Azzolini Riccardo

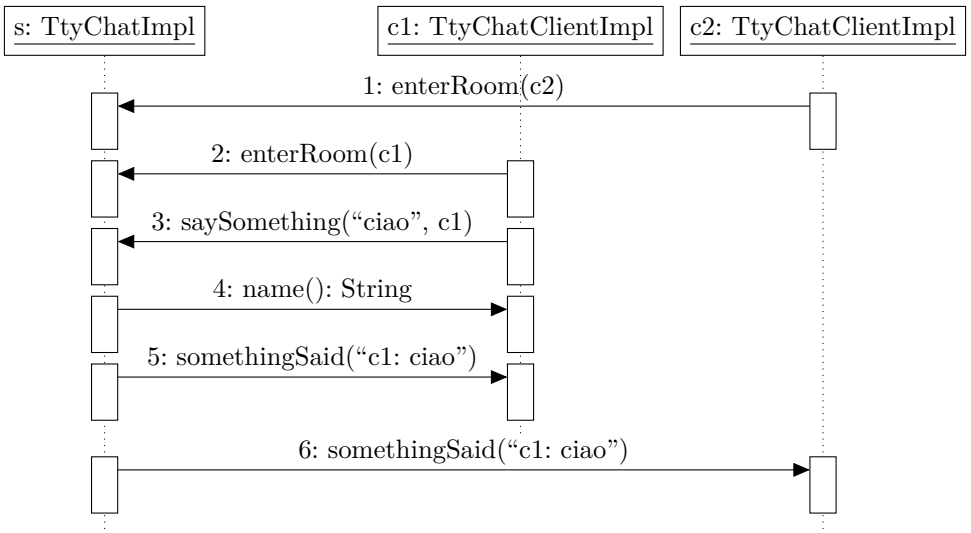
RMI callback

1 Esempio: sistema di chat

Al fine di introdurre la tecnica del *callback*, si svilupperà, come esempio, un sistema di chat (che verrà chiamato "`TtyChat`"):

- i partecipanti (client) entrano in una stanza virtuale (gestita dal server);
- ogni volta che un partecipante dice qualcosa (comunicando un messaggio al server), tutti i partecipanti che si trovano nella stanza ricevono lo stesso messaggio.

Il funzionamento dell'applicazione è esemplificato dal seguente sequence diagram:



Le chiamate remote 1, 2 e 3 sono analoghe a quelle viste finora: i client possono mandare messaggi all'oggetto server perché possiedono un riferimento remoto a quest'ultimo (reperito attraverso il registry). Invece, la caratteristica distintiva di quest'applicazione riguarda i punti 4, 5 e 6, nei quali *il server chiama metodi di un client*; per fare ciò, il server deve avere un *riferimento remoto al client*.

2 Callback server-to-client

Nel paradigma distribuito object-oriented, i ruoli di client e server non sono rigidamente fissati: non solo il client può iniziare un'interazione chiamando un metodo del server, ma è possibile anche il viceversa. Allora, la denominazione convenzionale di "client" e "server" si basa sull'interazione "prevalente": il client rimane quello che "prende l'iniziativa" più spesso (ma, appunto, non necessariamente sempre) e fruisce dei servizi forniti dal server.

Se il client è a sua volta un oggetto remoto, esso può inviare il suo riferimento al server, permettendo a quest'ultimo di chiamare i metodi remoti del client. Il termine **callback** identifica proprio un'operazione remota invocata da un oggetto che prevalentemente si comporta da server, verso un oggetto che ha un comportamento da client e che, mediante una chiamata precedente, ha passato al server il proprio riferimento.

3 Sviluppo del sistema di chat

3.1 Interfaccia remota del server

Per poter ottenere i riferimenti remoti dei client, il server definisce, nella propria interfaccia remota, dei metodi che i client invocheranno per inviare tali riferimenti (sotto forma di argomenti di tipo `TtyChatClient`):

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface TtyChat extends Remote {
    void enterRoom(TtyChatClient client) throws RemoteException;
    void saySomething(String something, TtyChatClient speaker)
        throws RemoteException;
}
```

1. Ogni client che entra nella stanza virtuale, chiama il metodo `enterRoom` per inviare il proprio riferimento remoto al server.
2. Quando un client vuole dire qualcosa, invoca il metodo `saySomething`, indicando, oltre al messaggio, anche il proprio riferimento remoto, in modo che il server sappia quale client ha "parlato".

3.2 Implementazione del server

L'interfaccia remota del server è implementata dalla seguente classe:

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.*;

public class TtyChatImpl extends UnicastRemoteObject implements TtyChat {
    private final List<TtyChatClient> occupants = new ArrayList<>();

    public TtyChatImpl() throws RemoteException {}

    public synchronized void enterRoom(TtyChatClient client) {
        occupants.add(client);
    }

    public synchronized void saySomething(
        String something, TtyChatClient speaker
    ) throws RemoteException {
        String message = speaker.name() + ": " + something;
        System.out.println(
            Thread.currentThread()
            + ":Server: received '" + message + "'"
        );

        Iterator<TtyChatClient> iter = occupants.iterator();
        while (iter.hasNext()) {
            TtyChatClient client = iter.next();
            try {
                client.somethingSaid(message);
            } catch (RemoteException e) {
                System.out.println("Someone left");
                iter.remove();
            }
        }
    }

    public static void main(String[] args) throws RemoteException {
        TtyChatImpl obj = new TtyChatImpl();
        Registry registry =
            LocateRegistry.createRegistry(Registry.REGISTRY_PORT);
```

```
            registry.rebind("TtyChat", obj);
            System.out.println("TtyChat Server bound in registry");
        }
}
```

- Le istanze di questa classe saranno automaticamente oggetti remoti, dato che essa estende `UnicastRemoteObject`.
- Il server mantiene una lista (inizialmente vuota) di riferimenti remoti ai client presenti nella stanza (`occupants`).
- `enterRoom` riceve un riferimento remoto al client chiamante e lo salva semplicemente nella lista `occupants`.
- `saySomething` riceve, da un client che vuole "parlare", ciò che vuole dire e il riferimento remoto al client stesso (`speaker`), dopo di che:
  1. chiede allo `speaker` quale sia il suo nome (mediante una chiamata remota a un opportuno metodo `name` offerto dal client) e lo aggiunge come prefisso al messaggio;
  2. prova a inviare il messaggio a tutti i client presenti nella stanza (con l'invocazione remota di un altro metodo dei client, `somethingSaid`), rimuovendo dalla lista `occupants` quelli eventualmente non più raggiungibili (cioè quelli che sono usciti);
- `enterRoom` e `saySomething` devono essere `synchronized` per evitare race condition sulla lista `occupants`, dato che client diversi potrebbero effettuare chiamate remote contemporanee (cosa che RMI permette creando automaticamente un thread server per la gestione di ciascun client).
- Il main deve semplicemente creare un oggetto, che è già remoto in quanto sottoclasse di `UnicastRemoteObject`, e registrarlo presso il registry.

3.3 Interfaccia remota del client

Perché sia possibile il callback, anche il client deve essere un oggetto remoto, e quindi avere un'interfaccia remota, che è il tipo `TtyChatClient` già usato dal server:

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface TtyChatClient extends Remote {
    String name() throws RemoteException;
    void somethingSaid(String something) throws RemoteException;
}
```

3.4 Implementazione del client

Il codice dell'oggetto remoto client è:

```
import java.io.*;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class TtyChatClientImpl
        extends UnicastRemoteObject implements TtyChatClient {
    private final String myName;

    public TtyChatClientImpl(String myName) throws RemoteException {
        this.myName = myName;
    }

    public String name() {
        return myName;
    }

    public void somethingSaid(String something) {
        System.out.println(something);
    }

    public static void main(String[] args) throws Exception {
        try (
            BufferedReader input =
                new BufferedReader(new InputStreamReader(System.in))
        ) {
            System.out.println("What is your name?");
            TtyChatClientImpl me =
                new TtyChatClientImpl(input.readLine());

            Registry registry = LocateRegistry.getRegistry();
            TtyChat server = (TtyChat) registry.lookup("TtyChat");
            server.enterRoom(me);
            System.out.println("You can now chat in the room");

            while (true) {
                server.saySomething(input.readLine(), me);
            }
        }
```

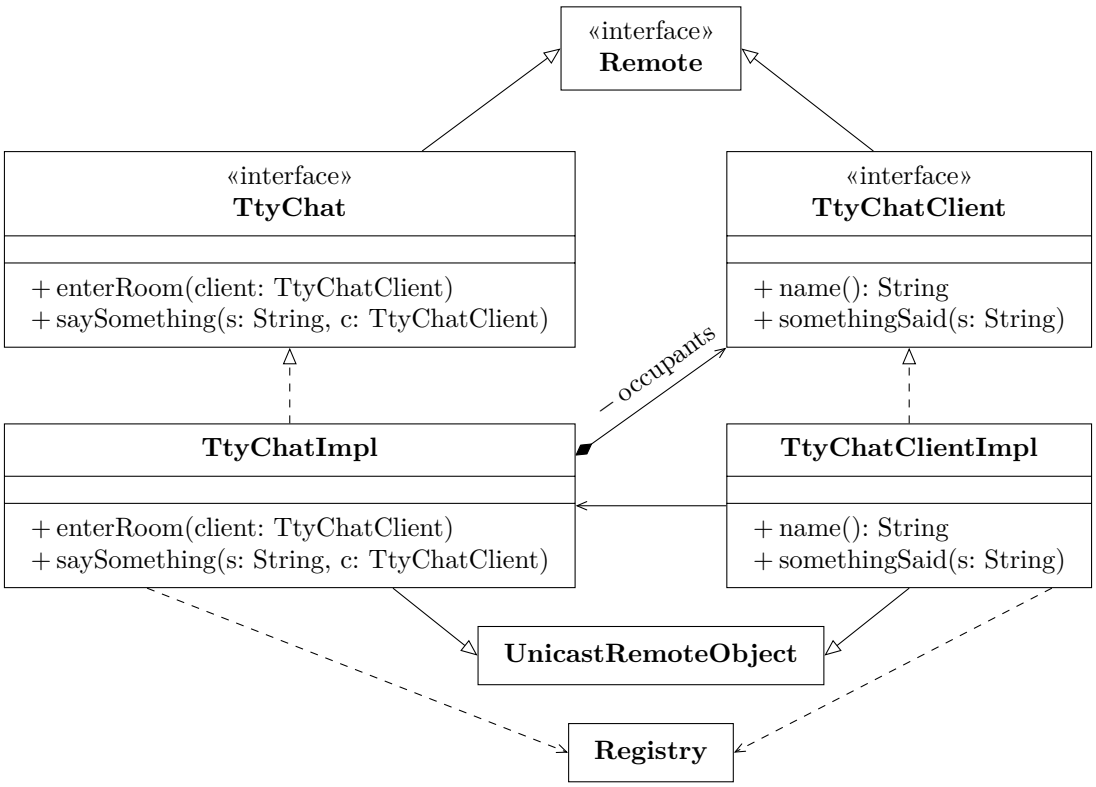
```
    }
}
```

L'implementazione dei metodi dell'interfaccia remota è banale. La parte interessante è invece il main, nel quale:

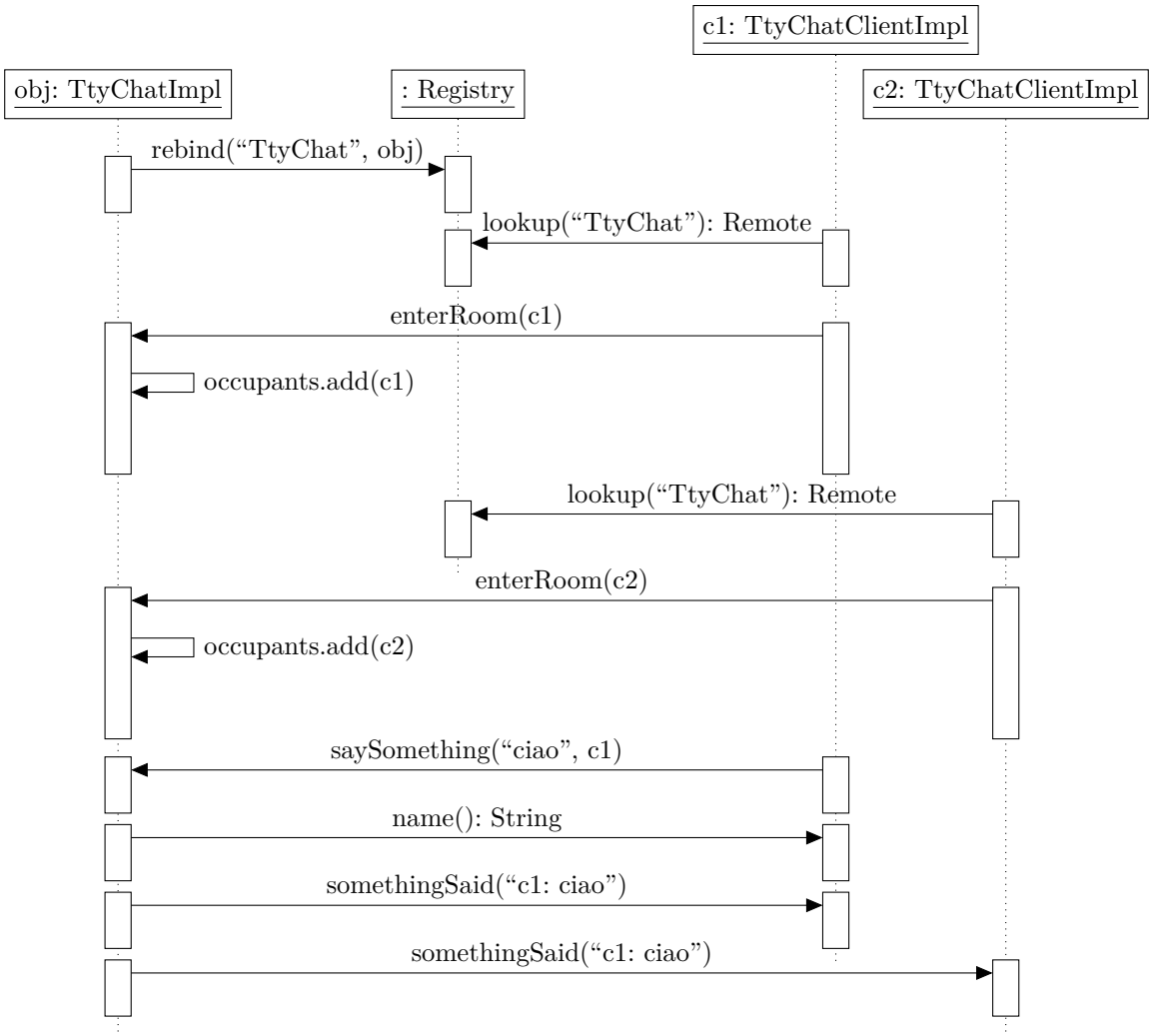
1. Si crea un oggetto remoto (con un nome letto da tastiera).
2. Si ottiene dal registry un riferimento remoto al server, come al solito.
3. Si invia al server un riferimento remoto all'oggetto client creato prima, attraverso il metodo `enterRoom`.
4. In un ciclo infinito, si inviano al server (attraverso `saySomething`) tutti i messaggi che l'utente inserisce da tastiera. Contemporaneamente, il server chiama `somethingSaid` ogni volta che c'è un messaggio da visualizzare.

3.5 Classi e comportamento del sistema

Il class diagram che rappresenta il sistema appena sviluppato è il seguente:



Invece, questo sequence diagram mostra (più nel dettaglio, rispetto al sequence diagram presentato all'inizio) un esempio di comportamento del sistema (a partire dal momento in cui gli oggetti remoti sono già stati creati):



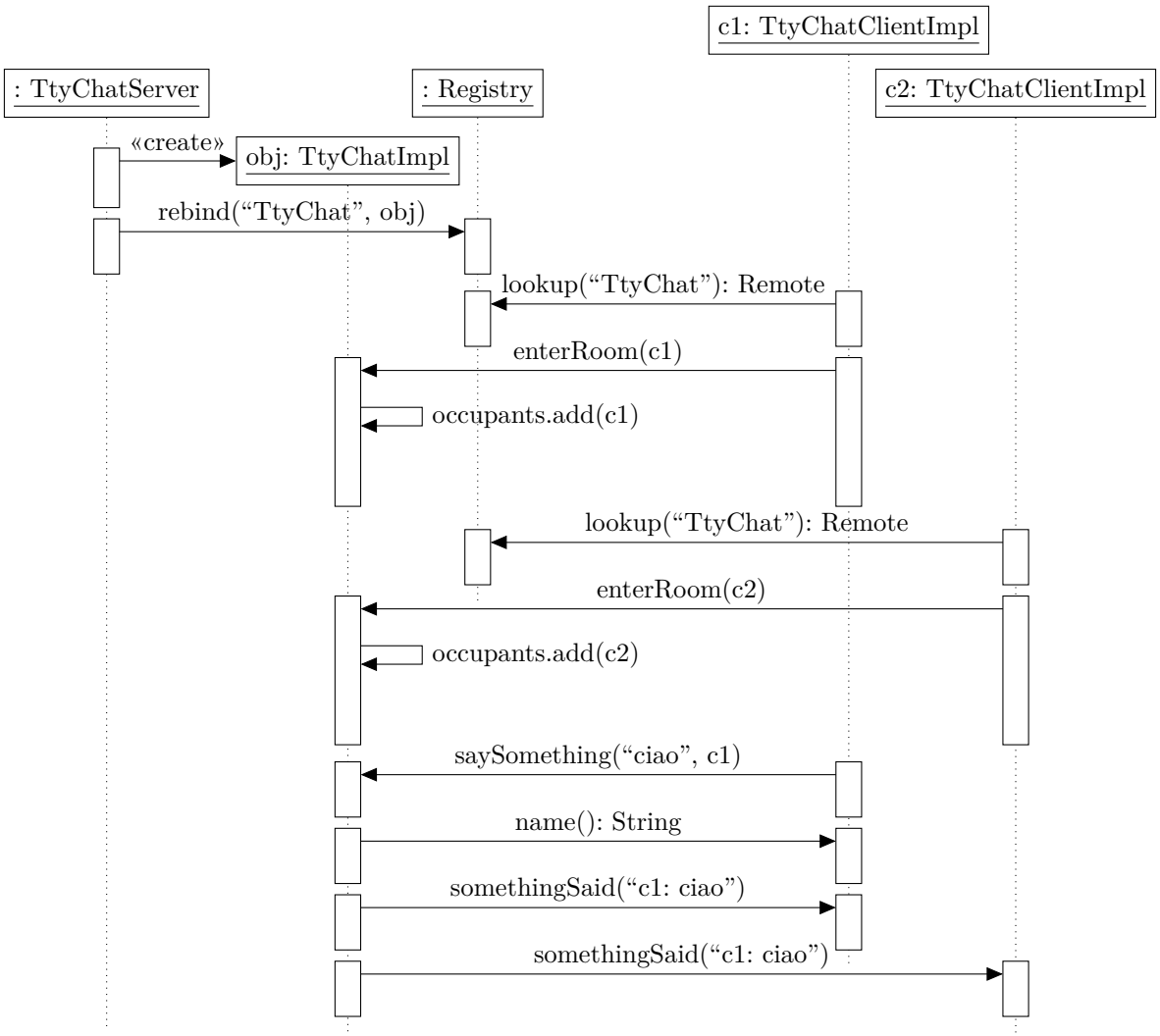
4 Alcune modifiche

Il programma appena visto è un'implementazione minimale di un sistema di chat. Due semplici modifiche che vi si possono apportare per migliorarlo sono:

- permettere ai client di uscire dalla stanza virtuale (attraverso un nuovo metodo `exitRoom`) e terminare in modo ordinato;

- separare il server dall'oggetto remoto che implementa l'interfaccia, "estraendo" in una nuova classe `TtyChatServer` il codice che gestisce la creazione e la registrazione dell'oggetto remoto.

Con la separazione del server dall'oggetto remoto, il sequence diagram che rappresenta il comportamento del sistema diventa:



4.1 Interfaccia remota del server

All'interfaccia remota del server viene aggiunto, come anticipato, il metodo `exitRoom`, che permette a un client di comunicare che lascia la stanza, e che quindi, da questo momento in avanti, esso non dovrà più ricevere notifiche riguardo ai messaggi inviati dagli altri client:

```java
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface TtyChat extends Remote {
    void enterRoom(TtyChatClient client) throws RemoteException;
    void exitRoom(TtyChatClient client) throws RemoteException;
    void saySomething(String something, TtyChatClient speaker)
        throws RemoteException;
}
```

4.2 Implementazione dell'oggetto remoto server

Alla classe `TtyChatImpl` viene aggiunta l'implementazione del metodo `exitRoom`, mentre si toglie il main, che verrà spostato in una classe a parte:

```java
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.*;

public class TtyChatImpl extends UnicastRemoteObject implements TtyChat {
    private final List<TtyChatClient> occupants = new ArrayList<>();

    public TtyChatImpl() throws RemoteException {}

    public synchronized void enterRoom(TtyChatClient client) {
        occupants.add(client);
    }

    public synchronized void exitRoom(TtyChatClient client) {
        occupants.remove(client);
    }

    public synchronized void saySomething(
        String something, TtyChatClient speaker
    ) throws RemoteException {
        // Uguale a prima
    }

    // Non c'è più il main
}
```

4.3 Classe server

L'oggetto remoto `TtyChatImpl` viene creato dalla nuova classe `TtyChatServer`, che contiene il main (identico a prima):

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.RemoteException;

public class TtyChatServer {
    public static void main(String[] args) throws RemoteException {
        TtyChatImpl obj = new TtyChatImpl();
        Registry registry =
            LocateRegistry.createRegistry(Registry.REGISTRY_PORT);
        registry.rebind("TtyChat", obj);
        System.out.println("TtyChat Server bound in registry");
    }
}
```

4.4 Implementazione dell'oggetto remoto client

Per quanto riguarda il client, è necessario modificare solo il main:

```
import java.io.*;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class TtyChatClientImpl
        extends UnicastRemoteObject implements TtyChatClient {
    // ...uguale a prima

    public static void main(String[] args) throws Exception {
        try (
            BufferedReader input =
                new BufferedReader(new InputStreamReader(System.in))
        ) {
            System.out.println("What is your name?");
            TtyChatClientImpl me =
                new TtyChatClientImpl(input.readLine());

            Registry registry = LocateRegistry.getRegistry();
```

```
            TtyChat server = (TtyChat) registry.lookup("TtyChat");
            server.enterRoom(me);
            System.out.println("You can now chat in the room");

            while (true) {
                String s = input.readLine();
                if (s.equals("<quit>")) {
                    break;
                }
                server.saySomething(s, me);
            }

            server.exitRoom(me);
            UnicastRemoteObject.unexportObject(me, false);
        }
    }
}
```

Quando si legge il messaggio da inviare, se esso è `<quit>`:

1. il ciclo di lettura e invio dei messaggi termina;
2. si chiama il metodo `exitRoom` (inviando il riferimento remoto all'oggetto client) per segnalare al server l'uscita dalla stanza virtuale;
3. si rende non più remoto l'oggetto client, consentendo al processo di terminare.