**COMP30220 Distributed Systems Practical**

**Lab 3: SOAP/HTTP-based Distribution**

Work individually. Submit your code on csmoodle.ucd.ie by the deadline given on moodle. Please submit a single ZIP file (name is <student-num>.zip) containing your source code and any text files required for the solution.

This practical again uses the QuoCo scenario. You should start by redownloading the original source code. Do not attempt to adapt your RMI based solution. The broad objective of this practical is to adapt the code provided to use SOAP/HTTP for interaction between each of the 4 services and between the broker and the client. I have chosen to do this is a piece-meal way. First working on the client / broker interface, and then on the broker / quotation service interface. As a final step the services should be discoverable using UDDI.

**Task 1: Splitting the Client & Broker** **Grade: D**

The first task is to use SOAP for interaction between the client and the broker components. We will continue to use the existing Registry for broker-quotation service interaction.

a) The first thing we need to do is to change the Broker service code to be a web service. In preparation, you should look over the Stock Quote code from the lecture notes. Here you should see that the StockService interface uses annotations to declare a web service. Specifically, the class should be annotated as a `@WebService` and each method should be annotated as `@WebMethod`. You should modify the Broker service interface to reflect this.

b) The next thing you need to do is to modify the implementation of the Broker Service. Again, using the StockService to guide you, you should add @WebService(…) and @SOAPBinding(…) annotations to the LocalBrokerService class.

The @WebService annotation Should take the form:

```
@WebService(
        serviceName="BrokerService",
        targetNamespace="http://core.service/",
        portName="BrokerServicePort"
)
```

This is more complex than the example in the slides because the BrokerService interface is in a different class to the LocalBrokerService implementation. The targetNamespace parameter is based on an inverted package name for the BrokerService interface (this is convention). This is necessary because we use the interface when creating the service connector (see task 2). This expects the service namespace to mirror the interface package name (i.e. interface package = “service.core”, targetNamespace = “http://core.service/”)

Similarly, the @SOAPBinding annotation needs to be different because the parameter passed is not a primitive type. To handle this, we need to use the DOCUMENT type instead of the RPC type:

```
@SOAPBinding(style = Style.DOCUMENT, use=Use.LITERAL)
```

c) Finally, you should deploy the Broker Service by implementing a main method that creates an Endpoint. Specifically, you should create a copy of the existing Main class called Server. You should remove the code registering the Broker Service with the registry. In the main, you should create the WS endpoint.

*NOTE: You will likely get an error when running this service, because Jax-B (a Java API that maps Java objects to XML and vice-versa). You will need to change the signature of the BrokerService interface to replace the List<Quotation> return value with an array of type Quotation[].*

*The following line of code will convert your list of quotations to an array of quotations:*

```
return quotations.toArray(new Quotation[quotations.size()]);
```

*This should be used in the LocalBrokerService class where the getQuotations(…) method is implemented.*

You should test that the service is working by viewing the generated WSDL document. Notice how the annotations have changed the WSDL document (in fact, try temporarily removing some of the @WebService parameters to see the effect on the WSDL document).

**Task 2: Implementing the Client** **Grade: C**

Due to the increased complexity of the deployment, creating the client code has been separated from the creation of the server code. To create the client application, start by making a copy of the Main class called Client. Remove all the service registration code, and replace the lookup code with the 3 lines of code that create a service connector to a web service. Remember that the service takes an object as a parameter and returns an object as a result (basically replace the 1 line ServiceRegistry lookup with the 3 lines of code mentioned above.

When getting the port reference "service.getPort(…)" you have to use a slightly different method because the interface and implementation of the Broker Service interface are in different packages:

```
BrokerService brokerService = service.getPort(
      new QName("http://core.service/", "BrokerServicePort"),
      BrokerService.class
);
```

The QName(…) parameters are based on the portName and targetNamespace parameters given in part (b).

**Task 3: Splitting the Broker & Quotation Services** **Grade: B**

The third problem invokes doing the same thing as tasks 1 & 2, but for the 3 quotation services. Each service should be recast as a SOAP-based Web Service. You should delete the ServiceRegistry class and the static initialisation block in the Server class. You also need to modify the code in the Broker Service because it uses the ServiceRegistry.

For this task, you should hard code the 3 WSDL URLs (you can store each one as a constant, and store the set of service URLs as a constant array – e.g. public static final String[] SERVICE_URLS = { URL1, URL2, URL3 })

*NOTE: You can use the same @WebService(…) annotation for all three quotation service implementations. This will allow you to make assumptions about the QName(…) objects you need to use in the broker (i.e. make the service name, target namespace, and port name the same for all three services).*

**Task 4: Service Registration and Discovery** **Grade: A**

The final task / challenge is to integrate the UDDI Registry into the implementation. For this, you should use jUDDI as the registry. Your solution should be based on the example code I have provided on moodle. Based on the philosophy of UDDI, each service should be deployed in a separate organisation (i.e. you need to create 4 businesses, and one service instance per business).

To gain top marks, you should develop a strategy for identifying quotation services / insurance businesses. Each service should be deployed and registered in isolation, and ideally, no duplicate businesses / services will exist. This will require that you learn a bit more about how jUDDI works and the options available via the inquiry service.

Write a SHORT text file that explains how to deploy your solution (call it DEPLOY.txt and put it in the root folder of the project).

NOTE: All usernames / passwords used in the example code are available "out of the box" with jUDDI.

**Additional Marks**

+ grades (e.g. A+) can be attained through consideration of boundary cases, good exception handling, nice features that enhance the quality of your solution.

- grades (e.g. A-) can be attained though lack of commenting and indentation, bad naming conventions or sloppy code.