CS 475

Dr. Sanjeev Setia

Name: Shohinjon Khamidjanov & Avelino Inguane
Login IDs: ainguane & skhamidj

RMI: Creating Distributed Applications

General Steps

1. Designing and implementing the components of distributed application.
2. Compiling sources.
3. Making classes network accessible.
4. Starting the application

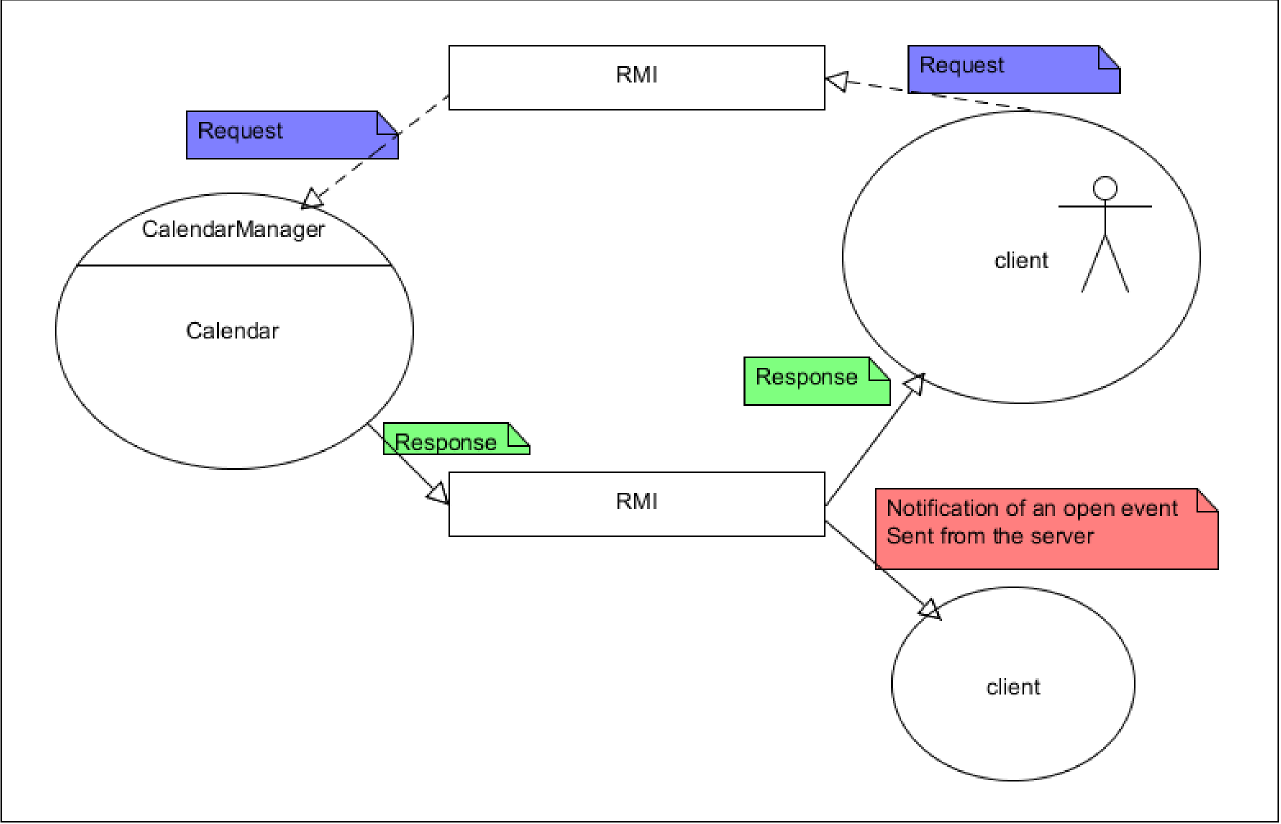
1. Designing and implementing the components of distributed application.

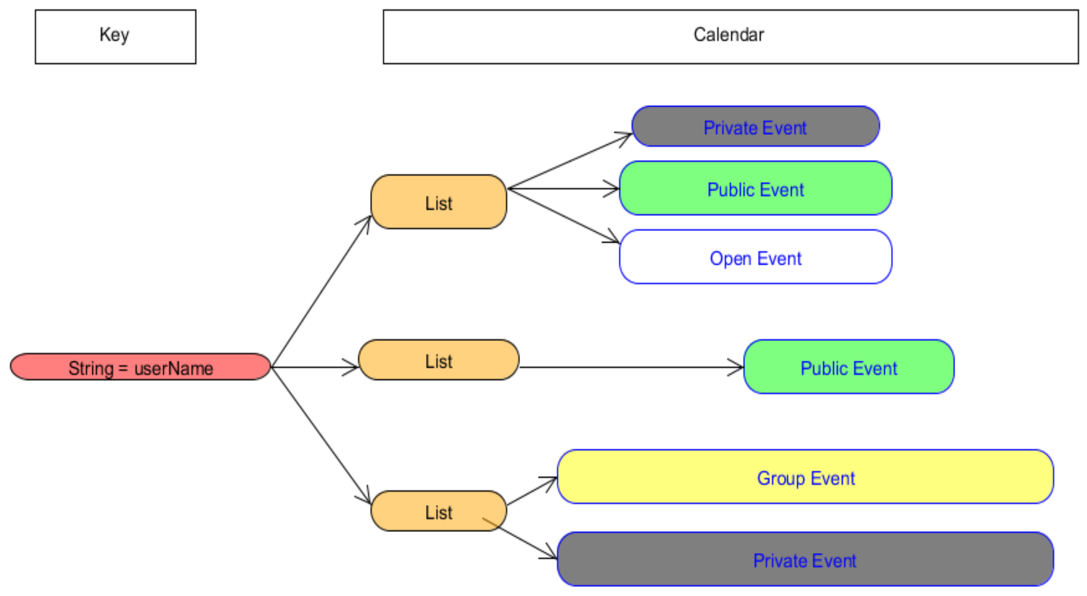
    a. Defining the remote interfaces. A remote interface specifies the methods that can be invoked remotely by a client. Clients program to remote interfaces, not to the implementation classes of those interfaces. The design of such interfaces includes the determination of the types of objects that will be used as the parameters and return values for these methods. If any of these interfaces or classes do not yet exist, you need to define them as well.

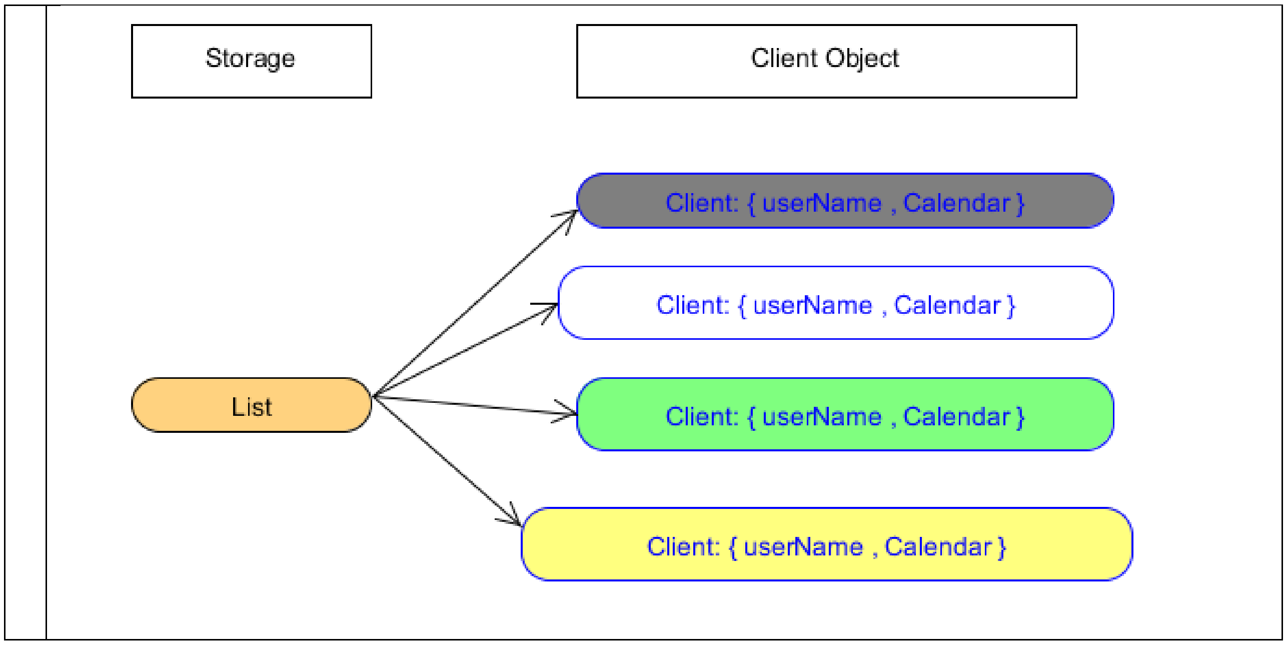
    b. Implementing the remote objects. Remote objects must implement one or more remote interfaces. The remote object class may include implementations of other interfaces and methods that are available only locally. If any local classes are to be used for parameters or return values of any of these methods, they must be implemented as well.

    c. Implementing the clients. Clients that use remote objects can be implemented at any time after the remote interfaces are defined, including after the remote objects have been deployed.

DESIGN


RMI
Request
Request
CalendarManager
Calendar
client
Response
Response
RMI
Notification of an open event
Sent from the server
client

	Calendar Overview
1	`private Map<String, ArrayList<Event>> userCalendar  = new TreeMap();`
2	 Key Calendar Private Event Public Event Open Event List String = userName List Public Event List Group Event Private Event
3	
4	**Notification Overview**
5	`private ArrayList<RemCalendar> chatClients;`


Storage
Client Object
Client: { userName , Calendar }
Client: { userName , Calendar }
List
Client: { userName , Calendar }
Client: { userName , Calendar }

Notification Functionality

Client: An Open Event

```
public boolean addEvent(String userName, String timeInterval, String eventDescription, String accessControl)
        throws RemoteException;
```

Server: broadcast to other clients (Alert other clients of other client availability)

```
public void broadcastMessage(String message)throws RemoteException
```

Other Clients:

```
//retrieve a notification sent by a chat server
public void retrieveMessage(String message) throws RemoteException
```

Compiling sources & Starting the application

1. Start the Server

```
javac CalendarManager.java
rmiregistry &
java -Djava.security.policy=policy.txt CalendarManager &
```

2. Start the Client

```
javac Client.java
java Client Alice
```

Making classes network accessible

SERVER

```
public interface RemCalendar extends Remote
public class Calendar extends UnicastRemoteObject implements RemCalendar
Calendar remote = new Calendar();
Naming.rebind("CalendarServices", remote);
```

CLIENT

```
public class Client extends UnicastRemoteObject implements RemCalendar
RemCalendar remcalendar = (RemCalendar) Naming.lookup(strName);

public void registerChatClient(RemCalendar chatClient) throws RemoteException
```

3. For each client

a. Create three clients for `Alice`, `Bob` and `Tom` respectively

```
protected Client(String chatClientName, RemCalendar chatServer) throws RemoteException
public void registerChatClient(RemCalendar chatClient) throws RemoteException
```

b. Try to create `another` **`client`** `for Alice` – should report error

```
protected Client(String chatClientName, RemCalendar chatServer) throws RemoteException
public boolean findClient(String client)throws RemoteException
```

c. Insert `public` and `private` events into Alice, Bob and Tom's calendars including an event that results in an alarm

```
public boolean createCalendar(String userName) throws RemoteException;
```

```
public boolean addEvent(String userName, String timeInterval, String eventDescription, String accessControl)
        throws RemoteException;
```

d. If necessary, `insert open events` too.

```
public boolean addEvent(String userName, String timeInterval, String eventDescription, String accessControl)
        throws RemoteException;
```

e. Enter `events with mistakes` (test robustness of interface) ????

f. From Alice's client, look at Bob and Tom's calendars.

```
public void viewOtherCalendar(String userName) throws RemoteException;
```

g. Insert `group event into all three` calendars.

```
public boolean addGroupEvent(String userName,String timeInterval,String eventDescription,String accessControl)
        throws RemoteException ;
```

h. Insert group event into Alice and Bob's calendars.

```
public boolean postInOtherCalendar(String userName)  throws RemoteException;
```

i. Check ability to `delete events {group or` individual}

```
public boolean deleteEvent(String userName, String eventTime) throws RemoteException;
```