CONCORDIA UNIVERSITY

DEPARTMENT OF
COMPUTER SCIENCE AND SOFTWARE ENGINEERING

COMP 6231 - SUMMER 2019 Instructor and Coordinator: Sukhjinder K. Narula, R. Jayakumar

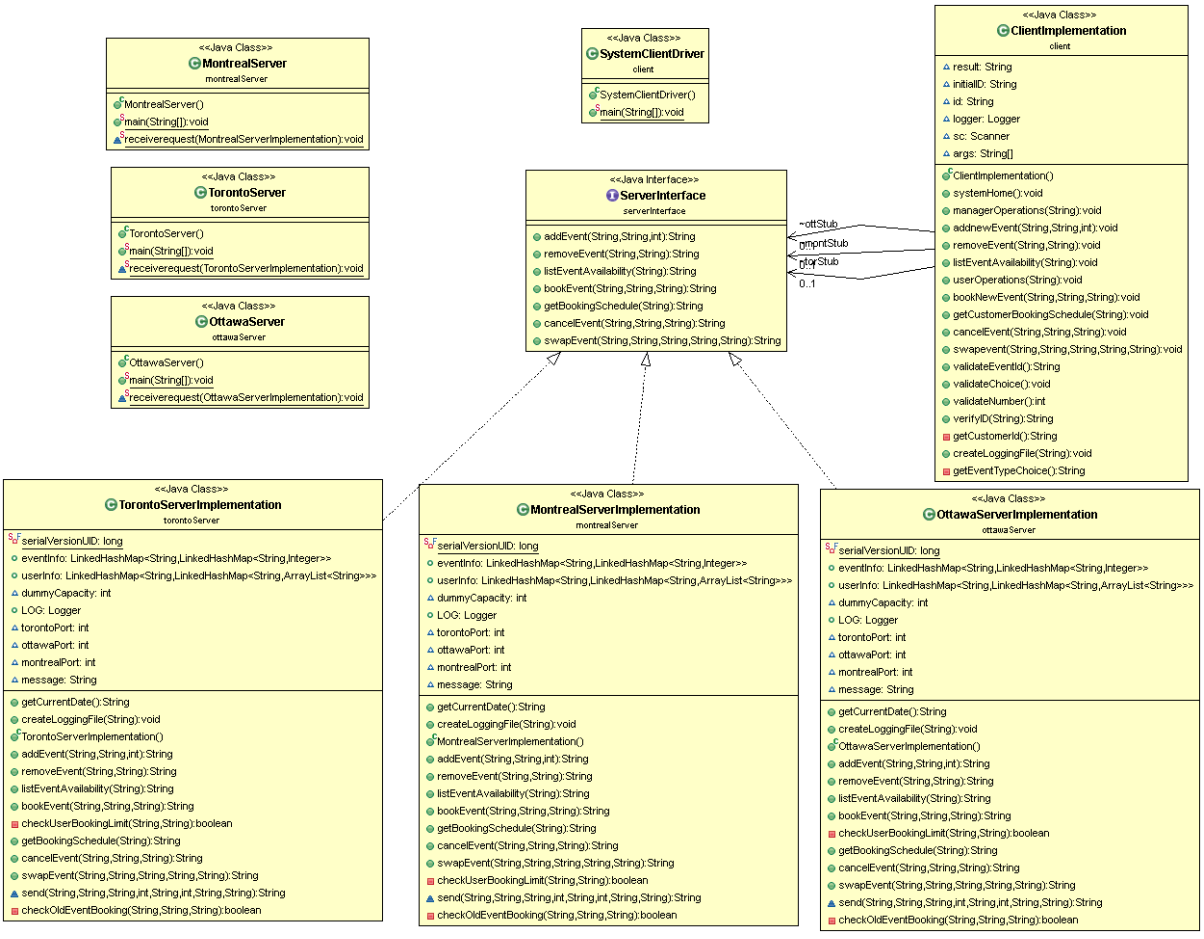
ASSIGNMENT 3

Student Id – 40093384 Name- Yogesh Nimbhorkar

Requirement: In this Assignment, we have been asked to implement a Distributed Event Management System with the help of the Web Services which is the technology of the future in industries to implement distributed systems.

ARCHITECTURE

Class Diagram



We have three servers: Montreal server, Toronto server and Ottawa server. I have designed an interface which consists of all the methods of customer and manager. This interface is then implemented by the implementation class. Client use this information to know the

method structure and definition. In turn, Server has the implementation that implements the methods of this interface. Below is the detailed description –

CLIENT

Client is responsible to remotely invoke the methods of the objects published as a service. It does so using http request and responses. It starts with the client discovering the published service. Client then binds to the server by establishing the TCP connection to the discovered address.
Client builds up the SOAP XML request by passing the arguments with the required service and sends it to the server side.

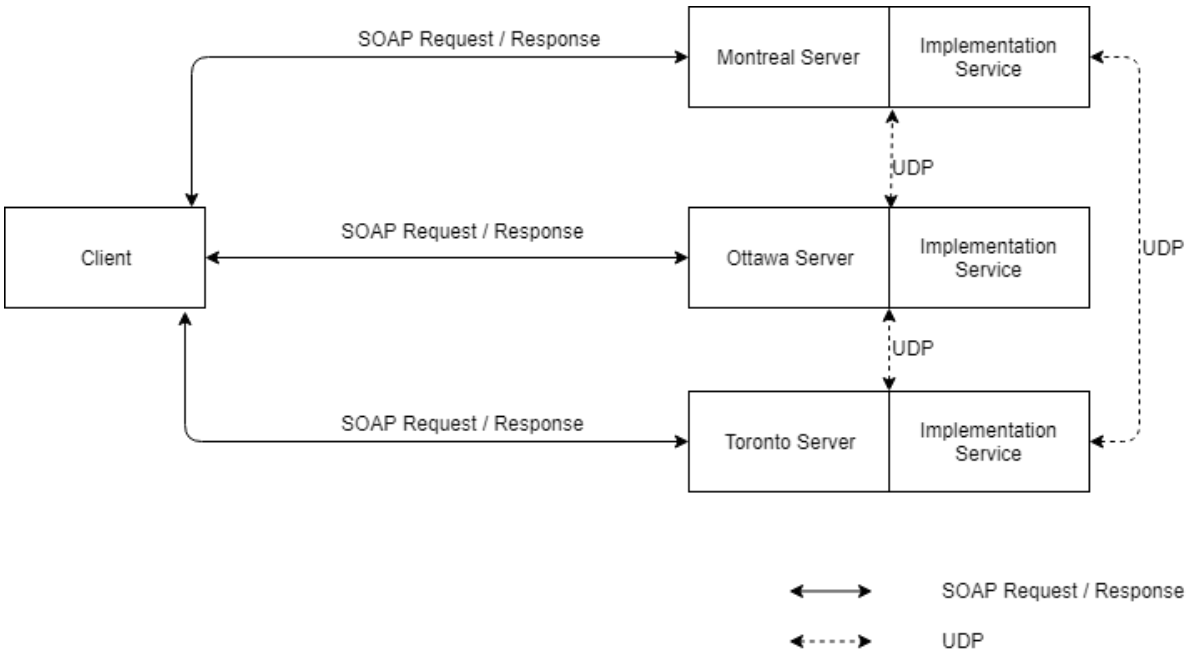
SERVER

In this case, I have designed six server classes for all the three servers respectively.

- MontrealServer.java
- MontrealServerImplementation.java
- OttawaServer.java
- OttawaServerImplementation.java
- TorontoServer.java
- TorontoServerImplementation.java

Server is primarily responsible for implementing the interface and then publishing the implementation objects as a service. Client then binds to this service and then forwards the request using SOAP request. SOAP router routes the request to the appropriate server. Server then unmarshals the request and computes the response which is then sent back to the client using SOAP router.

Below is the architectural diagram of my application –



Distributed Event Management System

This diagram illustrates simple client server architecture using Web Services design.

There are seven methods of user and manager that have been implemented. The significance of each method is mentioned below –

Manager Role:-

Add Event –
When an event manager invokes this method through the server associated with this event manager (determined by the unique eventManagerID prefix), attempts to add an event with the information passed, and inserts the record at the appropriate location in the hash map. The server returns information to the event manager whether the operation was successful or not and both the server and the client store this information in their logs. If an event already exists for same event type, the event manager can't add it again for the same event type but the new bookingCapacity is updated. If an event does not exist in the database for that event type, then add it.

Remove Event –
When invoked by an event manager, the server associated with that event manager (determined by the unique eventManagerID) searches in the hash map to find and delete the event for the indicated eventType and eventID. Upon success or failure it returns a message to the event manager and the logs are updated with this information. If an event does not exist, then obviously there is no deletion performed. Just in case that, if an event exists and a client has booked that event, then, delete the event and take the necessary actions.

List Event Availability –
When an event manager invokes this method from his/her branch's city through the associated server, that branch's city server concurrently finds out the number of spaces available for each event in all the servers, for only the given eventType. This requires inter server communication that will be done using UDP/IP sockets and result will be returned to the client. Eg: Seminars - MTLE130519 3, OTWA060519 6, TORM180519 0, MTLE190519 2.

User Role:-

1. **Book Event** –
   When a customer invokes this method from his/her city through the server associated with this customer (determined by the unique customerID prefix) attempts to book the event for the customer and change the capacity left in that event.
2. **Get Schedule** –
   When a customer invokes this method from his/her city's branch through the server associated with this customer, that city's branch server gets all the events booked by the customer and display them on the console.
3. **Cancel Event** –
   When a customer invokes this method from his/her city's branch through the server associated with this customer (determined by the unique customerID prefix) searches the hash map to find the eventID and remove the event.

4. **Swap Event** –
   A customer might invoke this operation to change an event(belonging to an event type) he/ she has already booked. In this case, the current_city_branch_server (which receives the request from the customer) first checks whether the customer has booked the old event, then checks with the new_city_branch_server (on which the new event has to be booked) whether there is available capacity for the new event, and if both checks are successful then atomically book the customer for the new event and cancel the old event for the customer.

Data Structures: -

1. **Evnet Information HashMap** – This is a nested hashmap which stores the information of an event in the system. It contains key as Event Type and its value as a hashmap which in turn stores Event Id as a key and capacity as the value.

```
public HashMap<String, HashMap<String, Integer>> eventInfo = new HashMap<String, HashMap<String, Integer>>>();
```

2. **User Information HashMap** – This map contains user ID as the key and sub hashmap of Event Type and array list of event ids as values.

```
public HashMap<String, HashMap<String, ArrayList<String>> userInfo = new public HashMap<String, HashMap<String,, ArrayList<String>>>();
```

TEST SCENARIOS:

Created a test class with 7 threads all trying to perform operations concurrently, three threads are trying to exchange an event and three threads are trying to book event. The last one is trying to print user schedule. All operations are happening concurrently, so they have to be synchronized. The expected output can be random.

Difficult Points from this Assignment: -

1. I find it difficult to use 'wsgen' and 'wsimport' command to generate the artifacts.
2. Understanding webservices to correctly implement these in this assignment.
3. Testing all the functionalities again to correctly adapt with webservices.

References: -

- https://www.geeksforgeeks.org/multithreading-in-java
- https://docs.oracle.com/javase/7/docs/technotes
- https://en.wikipedia.org/wiki