DISTRIBUTED SYSTEM DESIGN

Distributed Event Management System

Submitted to: Prof. Sukhjinder K. Narula

Submitted by:

First Name	Last Name	Student ID	Email ID
Krishnan	Krishnamoorthy	40089054	krish27794@gmail.com
Karthik	B P	40094485	karthikbeepi@gmail.com

Objective:

The main purpose of this assignment is to implement the Distributed Event Management System (DEMS) using Java RMI. It is a distributed system used by event managers and customers to manage information regarding the event across different centers.

Introduction:

Distributed event management System is a Distributed system used by Event Managers and Customers. Mangers to manage the information on creation of events and customer to book and cancel these events.
As it is a distributed system, there are more than one event server used in this project. We have implemented 3 event systems server Toronto (TOR), Ottawa (OTW) and Montreal (MTL). Each system has a Client interface which is used by the managers and customers to make use of the event scheduling.
Each client will request for certain details and it would be answered by its corresponding server making it a client-server architecture. However, this being a distributed system, a customer can also access other event servers though not directly.
The customer can interact with another event server through his own server, meaning, a client can only interact with its server. It is the server that can interact with the other server using UDP.

Functionality:

This distributed system has mainly two clients, the manager and the customer. Each individual operation is specified below:

- **Event Manager**

  The functionalities done by manager include:

  1. *addEvent (eventID, eventType, bookingCapacity):*
     Event Manager can add a new event only on its own server

  2. *removeEvent (eventID, eventType):*
     Event Manager can remove a event only on its own server

  3. *listEventAvailability (eventType):*
     Event Manger can look the availability of all the events present in all the servers

- **Customer**

  The Functionalities of the user include

  4. *bookEvent* (*customerID, eventID, eventType*):
     Customer can book a event in its own server as well on other server with max

count of 3

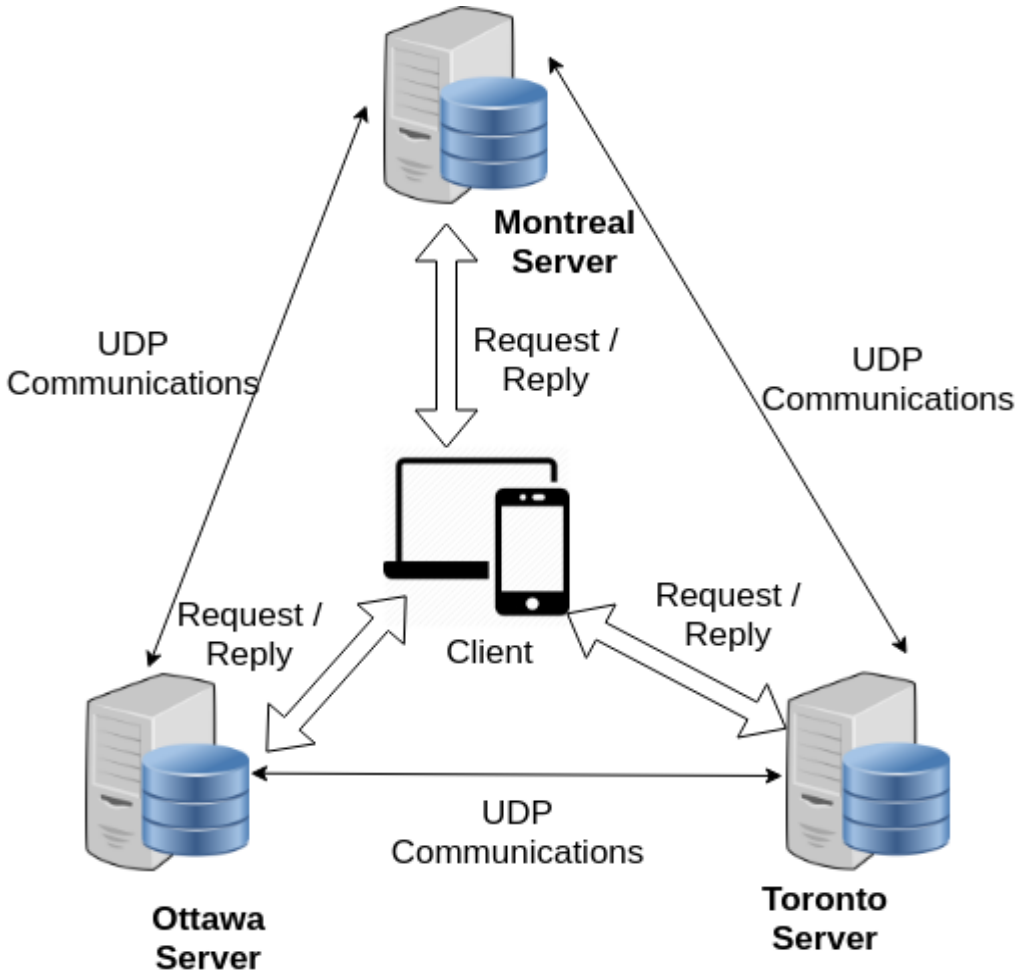
5. *getBookingSchedule* (*customerID*):
Get all the booking schedule from all the server for the customer

6. *cancelEvent (customerID, eventide,eventType):*
Cusromer can cancel the event from all the servers which is already booked by him.

Note: Event Manager has all the operations of a customer, but the other way is not possible.

Architecture:

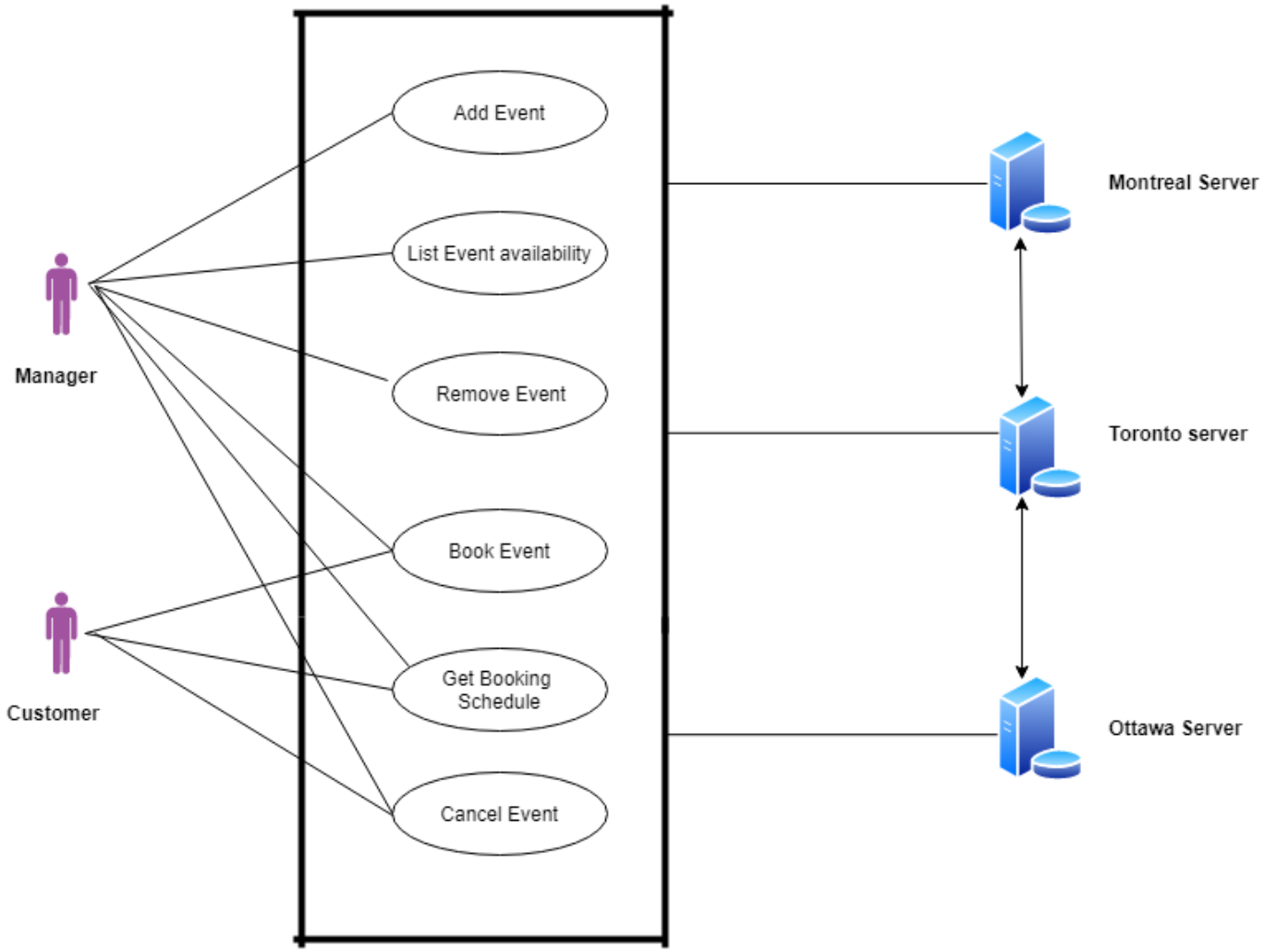
The code for implementing this system is written in JAVA language. The Client Server interaction is achieved using Java RMI Technology and Server – Server interaction using UDP socket connection.
You can find the flow of the system explained in the below diagrams.



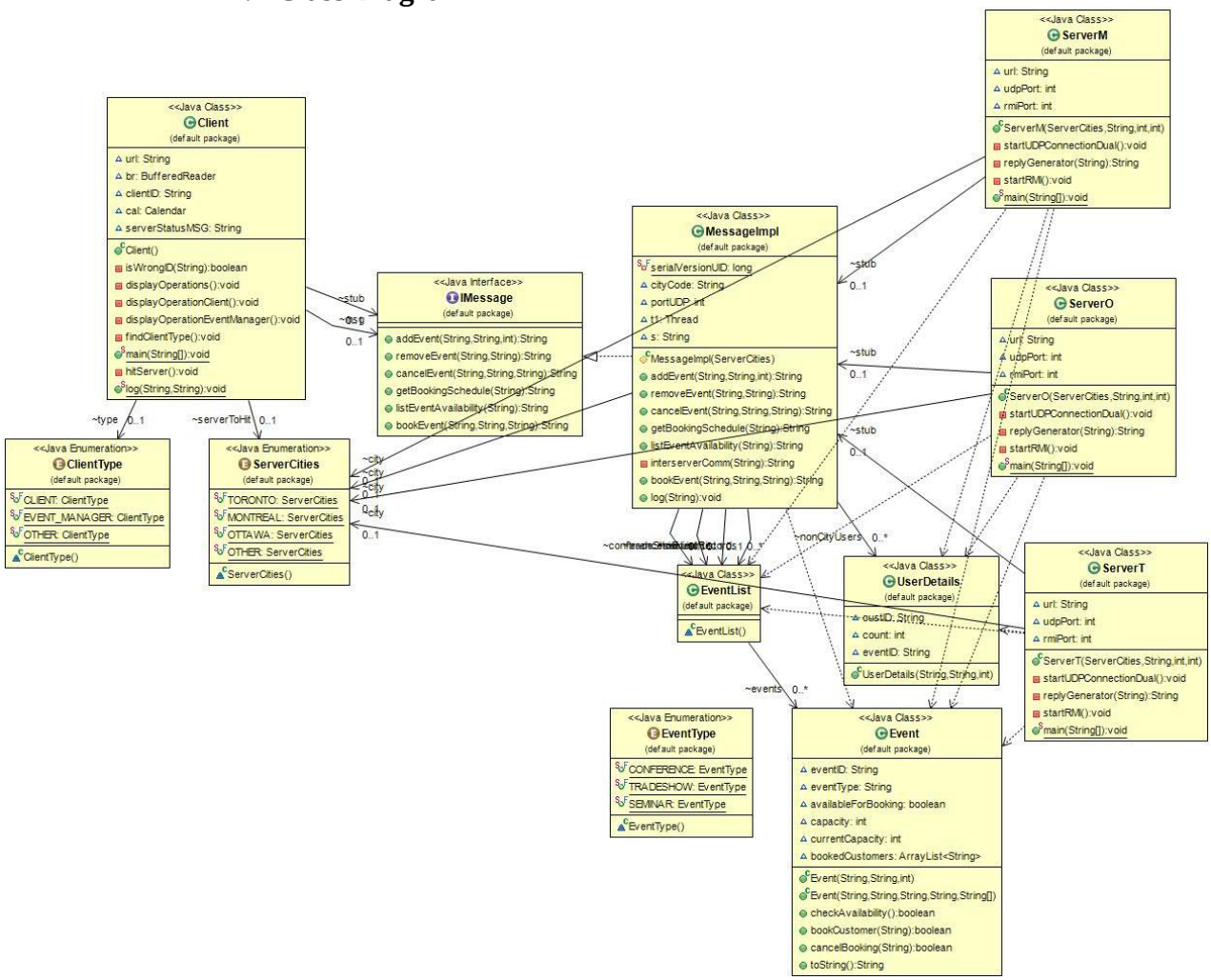
The Client Connects to one server at a time using RMI and each server communicates using UDP sockets with each other.

UML Diagrams

1. Use Case Diagram



2. Class Diagram



Tools and Technology used:

The implementation of this distributed systems involved many technologies, tools and data structures, let's look at each of them individually:

1. Java RMI

Java Remote Method Invocation allows an object running in one machine to invoke methods on an object running in another machine, thus enabling a robust Client-Server interaction mechanism. RMI creates a remote server object that enables client and server communicate through method on the server object. The middleware would enable this communication by using two intermediate object called Stub (on Client Side) and Skeleton (on Server side). The RMIC tool is used to create Stub and Skeleton object, however we had not explicitly created them for this project.

Once, we define all the interfaces and implementations, RMI Registry should be started. Registry

is a remote object that maps names to remote objects. The server registers its remote objects with registry so that they can be looked upon. Whenever client object wants to invoke a method on remote object, it must first lookup the remote object using its name, the registry then returns the reference to remote method.
Each server uses a registry which is created on a specific port. The server first creates registry on a port and binds its objects to it. The client would get the registry and lookup for the server's objects (remote objects) using the same port number. In case of our project, Manager and customer can access the details of their own server using RMI technology.

2. Datagram Socket Interaction

For any two servers to communicate with each other over a network, a socket connection can be used. The communication between them are done by sending packets of data. The Data packets we used here are Datagrams and it's supported by the UDP protocol. The sockets used in this scenario is called Datagram sockets.
A Datagram socket uses a port, which becomes the sending and receiving point for a packet delivery. Each of the Server is run on a different port.

3. Multi-Threading

Multi-threading is a process of executing multiple threads simultaneously and optimizing CPU time. A thread is a part of program that can run concurrently with any other part/thread.
Java RMI provides inbuilt multi-threading functionality, which would make each server interaction a thread. Hence, a server can concurrently service multiple clients.
In our project, the multi-threading is mainly used.

4. RMI and UDP Socket Communication

We run one thread for RMI communication and another thread for receiving and sending inter server communications.

5. Data-Structures

We have primarily used Hash-Map and ArrayList provided by the java.util.
HashMap is a format in which data is stored in key-value pairs. We have used one main HashMap for our project.

EventRecords is the main HashMap which contains event name as key and eventlist as it is a value for another HashMap is the events which contains the event type as key and another important details like availability, capacity and event type.

We have used an ArrayList to store the non-city users to check if only 3 events have been booked for the particular customer. We have used ArrayList to store the value of booked customers as well in the event class.

6. Synchronization

Synchronization is a thread safe phenomenon used in multi-threaded environments. Synchronization limits the access of a shared resource such as Array list or HashMap to one thread at a time. It usually does this by enabling the thread to lock the resource until it is done with it. This would also limit the power of multi-threading as each thread needs to be processed one by one.

7. Logging

All the operations in the system have been logged as part of this project. The logging is done in external file and to access these external file resources. We have used JAVA NIO Package. NonBlocking IO provides a mechanism where thread does not have to wait for the completion of writing into the external file, as in traditional IO, thus optimizing the time utilization.

Test Cases:

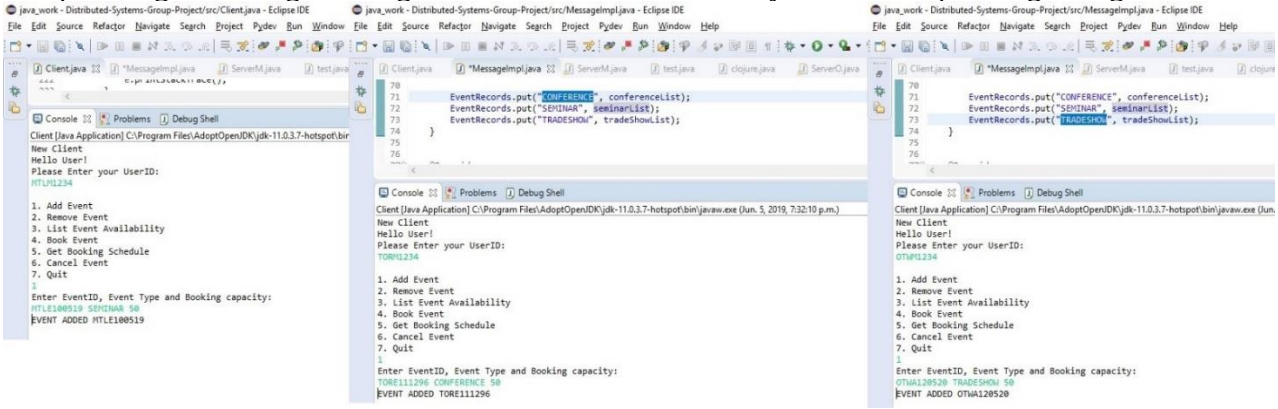
Below are the table list all the test case we performed, we have also attached few test case screen shots for your reference

Test ID	Test Description	Expected Result	Actual Result
T1	The Manager or Customer login must be specific with Letter "M" or " C" and the operations for the particular customer must be displayed.	Specific Operations for Manager or Customer is shown if M or C is specified if anything other than M or C is specified invalid message should be shown	The Corresponding Manager and Customer specific operation was if anything other than M or C is specified invalid message is shown, and a log file is created for each individual if it is valid.
T2	A log file is created for each individual login or appended if the login is already having data in the server	A log file has to be created for new Customer/Manager login and file have to be appended if the login is already present	A log file is created for new Customer/Manager login and the files are appended if the login is already present.
T3	The manager creates an event in his own server and if he tries to create an event in another server	The manager creates an event in his own server and the value should be inserted in the Hashmap and if he tries to create an event in another server it should show a message it is not Possible. If an event is already there the event should be updated with its new capacity	The manager creates an event in his own server and the value got inserted in the Hashmap and if he tries to create an event in another server it showed a message it is not Possible. If an event is already there the event capacity is updated with a new value.
T4	The manager removes an event in his own server	The manager tries to create an event in his own server and	The manager try removes an event in his own server and

	and if he tries to remove an event in another server	the value should be removed in the Hashmap and if he try to remove an event in other server it should show a message it is not Possible and if there is no event of that name the system should say that event don't exist	the value is removed in the Hashmap and if he tries to remove an event in other server it shows a message it is not Possible and if there is no event of that name the system shows that event don't exist
T5	Event Manager tries to list all the event of particular event type.	The System should list all the available event of particular type from all the servers.	The System shows list all the available event of a particular type from all the servers.
T6	Event Manager must be able to do all the client activities as well	The system should be able to allow the manager to do client activities as well, but the other way shouldn't be possible	The system is be able to allow the manager to do client activities as well and the other way wasn't possible
T7	Event Manager tries to book an event for a customer – takes customer id event id and event type	The system should be able to book as many as event for a customer when requested from his/her own city and can book for a customer maximum of 3 events outside his/her own city	The system was able to book as many as event for a customer when requested from his/her own city and can book for a customer maximum of 3 events outside his/her own city
T8	Event Manager tries to cancel an event for a customer- takes customer id and event type and id	The system should be able to cancel the event for a customer when requested if it is present in any server and it show a message as event not present if it is not there	The system is able to cancel the event for a customer when requested if it is present in any server and it show a message as event not present if it is not there
T9	Event Manager tries to get all the booking schedule for a Customer	The System should list all the booked event of given customer from all the servers.	The System shows list all the booked event of given customer from all the servers

T10	Customer tries to book an event- Customer Id is not given as input again	The system should be able to book as many as event for a customer when requested from his/her own city and can book for a customer maximum of 3 events outside his/her own city	The system was able to book as many as event for a customer when requested from his/her own city and can book for a customer maximum of 3 events outside his/her own city
T11	Customer tries to cancel an event- Customer Id is not given as input again	The system should be able to cancel the event for a customer when requested if it is present in any server and it show a message as event not present if it is not there	The system is able to cancel the event for a customer when requested if it is present in any server and it show a message as event not present if it is not there
T12	Customer tries to get all his/her schedule	The System should list all the booked event of given customer from all the servers.	The System shows list all the booked event of given customer from all the servers

Sample image showing creating events in all the 3 servers by their corresponding Mangers



Important/Difficulties Faced:

Client-Server RMI architecture implementation: Understanding RMI and implementing the basic architecture with all the methods was also time-consuming.

UDP Connection establishment: Establishing UDP connection among the 3 servers was time-consuming and challenging.

Deadlock: avoiding deadlocks while accessing the same database for multiple operations simultaneously we used synchronization to avoid this problem**.**

Book Event: Check whether the customer has booked only 3 events booked outside his/her own city

Reference:

https://www.geeksforgeeks.org/remote-method-invocation-in-java/
https://docs.oracle.com/javase/tutorial/rmi/server.html
http://www.objectaid.com/
https://www.javatpoint.com/RMI
http://www.baeldung.com/udp-in-java
https://systembash.com/a-simple-java-udp-server-and-udp-client/
https://www.loggly.com/ultimate-guide/java-logging-basics/
https://www.callicoder.com/java-concurrency-issues-and-thread-synchronization/