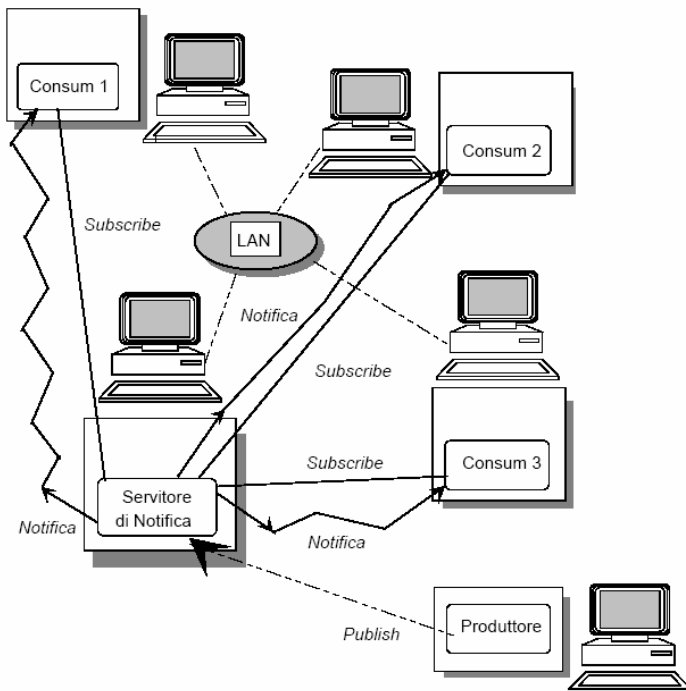


7° Esercitazione (svolta):

Java RMI: callback

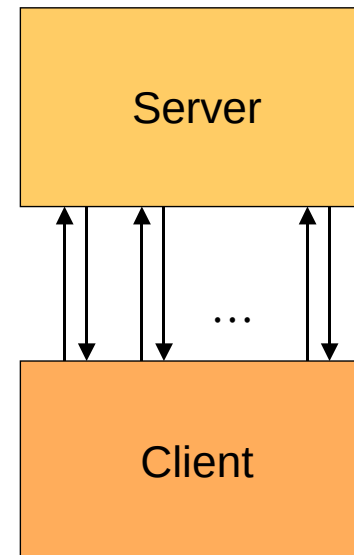
Molte applicazioni
richiedono un
meccanismo
publish/subscribe ...

I partecipanti (client)
necessitano di notifiche da
parte del coordinatore (server)

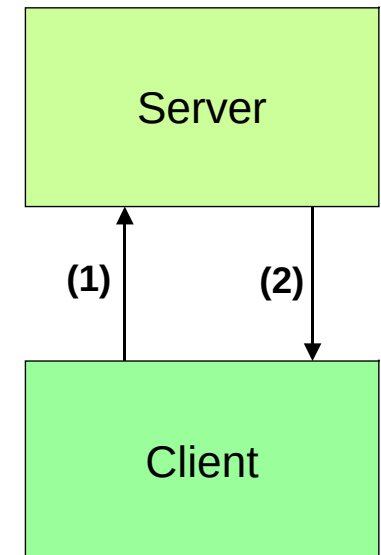


➡ due possibilità:

Polling



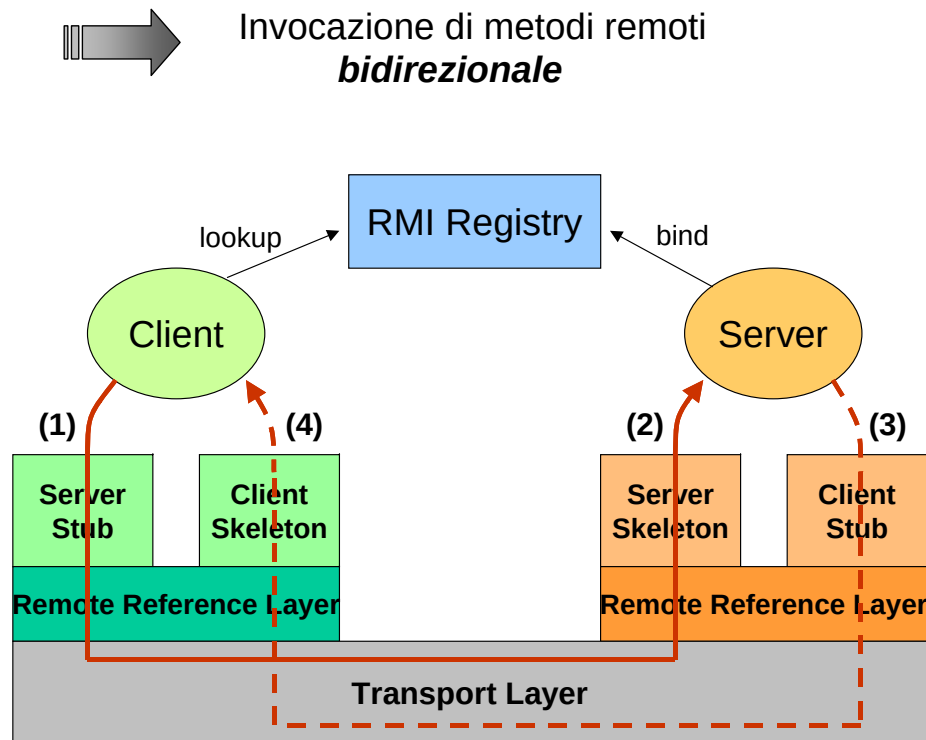
Callback



Callback con Java RMI

Client Callback: meccanismo che permette ad un client di essere *notificato dal server* al verificarsi dell'evento per cui si è registrato e messo in attesa.

Notifica del server = Invocazione di un metodo remoto del client



Come realizzare la callback

Lato client:

bisogna fornire il *metodo remoto* che il server deve invocare per fare la callback

1. definire l'interfaccia remota del client, in cui viene dichiarato il **metodo di callback** invocato dal server
2. implementare l'interfaccia remota del client, ovvero implementare il metodo di callback
3. istanziare l'implementazione dell'interfaccia remota del client e passarla al server che la utilizzerà per invocare il metodo remoto di callback

Lato server:

bisogna mantenere un riferimento al client su cui fare la callback

1. definire l'interfaccia remota del server, come sempre, in particolare è necessario definire il **metodo di de/registrazione** che il client invoca per inviare al server il proprio riferimento remoto
2. implementare l'interfaccia remota del server
3. istanziare l'implementazione dell'interfaccia remota del server e pubblicarla sul registry

Esercizio: Chat

Realizzare un'applicazione che implementi una chat, in cui:

- i **client** si registrano, mandano messaggi fino all'inserimento dell'EOF da tastiera, infine si cancellano e terminano;
- il **server** tiene traccia dei client registrati, per ogni nuova registrazione notifica a tutti i presenti nella room il nuovo ingresso, per ogni messaggio ricevuto notifica a tutti i presenti il messaggio con il mittente, per ogni uscita notifica a tutti i presenti l'uscita. Ogni notifica viene effettuata invocando una **callback**.

Implementazione del client:

Definizione dell'**interfaccia** remota del client:

```
public interface ChatClientInterface
    extends java.rmi.Remote{

    public String name()
        throws java.rmi.RemoteException;

    public void callback(String who_what)
        throws java.rmi.RemoteException;

}
```

Implementazione dell'interfaccia remota del client:

```
public class ChatClientImpl
    extends UnicastRemoteObject
    implements ChatClientInterface {

    String my_name;

    public ChatClientImpl(String name)
        throws RemoteException
    { my_name=name; }

    public String name()
        throws java.rmi.RemoteException
    { return my_name;}

    public void callback(String who_what)
        throws java.rmi.RemoteException
    { System.out.println(who_what); }

}
```

Programma client:

Creazione di una istanza di ChatClientImpl, che viene poi passata per riferimento remoto al server; il server utilizzerà tale riferimento per invocare il metodo remoto di callback:

```
public class ChatClient {

    public static void main(String args[]) {
        String hostName = null;
        int RMIPort = -1;

        /* controllo argomenti (vedi sorgenti) ... */
        try {
            InputStreamReader is =
                new InputStreamReader(System.in);
            BufferedReader br =
                new BufferedReader(is);
            String registryURL =
                "rmi://" + hostName + ":" + portNum + "/chat";

            /* recupero riferimento al server */
            ChatServerInterface server_ref =
                (ChatServerInterface) Naming.
                    lookup(registryURL);
            System.out.println("Lookup completed " );
```

```
        /* creazione istanza del client */
        System.out.println("What is your name?");
        String name = br.readLine();

        ChatClientImpl client_ref =
            new ChatClientImpl(name);

        /* registrazione client-->ingresso chat room */
        System.out.println("Registering for chat");
        server_ref.register(client_ref);

        /* ciclo di permanenza nella chat room */
        System.out.println("Enter messages,
                            EOF to exit ");

        String message;
        while((message=br.readLine())!=null){
            server_ref.saySomething(message, client_ref);
        }

        /* uscita dalla chat */
        server_ref.unregister(client_ref);
        System.out.println("Unregistered from chat.");
        System.exit(0);
    } // end try

    catch (Exception e) {
        System.out.println(
            "Exception in ChatClient: " + e);
        System.exit(1);
    }
}
```

Implementazione del server:

Definizione dell'interfaccia remota del server; si noti che sono presenti sia il metodo per la de/registrazione dei client, sia il metodo per la pubblicazione dei messaggi:

```
public interface ChatServerInterface
    extends Remote {

    public void register(
        ChatClientInterface client_ref)
        throws java.rmi.RemoteException;

    public void saySomething(String message,
        ChatClientInterface client_ref)
        throws java.rmi.RemoteException;

    public void unregister(
        ChatClientInterface client_ref)
        throws java.rmi.RemoteException;
}
```

Implementazione dell'interfaccia remota del server:

```
public class ChatServerImpl
    extends UnicastRemoteObject
    implements ChatServerInterface {

    private Vector clientList;

    public ChatServerImpl() throws RemoteException {
        super( );
        clientList = new Vector();
    }
}
```

```
/* metodo di registrazione: inserisce in una
   lista il client che vuole registrarsi */
public synchronized void register(
    ChatClientInterface client_ref)
    throws RemoteException{
    if (!(clientList.contains(client_ref))) {
        clientList.addElement(client_ref);
        System.out.println("Registered new client ");
        String callback_msg =
            "Entrance of "+client_ref.name()+"\n";
        doCallbacks(callback_msg);
    }
    else {
        System.out.println(
            "register: client already registered.");
    }
}
```

```
/* metodo di invio di un messaggio */
public void saySomething(String message,
    ChatClientInterface client_ref)
    throws java.rmi.RemoteException{
    String callback_msg =
        client_ref.name()+": "+message+"\n";
    doCallbacks(callback_msg);
}
```

```

/* metodo di cancellazione:
   elimina dalla lista il client che vuole
   cancellarsi */
public synchronized void unregister(
    ChatClientInterface client_ref)
    throws RemoteException{
    if(clientList.removeElement(client_ref)) {
        System.out.println("Unregistered client ");
        String callback_msg =
            "Exit of "+client_ref.name()+"\n";
        doCallbacks(callback_msg);
    } else {
        System.out.println(
            "unregister: client wasn't registered.");
    }
}

/* metodo che fa la callback
   a tutti i client registrati */
private synchronized void doCallbacks
    (String msg) throws RemoteException{
    System.out.println(
        "***** Callbacks execution *****\n");

    for (int i = 0; i < clientList.size(); i++){
        ChatClientInterface nextClient =
            (ChatClientInterface)clientList.elementAt(i);
        nextClient.callback(msg);
    }
}
}

```

Programma server:

Creazione e pubblicazione presso il registry di una istanza di ChatServerImpl:

```

public class ChatServer {

    public static void main(String args[]) {

        InputStreamReader is =
            new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(is);
        String registryURL; int RMIPortNum = -1;

        /* controllo argomenti (vedi sorgenti) ... */
        try{
            startRegistry(RMIPortNum);

            /* istanziazione del server e
               pubblicazione sul registry */
            ChatServerImpl server_ref =
                new ChatServerImpl();

            registryURL =
                "rmi://localhost:"+ portNum +"/chat";
            Naming.rebind(registryURL, server_ref);
            System.out.println("Chat Server ready.");
        }
        catch (Exception re) {
            System.out.println(
                "Exception in ChatServer.main: "+re);
            System.exit(1);
        }
    }
}

```

```

/* metodo che verifica se c'è già un registry
attivo sull'host e porta indicati, e se non c'è
lo crea */
private static void startRegistry(int RMIPortNum)
    throws RemoteException{
    try {
        Registry registry =
            LocateRegistry.getRegistry(RMIPortNum);
        registry.list( );
    }
    catch (RemoteException e) {
        Registry registry =
            LocateRegistry.createRegistry(RMIPortNum);
    }
}

```

Implementazione di LocateRegistry.getRegistry(String):

```

public static Registry getRegistry
    (String host)throws RemoteException{
    ...
    Registry registry = (Registry)
        sun.rmi.server.RemoteProxy.
            getStub("sun.rmi.registry.RegistryImpl",
                ObjID.REGISTRY_ID, host, port);
    ...
    return registry;
}

```

Implementazione di
LocateRegistry.createRegistry(RMIPortNum)

```

public static Registry createRegistry(int port)
    throws RemoteException{
    return new sun.rmi.registry.RegistryImpl(port);
}

```

Compilazione:

Compilazione delle interfacce remote:

```

javac ChatClientInterface.java
javac ChatServerInterface.java

```

Compilazione delle implementazioni delle interfacce,
del client e del server:

```

javac ChatServerImpl.java
javac ChatClientImpl.java
javac ChatServer.java
javac ChatClient.java

```

Compilazione con l'rmi compiler delle implementazioni
delle interfacce remote (**N.B.: sia del server che del
client!**):

```

rmic ChatServerImpl
rmic ChatClientImpl

```

per generare stub e skeleton:

```

ChatServerImpl_Skel
ChatServerImpl_Stub
ChatClientImpl_Skel
ChatClientImpl_Stub

```