COMP30220 Distributed Systems Practical

Lab 3: Web Services-based Distribution

Work individually. Submit your code on csmoodle.ucd.ie by the deadline given on moodle.

Scenario:

After seeing all of the problems with the RMI-based implementation of the Car Insurance Quotation System, your enterprising manager has decided that moving forwards, the company should promote web services over "old" proprietary technology like RMI. As a result, you are now tasked with the creation of a web services version of the system. Each quotation service should be implemented as a separate web service. The broker service should then use a web services client to connect to each of these services, requesting quotations.

Given your previous work on the vetting service, you are also required to implement this service in the new system. To help you with this, you have found a version 2.0 of the local service implementation that includes a vetting service. This revised source code is provided on moodle. Please be aware that the changed package structure has been done on purpose to make the implementation of the web services easier. This is because the package structure is used to define XML namespaces in web services and the Jax-WS implementation does not like working with services whose interface and implementation are in separate packages.

Again, use the sample client info that has been provided in the `test.Main` class.

Key Tasks to be completed:

The first task is to implement the quotation services as web services.

a) Modify the Quotation and Vetting Service interfaces to declare them as web services (you can remove the core.Service interface from this implementation as it was only used to allow the Local ServiceRegistry class to work for multiple service types). **10%**

b) Modify the implementations of each service to be declared as web services.
One of the main problems here is that Jax-WS does not like interfaces as either parameters or return values, and in our local implementation, we have used interfaces in both places (ClientInfo for the parameter and Quotation for the return value). To modify each service to be a web service, we must transform these interfaces into Data Beans – classes that contain only data.

An example of a data bean class is:

```
public class Person {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public Person() {}


    public void setName(String name) {
        this.name = name;
    }

    public void setAge(int age) {
        this.age = age;
    }
```

```
    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }
}
```

NOTE: This class must implement a set/get method for each private field in the class. It must also implement a default constructor (a constructor with no arguments). The constructor with arguments is used for initialising a newly created object only.

You must transform the ClientInfo and Quotation interfaces into data beans. This means that the inner class implementations of the Quotation interface will need to be removed and all references to these inner classes will have to be replaced with references to the Quotation class (no longer an interface).

40%

c) Write a class called Deploy.java that deploys each of the four web services on ports 9000-9003 (one port per service). Check that you can see the WSDL definitions for each one. **10%**

d) Create a class called impl.WSBrokerService that implements the web services equivalent of the impl.LocalBrokerService code – namely that it contacts the vetting service (on the predefined URL) to vet the client; and if the client info is validated as correct, then the client contacts the three quotation services (on predefined URLs) to get quotations. **30%**

10% of the final mark will be for commenting / formatting / doing something beyond the spec.

NOTE: Use of UDDI

The solution you have developed is web service based, but the client is hard-coded in that the URIs of the services it is to use are hard coded into it. In some respects, this means your solution is not as flexible as the RMI based solution…

Overcoming this involves the use of UDDI – this is the web services equivalent of the RMI Registry. While you are not going to be examined on this, I would recommend that you try to deploy a UDDI server and modify your code to use the server instead of being hard coded. I recommend you do this by downloading and deploying jUDDI (pronounced Judy), a Java-based UDDI implementation.

jUDDI is available from here: https://juddi/apache.org/

Chapter 2 of the Server User Guide explains how to start up the service:
https://juddi.apache.org/docs/3.3/juddi-guide/html/ch02.html

You will need to (a) find a way to register the four web services with jUDDI, and (b) modify your client code to use jUDDI to locate the four web services**.**