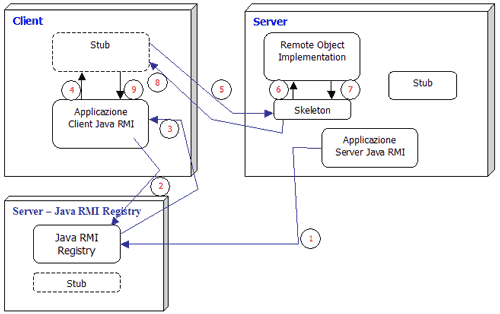
**­Figura 1. Interazione tra i vari componenti in gioco in RMI di Java**

****

Come si vede dal diagramma, RMI si basa sull’interazione tra tre entità distinte:

* Uno o più Server RMI (per semplicità ne considereremo uno)
* Il Java RMI Registry (localizzato sul server)
* Uno o più Client RMI (per semplicità ne considereremo uno)

Il **Server RMI** implementa un’interfaccia relativa ad un particolare oggetto RMI e registra tale oggetto nel Java RMI Registry. Il Java RMI Registry è, semplicemente, un processo di tipo daemon che tiene traccia di tutti gli oggetti remoti disponibili su un dato server.

Il **Client RMI** effettua una serie di chiamate al registry RMI per ricercare gli oggetti remoti con cui interagire.

A questo punto ci si potrebbe porre un paio di domande:

* Come fa un oggetto che si trova su una macchina client ad invocare dei metodi che sono definiti su un’altra macchina (server) ?
* Chi si occupa di gestire le problematiche legate alla comunicazione di rete?

Il trucco è tutto da ricercare nei due componenti che nel diagramma precedente sono denominati Skeleton e Stub.

Lo **Skeleton**, sul server, si occupa di interagire direttamente con l’oggetto RMI che espone i metodi “remotizzati”, inviando a quest’ultimo tutte le richieste provenienti dallo Stub.

Lo **Stub**, invece, rappresenta una sorta di classe “clone” che ripropone e mette a disposizione del client tutti i metodi che sul server sono stati definiti e implementati come remoti. In altre parole, lo Stub fa le veci di una classe spesso denominata “proxy class”.

Sia lo Skeleton sia lo Stub si occupano, infine, in modo trasparente all’utente (e al programmatore) della gestione della comunicazione tra il client ed il server.

Vediamo ora di dettagliare, passo per passo, il meccanismo descritto nel diagramma iniziale, utilizzando la medesima numerazione riportata in figura.

1. Viene creata sul server una istanza dell’oggetto remoto e passata in forma di stub al Java RMI registry. Tale stub viene, quindi registrato all’interno del registry stesso.
2. L’applicazione client richiede al registry RMI una copia dell’oggetto remoto da utilizzare.
3. Il Java RMI registry restituisce una copia serializzata dello stub al client
4. L’applicazione client invoca uno dei metodi dell’oggetto remoto utilizzando la classe “clone” fornita dallo stub
5. Lo stub richiama lo skeleton che si trova sul server chiedendogli di invocare sull’oggetto remoto lo stesso metodo che il client ha invocato sullo stub
6. Lo skeleton invoca il metodo richiesto sull’oggetto remoto
7. L’invocazione del metodo sull’oggetto remoto restituisce il risultato allo skeleton
8. Lo skeleton comunica il risultato allo stub sul client
9. Lo stub fornisce il risultato all’applicazione client iniziale

**Creazione della applicazione Server**

Per ogni interfaccia remota definita (nel nostro semplice caso, una soltanto) è necessario creare una classe sul server che contenga l’implementazione dell’oggetto remoto vero e proprio. Tale classe deve:

Derivare dalla classe **java.rmi.server.UnicastRemoteObject**

Implementare tutti i metodi definiti nell’interfaccia remota (ma questa è una regola sempre valida in Java)

Tutti i costruttori definiti nella classe devono “lanciare” un’eccezione del tipo java.rmi.RemoteException. Si noti che se si decidesse di utilizzare soltanto il costruttore di default, sarà comunque necessario scriverne il codice per gestire proprio la throw dell’eccezione RemoteException.

**Creazione dell’applicazione Client**

Il compito dell’applicazione client è ricercare l’applicazione server che espone gli oggetti remotizzati messi a disposizione attraverso RMI ed invocarne, quindi, i metodi opportuni.