**ASSIGNMENT-1**

**DISTRIBUTED ROOM RESERVATION SYSTEM (DRRS)**

**USING JAVA RMI**

**CONCORDIA UNIVERSITY**

DISTRIBUTED SYSTEM DESIGN(COMP 6231)

SUBMITTED BY

MISBAHUDDIN ADIL SYED

STUDENT ID:40058698

**Distributed Room Reservation System (DRRS) using Java RMI**

A Distributed System is a system that comprises of autonomous computers that are networked together using a distribution middleware. They help in sharing distinctive resources and capabilities to ensure users with a single and integrated coherent network.

The Distributed Room Reservation System is a distributed system that allows the Administrators to manage rooms by creating and deleting according to their availability. It allows the Students to manage their bookings by letting them book, cancel and check available rooms.

The system uses JAVA RMI to facilitate the communication between remote objects. It also uses UDP connection to communicate and get available timeslots from peer servers.

1. **JAVA RMI:**

The RMI ( Remote Method Invocation ) is an API that provides a mechanism to create distributed applications In java. The RMI mechanism works by allowing an object to invoke a method on an object running in another JVM.

RMI provides communication between two remote applications using two objects stub and skeleton.

STUB**:**

The Stub is an object that is present on the client side and represents a remote object.It acts like an entryway for all the outgoing requests.

The function of the stub is to marshal and unmarshal the messages that are sent and received on the client or the server side.

Marshalling of messages or parameters is to encode them into a specific format that is apt for transporting on the network.

SKELETON:

The Skeleton is an object that is present on the server side and represents a remote object.It acts like an entryway for all the incoming requests.The function of the stub is to invoke the method on the actual remote object and return the result after marshaling it.

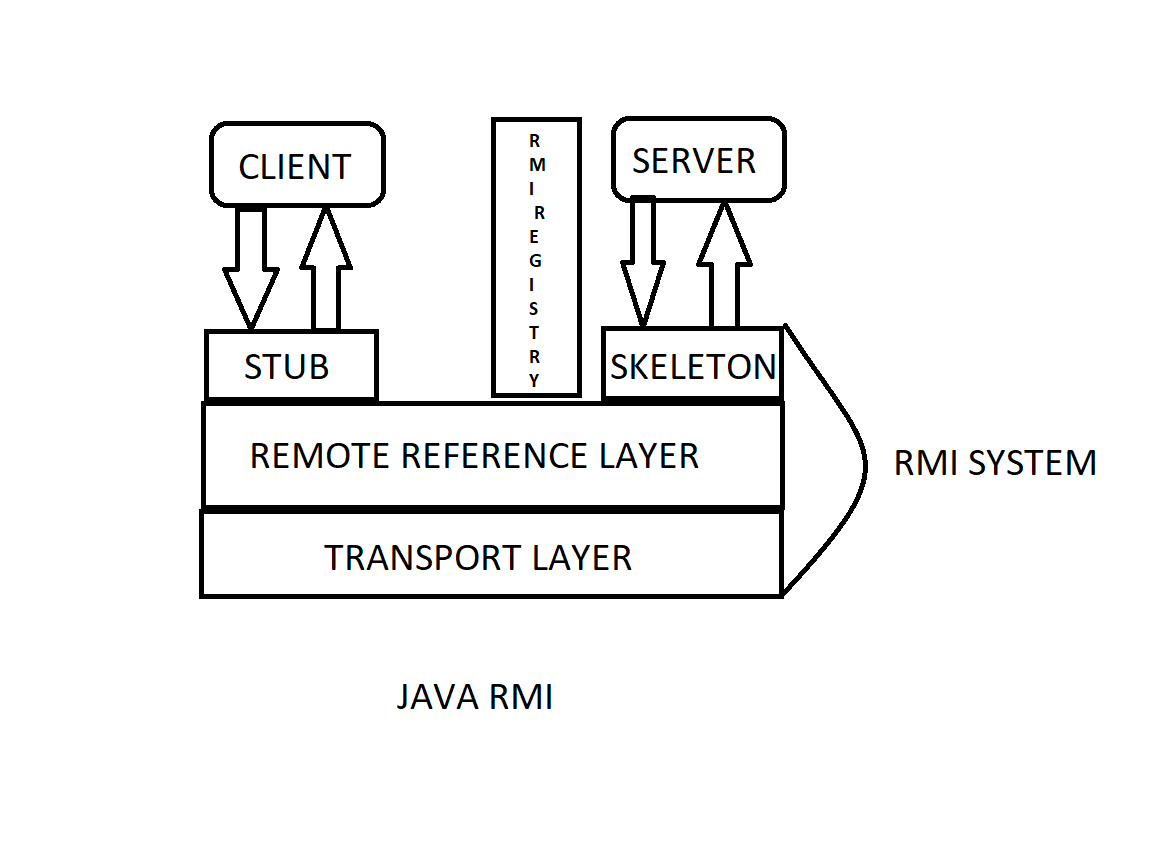
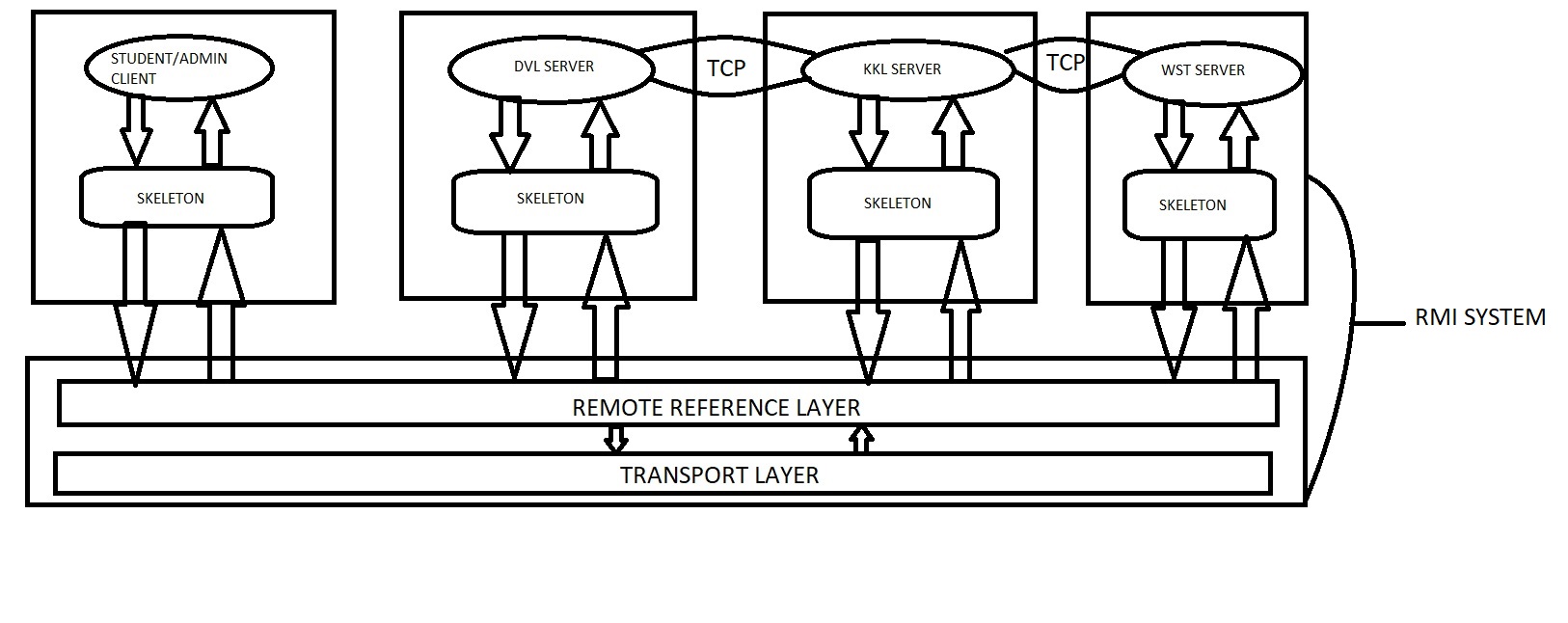


Fig: Working of JAVA RMI

1. **ARCHITECTURE**



The Distributed Room Reservation System is a distributed system which is used by the administrators to show the availability of the rooms and also by the students who reserve the rooms across the universities different campuses.

The Clients i.e. the Students and the Admins have varying accessibility. The Admins can either Create a room or Delete a room whereas the Students can Book a room, get available Time-Slots and Cancel the booking.

There are 3 campuses namely Dorval-Campus ( DVL), Kirkland-Campus (KKL) and Westmount-Campus (WST). Each Campus is assigned a Server. Each server has methods that are accessible by the Students and Admins separately.

The Student and the Admin and represented by respective classes. These classes can call remote methods present on the server through JAVA RMI. Each student is associated with a particular campus which is known by the User Id. To get available Time-Slots present on other servers, the server associated with the student has to communicate with them using UDP Connection.

1. **STRUCTURE**

The Distributed Room Reservation System has the following Structure:

1. Clientpackage

* Homepageclass
* Clientadmin
* Clientstudent

1. Server1package

* Server1interface
* Server1impl
* Server1rmireg
* Threadlistener
* Threadsender1

1. Server2package

* Server2interface
* Server2impl
* Server2rmireg
* Threadlistener
* Threadsender1

1. Server1package

* Server1interface
* Server1impl
* Server1rmireg
* Threadlistener
* Threadsender1

1. **SERVER METHODS**

Each Server contains methods that are separately accessible by Students and Admins respectively. The methods are

1. **Boolean CreateRoom(String rno,String date, String timeslots):**
2. **Boolean DeleteRoom(String rno, String date, String timeslot):**
3. **String Bookroom(String campusName, String rno, String date, String Timeslot):**
4. **String getAvailableTimeSlot (date):**
5. **Boolean CancelBooking (bookingID):**

.

The methods accessible by **Admin** are:

1. **Boolean CreateRoom(String rno, String date, String timeslots):**

The method CreateRoom is a method that is only accessible by an Admin. An Admin, who has the information about the availability of the rooms can create a room record of availability of a room using this method. The admin has to provide Room Number, Date and the Time-Slot of the available room. It returns True if the record creation is successful else returns False.

1. **Boolean DeleteRoom(String rno, String date, String timeslot):**

The method DeleteRoom is another method that is only accessible by an Admin. An Admin can only delete a room only if it was earlier created by an Admin. It returns True if the Room Deletion was successful or else returns False.

The methods accessible by **Students** are:

1. **StringBookroom(String campusName,String rno,String date,String Timeslot):**

The method Bookroom is a method that is accessible by a Student. A Student can book a room by providing the Campus Name, Room Number, Date and the Timeslot. It returns a bookingID on successful booking of room.

1. **String getAvailableTimeSlot (date):**

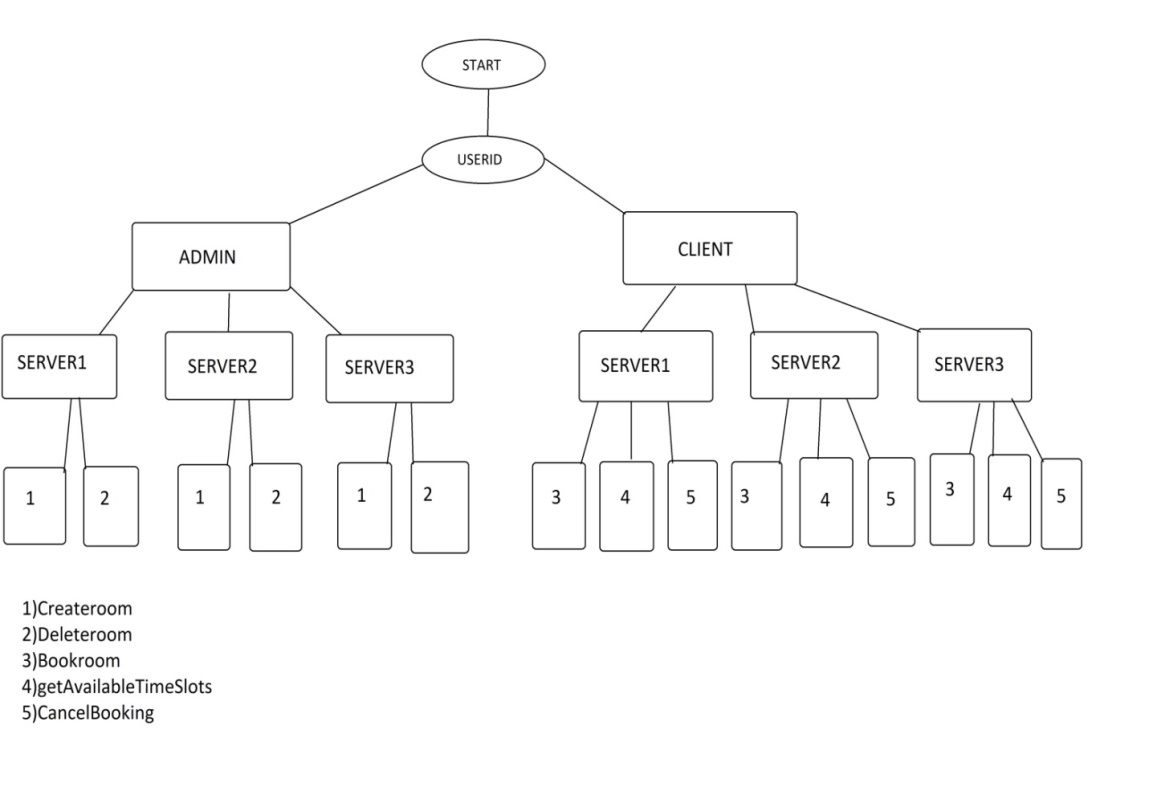
This method checks the HashMap for the available Timeslots. It gets the available Timeslots from other Servers by communicating with them through UDP connection. It receives the count from other servers and returns all the timeslots as a String.

1. **Boolean CancelBooking (bookingID):**

This method checks the Hashmap for the given bookingID and deletes the booking record. It returns True if successful else returns False. It is accessible by a Student.

1. **WORKFLOW**

The Workflow of the System has the following workflow:



The System first asks the user to enter the Userid and based on the 4th letter of the UserID , it redirects the control to the respective class. If the 4th letter is “A”, then the control is redirected to AdminClient and if 4th letter is “S”, then the control is redirected to StudentClient.

If the control is passed to AdminClient, it asks the user to enter a number associated with an action. 1 for createroom and 2 for deleteroom. Based on the Userid, the programs calls the method of the server associated with the Userid. 1 and 2 gives a call to the respective methods of the server using the server object through RMI.

If the control is passed to StudentClient, it asks the user to enter a number associated with an action. 1 for bookroom, 2 for getAvailableTimeSlots and 3 for CancelBooking. Based on the Userid, the programs calls the method of the server associated with the Userid. 1,2 and 3 gives a call to the respective methods of the server using the server object through RMI.

The Createroom method on the sever-side creates an entry in the HashMap for the given Date, Room number and Timeslot and makes the availability field Available and returns True to the Admin.It also increments the roomcount.

The Deleteroom method deletes the specified room on the specific Date and Timeslot from the HashMap and decrements the roomcount.It returns a Boolean value depending upon the operation.

The Bookroom method sets the availability field with the bookingID in the HashMap and returns the bookingID to the User.

The getAvailableTimeSlots method iterates through the HashMap to get the count of the available rooms for the local server. It gets the count of other servers by communicating through UDP connection. It uses threadListner and threadSender class to send the count.

The CancelBooking method searches for the bookingID and the sets the availability field to available.It returns a Boolean value.

1. **DATABASE STRUCTURE**

The database Structure used here is nested HashMap.

The first HashMap ‘a’ has Date as the key and another HashMap ‘b’ as the value.

HashMap<String,HashMap<String,HashMap<String,String>>> a=

**new** HashMap<String,HashMap<String,HashMap<String,String>>>();

The second HashMap ‘b’ has the Room Number as the key and another HashMap ‘c’ as the value.

HashMap<String,HashMap<String,String>> b=**new** HashMap<String,HashMap<String,String>>();

The third HashMap ‘c’ has Timeslot as the key and the availability as the value.

HashMap<String,String> c=**new** HashMap<String,String>();

The following snippet demonstrates addition of a record:

c.put(timeslot,"Available");

b.put(rno,c);

a.put(date,b);

1. **CONCURRENCY**

Concurrency in distributed systems can be attained using Multithreading and Synchronization.

In this system, we need concurrency when many students access the methods such as bookroom(), getAvailableTimeSlots() and Cancelbooking().

The logic of these methods are placed in respective classes and objects of these classes are called using dynamic threads.

The logic code is placed in the critical block so that the modification of the HashMap does not overlap and cause data inconsistency.

1. **TEST CASES**

Test Cases using Admin Client and Student Client and by providing different inputs:

1. UserID :DVLA10005

createroom(String rno, String date, String timeslot)

INPUT:

101

28-10-2017

1:00-3:00

OUTPUT:

Room Created

2)UserID: DVLA10005

deleteroom(String rno, String date, String timeslot)

INPUT:

101

28-10-2017

1:00-3:00

OUTPUT:

Room deleted

3)UserID: KKLA10006

createroom(String roomnumber,String date,String t\_slot)

INPUT:

201

08-10-2017

10-1

OUTPUT:

Room Created

4)UserID:DVLA12345

createroom(String roomnumber,String date,String timeslot)

INPUT:

301

08-10-2017

10-1

OUTPUT:

Room Created

5)UserID:DVLS12345

bookroom(String campusName,String rno,String date,String timeslot,String UID)

INPUT:

DVL

101

08-10-2017

10-1

OUTPUT:

your booking id is 5406c32e-cd82-4096-9364-42e6292a4602

6)UserID: DVLS12345

Cancelbooking(bookingID)

INPUT:

5406c32e-cd82-4096-9364-42e6292a4602

Booking cancelled