A picture containing graphical user interface

Description automatically generated

Winyul Yin - Version 3.0

**Distributed Ticket Reservation System (DTRS) using Web Services**

Assignment #3

Soen 423 Section H

Professor – Rajagopalan Jayakumar

**Techniques:**

In this assignment, we replaced our CORBA implementation with Webservices,

In order to do that, we need to annotate the interface with @webservices and @SOAPBinding(style=style.RPC).

Same thing with the implementation of the interface however, @webservices we need to add an endpoint reference

With the server class, we replace all our CORBA,POA,SERVANT… bindings with an end point which includes our implementation and prints out (endpoint.isPublished()) to validate that the server is up and running

At that point we can look and validate the WSDL link of our server to validate all our server interface details

For the client,we create a service with the link to the server’s WSDL and a Qname of the server with the endpoint used in the server and a name for the service. Finally, we use the interface given the port of the service to run methods remotely

**Diagram

Description automatically generated**

**Architecture:**

3 Servers: Montreal(MTL)

Toronto(TOR)

Vancouver(VAN)

2 types of clients:

Participants and Admin

Participants methods:

-book event:

-get booking:

-cancel booking:

-Exchange event:

-logout(): switch to another user id

Admin methods

-add event:

-remove event:

-list event availability:

-participant methods:

user recognised from clientID which is 8 characters: serverID(3 char)+ client type(1 char)+id(4 digits)

event determined by eventide which is 10 character : serverID(3 char)+ time(‘M’|’A’|’E’) + event date(ddmmyy)

user must connect to their respective server by CORBA

user and server must have a log of their respective outputs.

**Data Structure:**

Classes used:

-Clients

-ClientObject

-DTRSWebService(interface)

-DTRSWebServiceImpl(implementation of the interface)

-Event

-Logger

-Server Starter

-Server

Concurrent HashMap

Events:<eventType,<eventID,Event>>

ClientEvents:<ParticipantID,<eventType,EventID>>

ServerClient<ClientID,ClientObject>

**Proper and sufficient test scenarios:**

Check for atomicity of the exchange ticket operation as well as concurrency. This can probably be done by having 2 or more clients try to exchange their ticket for an event that has 1 capacity left. Multiple times to guarantee its accurate. Afterwards verifying that only 1 of the 2 got one of each ticket and that if they got their ticket the old event is now available. At the same time, we can have a client try to get one of those old event to see if they didn’t release it before actually getting an new event.This would test for atomicity of the operations as we will test for a client if he only releases the old event when he is able to get a new event that is available. Otherwise return an error.

Other than that, the requirements are pretty much the same as in version 1. Therefore I would reuse the same test scenarios in order to see if the conversion from Java RMI to CORBA has been successful.

Operations to be tested would be test admin and participant account for differing available operations. Verify if we have a concurrent server by adding some operation over the same data/events then verifying that all 5 functions for participants and 8 for admin are running properly.

Finally, inspect the log files to see if it matches with what is printed onto the console.

**Most important part of the assignment:**

The most important part of this assignment is making sure that everything works and is concurrent and highly available since this part is going to be used in the project. Therefore, all test case must pass.

The use of OpenJDK 8 was also important since like CORBA, the javaX package was removed from java version past 11 but since my processor is relatively new and does not support older version of java natively, Only javaSE 17 and up.

**Most difficult part of the assignment:**

The most difficult part for this assignment at the moment is knowing what more to change to the existing code. This is because. The submission for this assignment is schedule at the same day as our demo for the previous assignment therefore, if any changes would have to be made, it would be very rushed in order to implement all the required changes pointed out in demo.

Another difficult part was getting the wsdl files. As loading it from a browser was not possible as I would get a 403 access forbidden response. The reason is because Apple reserved the port 8080 for other use therefore I could not have used it for this application. As a result, it was difficult to figure out the issue.

References

<https://www.concordia.ca/> (concordia university logo)

<https://docs.oracle.com/en/java/javase/16/migrate/removed-tools-and-components.html#GUID-460800EF-A523-4B10-B694-E3536AC419C1> (CORBA removed from java 11 and newer)

<https://docs.azul.com/core/zulu-openjdk/install/macos> (openjdk for macOS m1 )

[**https://moodle.concordia.ca/moodle/pluginfile.php/5667336/mod\_folder/content/0/TUT/TUT\_4\_WebServices.pdf?forcedownload=1**](https://moodle.concordia.ca/moodle/pluginfile.php/5667336/mod_folder/content/0/TUT/TUT_4_WebServices.pdf?forcedownload=1) **(WebService diagram)**