – WSDL of MyuserFacadeWS (cont'd)

```
2    <types>
3        <xsd:schema>
4            <xsd:import namespace="http://ws/" schemaLocation="http://localhost:8080/ED-JEE-DTO-war/MyuserFacadeWS?xsd=1"/>
5        </xsd:schema>
6    </types>
```

Example (Client side) – MyuserWSAppClient

```
/**
 *
 * @author elau
 */
public class MyuserWSAppClient {

    @WebServiceRef(wsdlLocation = "http://localhost:8080/ED-JEE-DTO-war/MyuserFacadeWS?wsdl")
    private static MyuserFacadeWS_Service service;

    /**...3 lines */
    public static void main(String[] args) {...22 lines }

    public void createRecord(MyuserDTO myuserDTO) {
        MyuserFacadeWS port = service.getMyuserFacadeWSPort();
        boolean result = port.createRecord(myuserDTO);

        if (result) {
            System.out.println("Record " + myuserDTO.getUserid() + " has been added to database.");
        } else {
            System.out.println("Record " + myuserDTO.getUserid() + " cannot be added to database.");
        }

        return;
    }

    public void getRecord(String userid) {
        MyuserFacadeWS port = service.getMyuserFacadeWSPort();
        MyuserDTO myuserDTO = port.getRecord(userid);

        this.displayMyuserDTOInfo(myuserDTO);
    }
```

References

- [MEJB3] R.P. Sriganesh, G. Brose, M. Silverman (2008) *Mastering Enterprise JavaBeans 3.0*, 4th ed., John Wiley & Sons – Chapter 5
- Java EE Tutorial