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

**Java Rmi:**

The java RMI is a mechanism in java that is used to create Distributed Applications.

The java Remote method invocation gives flexibility to an object to call methods of another object that is running on a different host altogether thus providing mechanism for creating Distributed Applications.

RMI makes use of stub and skeleton for communicating with the Remote object (an object whose methods can be invoked from another object).

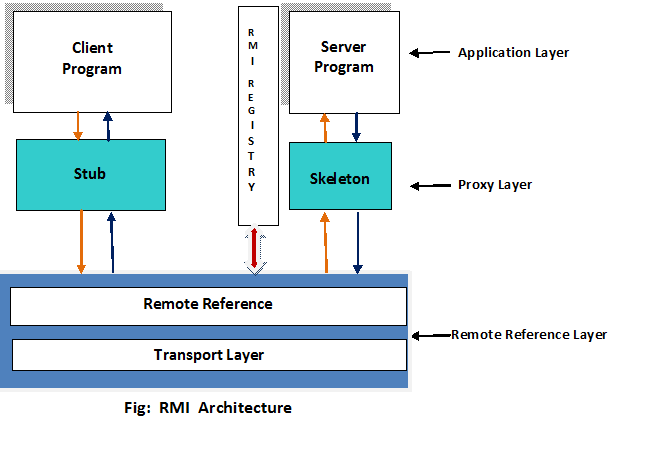
**Stub:**

Stub is an object that resides on the client machine and acts as a remote object(proxy), when the caller calls a method on the stud object it first initiates a connection with the JVM and marshals the data i.e. the parameter passed by the caller and waits for the result, once it receives the result from the remote object it unmarshalls it and passes the result to the caller.

**Skeleton:**

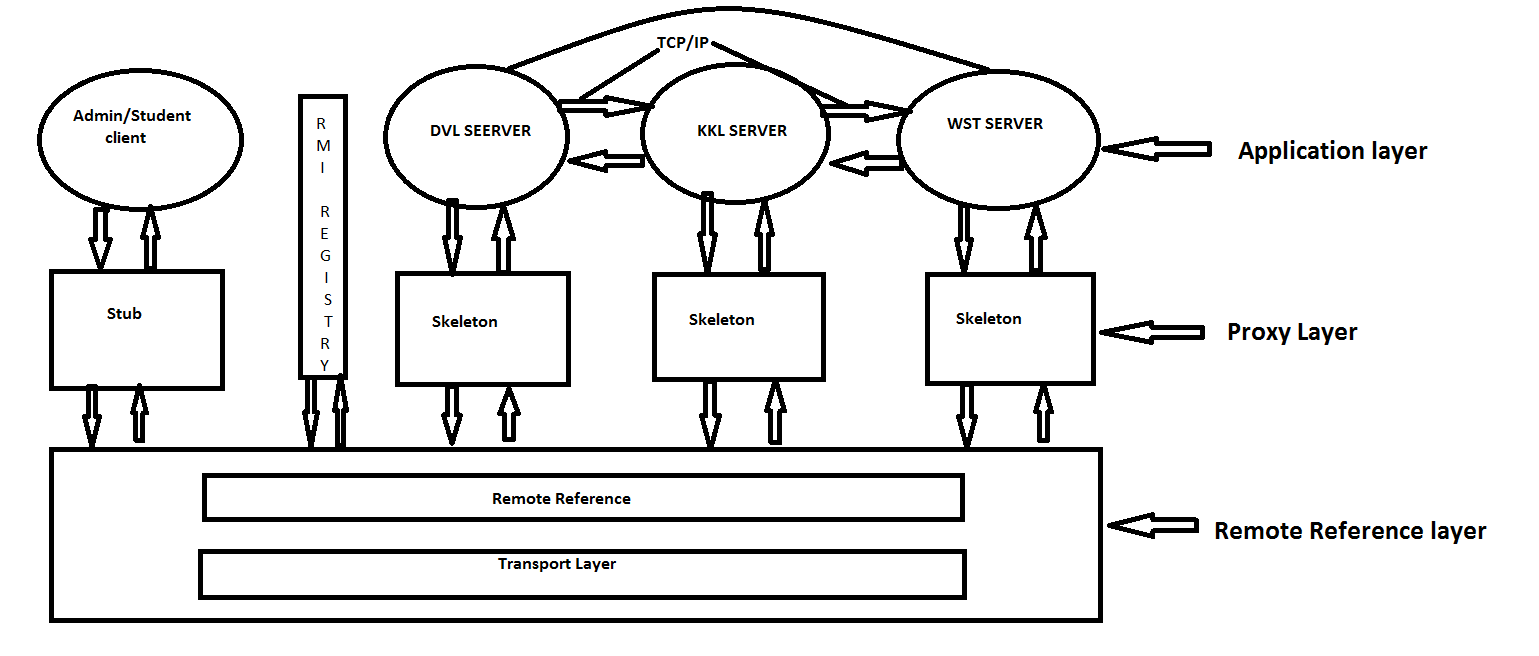
Skeleton is similar to the stub, it acts as a gateway for the remote object. All the incoming requests for the remote object are passed through the skeleton object. It takes the marshalled parameters from the calling object and unmarshalls it so that the remote object can execute its method with the parameters passed from the caller. It then takes the result from the remote object , marshals it ad transmits to the caller.

Below picture describes the architecture of the Java RMI.



Source: Taken from internet.

Architecture of the DRRS system



The Distributed Room Reservation System has 3 server’s namely DVL(Dorval-Campus), KKL(Kirkland Campus) and WST(Westmount-Campus) .The users of the servers are administrators who manage the availability information about the university’s rooms .Every server has a specified administrator who can create and delete the rooms in the server .

The other users of the system are the students who reserve the rooms across the university’s different campuses.

Here the client communicates with the server through Java RMI and the server -server communication is done by UDP/IP.

The System is designed in such a manner that the Student can only perform operations specific to him i.e., booking a room, Cancel booking and getavailabletimeslots.

The admin can perform functions such as Creating a room and Deleting a room.

**Structure of the project:**

The project has the following packages and the corresponding classes.

1)Client\_package

* Mainclass
* Adminclass
* Studentclass

2)package1

* Alwayslisterner1
* Sender1
* Server1code
* Server1interface
* Server1registry

3)package2

* Alwayslisterner2
* Sender2
* Server2code
* Server2interface
* Server2registry

4)package3

* Alwayslisterner3
* Sender3
* Server3code
* Server3interface
* Server3registry

**Server Methods:**

The server Implements the following five functions,

1. **public** **boolean** createroom(String rno,String date,String timeslot)**throws** java.rmi.RemoteException;;
2. **public** **boolean** deleteroom(String rno, String date, String timeslot)**throws** java.rmi.RemoteException;;
3. **public** String bookroom(String campusName,String rno,String date,String timeslot,String UID)**throws** java.rmi.RemoteException;;
4. **public** **int** getAvailableTimeSlot ( String date)**throws** java.rmi.RemoteException, InterruptedException;;
5. **public** **void** cancelBooking(String bookingID)**throws** java.rmi.RemoteException;;

1.createroom(String rno,String date,String timeslot)

The admin invokes this method to create a room record ,the admin should pass the room number ,date and timeslot to create the record. This method returns true if the room creation is successful ,it returns false if the room already exists.

2. deleteroom(String rno, String date, String timeslot)

The admin invokes this method in order to delete a room record, the admin must pass The specified parameters i.e., room number ,date and timeslot that he needs to get deleted .This method returns true if the deletion is successful and reduces the booking count of the student. It returns false if the specified record does not exist.

**3.String bookroom(String campusName,String rno,String date,String timeslot,String UID)**

The student invokes this method to book a room created by the admin ,the student must pass the appropriate parameters that are required to book a room i.e., Name of the campus name,roomnumber,date on which he wants to book a room and the timeslot which he prefers to book. Student can book a room on any campus but initially the requests comes to the server to which student belongs .so if a student wants to book a room outside of his campus then there should be a request to the other servers from the server to which the student belongs. This method returns a unique booking id to the student upon successful booking.

**4. getAvailableTimeSlot ( String date)**

The student invokes this method in order to get the number of available timeslots in all the servers. The request initially comes to the server of the campus to which the student belongs It calculates it available timeslots locally, in order to calculate the available timeslots of the other’s servers we are using udp/ip .it calculates the number of available timeslots in each server and returns the result to the client.

**5. cancelBooking(String bookingID)**

The student invokes this method to cancel the room that he booked previously. The student must provide the unique bookingid that was provided to him at the time of booking room. This method check’s that an occupied time slot can only be cancelled if it was booked by the same student who sends cancel request and has a valid bookingID.It returns true if the cancellation is successful ,returns false otherwise.

**Workflow of the project:**

Initially the application asks for the user id of the client ,depending upon the user id provided the application redirects the users.

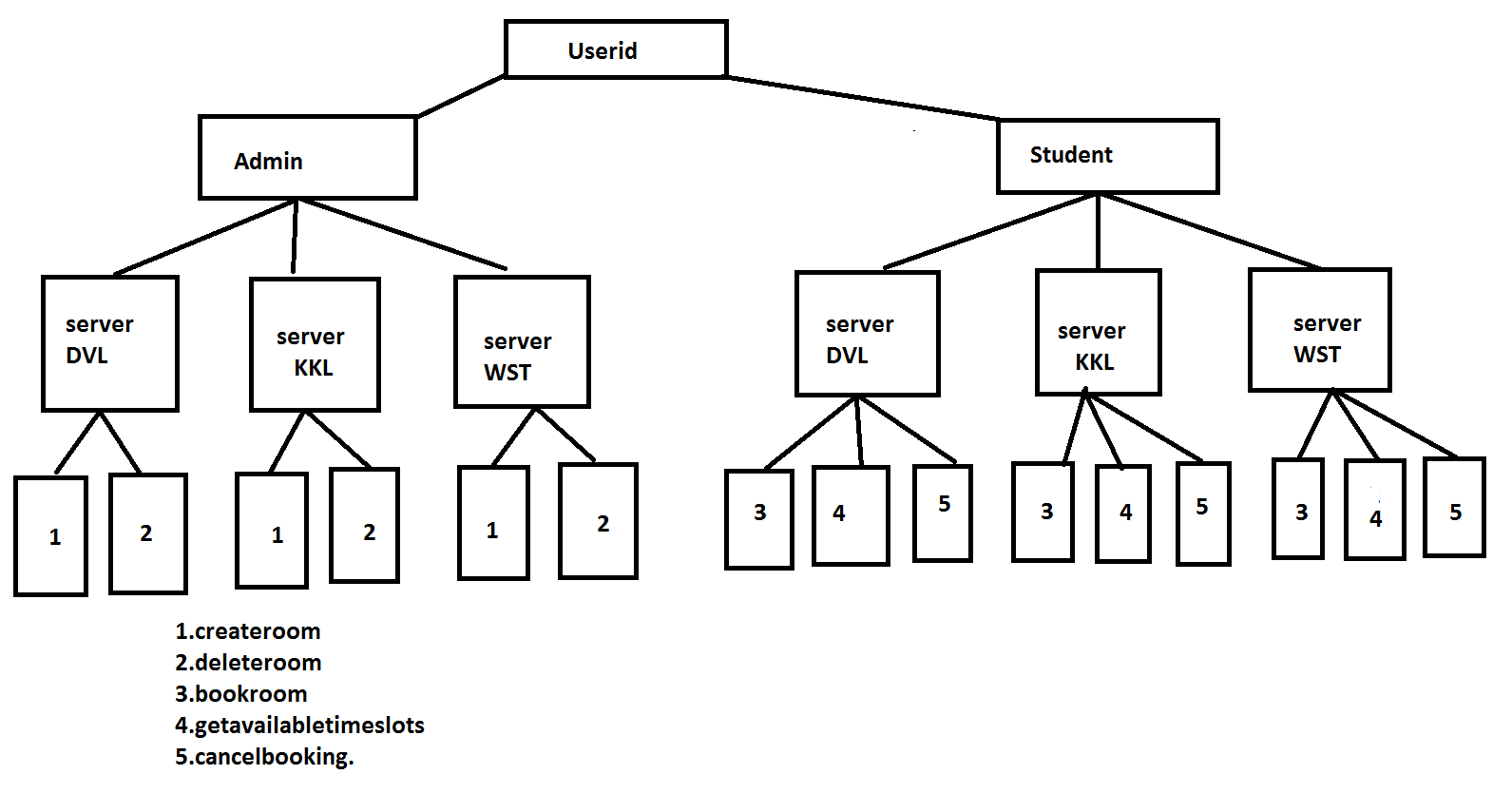
If the fourth character of the user id is A then the application recognizes the user as an admin and redirects it to the admin class.

If the fourth character of the user id is S then the application recognizes the user as an student and redirects it to the student class.

If the user is an admin the application asks the admin to select an appropriate option from the presented options i.e, one to create room record and the other is to delete the room record .Then the application checks the user id to see to which campus the admin belongs to .if the admin belongs to DVL campus then the application calls the createroom or deleteroom of the DVL server based on the admins choice. if the admin belongs to KKL campus then the application calls the createroom or deleteroom of the KKL server based on the admins selection.or if the admin belongs to WST campus then the application calls the createroom or deleteroom of the WST server based on the admins input.

If the user is a student the application asks the student to choose from the options presented to him i.e, to book a room,to get the available timeslots in all the campuses and to cancel the booking .when the student selects an option then the application checks the user id of the student to see to which campus the student belongs to .If the student belongs to DVL campus then the application calls the bookroom,getavailabletimeslots,cancalbooking methods of the DVL server depending upon the student’s choice. If the student belongs to KKL campus then the application calls the bookroom,getavailabletimeslots,cancalbooking methods of the KKL server depending upon the student’s choice.or If the student belongs to WST campus then the application calls the bookroom,getavailabletimeslots,cancalbooking methods of the WST server depending upon the student’s choice.

The following figure depicts the workflow of the application.



**Database Structure:**

The application uses HashMap to create the database ,the following snippet shows the Database of each server :

HashMap<String,HashMap<String,HashMap<String,String>>> hm1=new HashMap<String,HashMap<String,HashMap<String,String>>>();

HashMap<String,HashMap<String,String>> hm2=new HashMap<String,HashMap<String,String>>();

HashMap<String,String> hm3=new HashMap<String,String>();

hm3 is a Hashmap which stores the timeslot and availability as a key-value pair(hm3 is a sub-Hashmap for hm2).

Hm2 is Hashmap which stores the roomnumber and the object of hm3 as a key-value pair(hm2 is a sub-Hashmap for hm1).

Hm1 is a Hashmap which stores the date and the object of hm2 as a key-value pair.

The following snippet shows the way to insert a room record

hm3.put(t\_slot,"Available");

hm2.put(roomnumber,hm3);

hm1.put(date,hm2);

**Concurrency:**

As there only few administrators concurrency in the methods createroom and deleteroom is not required.

As there will be a lot of student accessing the servers, concurrency for the student methods such as bookroom ,getavailabletimeslots and cancelbooking is required.

To achieve concurrency the whole logic of these three methods is placed in three different classes which implements the Thread class , dynamic threads are created and allocated to each client request.

To achieve synchronization in bookroom ,getavailable and cancelbooking methods im putting the critical block of code in the synchronized block .

**Test cases:**

1)DVL admin creates a room.

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

INPUT:

201

08-10-2017

10-1

OUTPUT:

Room Cretated

2)DVL admin deletes the created room

2)deleteroom(String roomnumber, String date, String t\_slot)

.INPUT:

201

08-10-2017

10-1

OUTPUT:

Room deleted

3)KKL admin creates a room.

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

INPUT:

201

08-10-2017

10-1

OUTPUT:

Room Cretated

4)WST admin creates a room.

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

INPUT:

301

08-10-2017

10-1

OUTPUT:

Room Cretated

5)WST Student books a room.

bookroom(String campusName,String roomnumber,String date,String t\_slot,String UID)

INPUT:

WST

301

08-10-2017

10-1

WSTS1234

OUTPUT:

Your booking id is ae2df745-635d-4522-8497-591c2d1c9065

6)WST cancels the booking

INPUT:

ae2df745-635d-4522-8497-591c2d1c9065

**OUTPUT:**

Cancellation is succesful