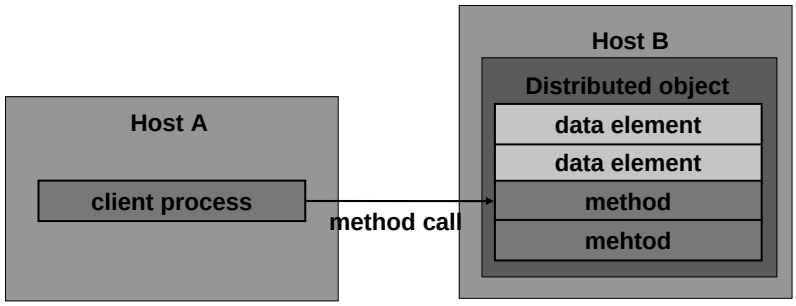
Oggetti Distribuiti e Java RMI

Oggetti Locali - Oggetti Distribuiti

- Oggetti Locali: sono oggetti i cui metodi possono essere invocati solo da un processo locale, cioè da un processo in esecuzione sulla stessa macchina su cui risiede l'oggetto.
- Oggetti distribuiti (remoti): Oggetti i cui metodi possono essere invocati da un processo remoto, cioè un processo in esecuzione su una macchina connessa attraverso una rete alla macchina su cui risiede l'oggetto.

Paradigma ad Oggetti Distribuiti

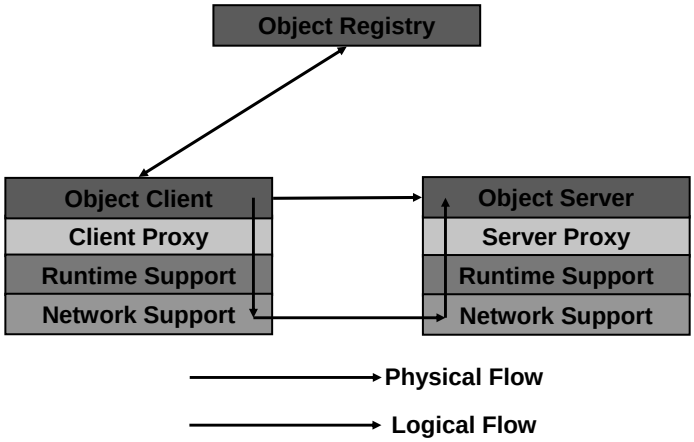
- Le risorse di rete sono rappresentate da oggetti distribuiti
- Per richiedere un servizio ad una risorsa di rete, un processo invoca uno dei metodi o operazioni messe a disposizione, passando al metodo i dati come parametri.
- Il metodo viene eseguito sull'host remoto, e la risposta è inviata al processo richiedente come valore di ritorno.



Paradigma ad Oggetti Distribuiti - 2

- Un processo in esecuzione su un host A chiama un metodo su un oggetto distribuito che risiede su un host B, passando allo stesso eventuali parametri
- La chiamata del metodo invoca un' azione che il metodo eseguirà sull' host B. Il valore di ritorno, se c'è, verrà trasferito dall' host B all' host A
- Un processo che fa uso di un oggetto distribuito viene detto processo client di quell' oggetto, e i metodi dell' oggetto sono chiamati metodi remoti (contrariamente ai metodi locali, o metodi appartenenti ad un oggetto locale)

Architettura di un Sistema ad Oggetti Distribuiti



Sistemi ad Oggetti Distribuiti

- Un oggetto distribuito è esposto (messo a disposizione) attaverso un processo chiamato oggetto server
- Un oggetto detto registry deve essere presente nell'architettura per sistemi distribuiti affinché gli oggetti distribuiti vengano registrati
- Un riferimento permette ad un oggetto remoto di essere localizzato, cioè di localizzare la macchina su cui l'oggetto risiede
- Per poter accedere ad un oggetto distribuito, un processo – cioè un oggetto client – si serve del registry per ottenere un riferimento all'oggetto. Questo riferimento è usato dall'oggetto client per chiamare i metodi sull'oggetto remoto.

Sistemi ad Oggetti Distribuiti - 2

- Logicamente, l'oggetto client fa una chiamata direttamente al metodo remoto.
- Realmente, la chiamata è gestita da un componente software, chiamato client proxy, il quale interagisce con il software sul client host che fornisce il supporto a runtime per il sistema di oggetti distribuiti.
- Il supporto a runtime si fa carico delle chiamate inoltrate dal proxy verso l'oggetto remoto, includendo il marshalling dei parametri che verranno trasmessi all' oggetto remoto

Sistemi ad Oggetti Distribuiti - 3

- Un'architettura simile è richiesta anche sul lato server, dove il supporto a runtime per il sistema ad oggetti distribuiti gestisce la ricezione dei messaggi e l' unmarshalling dei dati, e inoltra la chiamata ad un componente software chiamato server proxy
- Il server proxy comunica con l'oggetto distribuito ed invoca il metodo chiamato, passandogli come argomenti i dati unmarshalled
- L'esecuzione del metodo sull'host remoto, include il marshallind del valore di ritorno, il quale è inoltrato dal server proxy al client proxy, attraverso il supporto runtime e il supporto si rete.

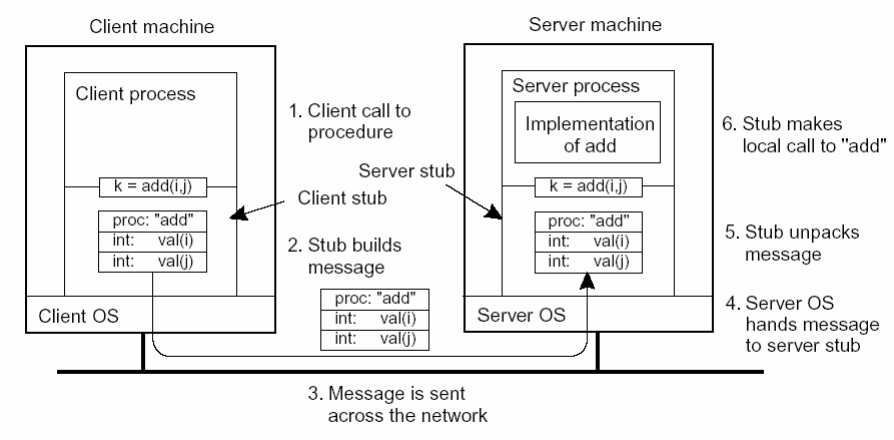
Alcuni meccanismi ad oggetti distribuiti

- Esistono diversi meccanismi basati sul paradigma ad oggetti distribuiti:
  - Java Remote Method Invocation (RMI)
  - Common Object Request Broker Architecture (CORBA)
  - Distributed Component Object Model (DCOM)
  - Meccanismi che supportano Simple Object Access Protocol (SOAP)

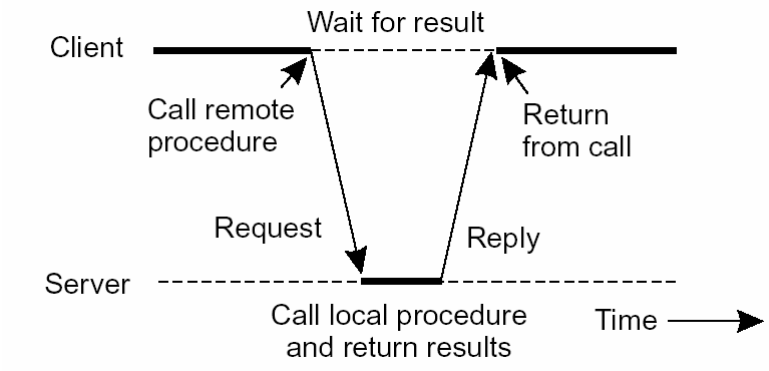
Remote Procedure Call (RPC)

- Remote Method Invocation ha origine da un paradigma chiamato Remote Procedure Call
- Nel modello RPC, una “procedure call” viene fatta da un processo ad un altro, con i dati passati come argomenti
- Alla ricezione della chiamata, le azioni previste nella procedura vengono eseguite, al chiamante viene notificato l’avvenuto completamento della chiamata, ed un valore di ritorno, se c’è, viene trasmesso dal processo remoto al processo chiamante

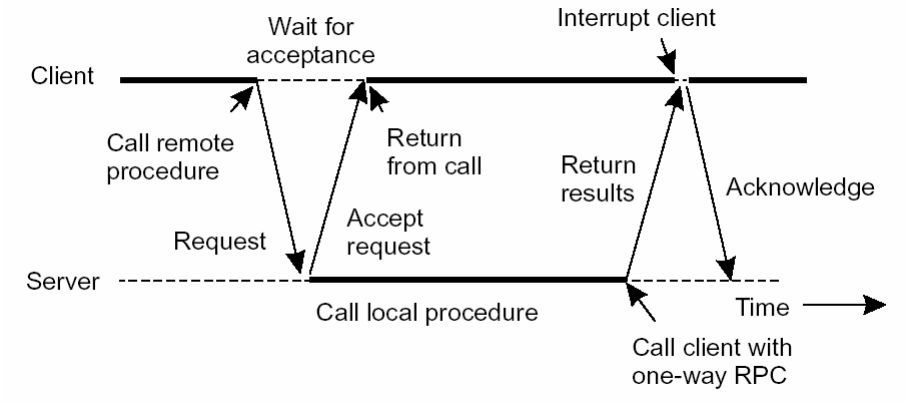
RPC: Remote Computation



RPC: Remote Computation - 2



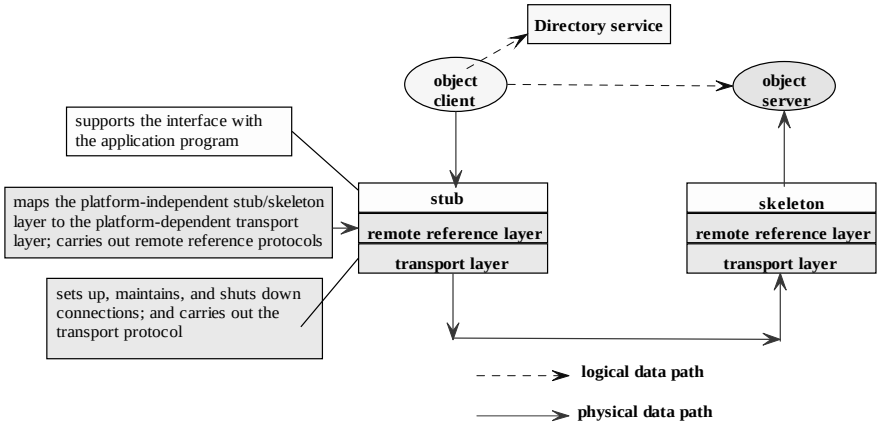
RPC: Remote Computation - 3



Remote Method Invocation

- RMI è un implemetazione object-oriented del modello Remote Procedure Call; è un API per soli programmi Java
- Usando RMI, un oggetto server esporta un oggetto remoto registrandolo (pubblicandolo) in una directory service. L'oggetto fornisce i metodi remoti, i quali possono essere invocati nei programmi client
- Sintatticamente:
  - Un oggetto remoto è dichiarato con una interfaccia remota
  - L'interfaccia remota è implementata dall' oggetto remoto
  - Un oggetto client accede all'oggetto remoto invocando i metodi remoti associati all'oggetto stesso

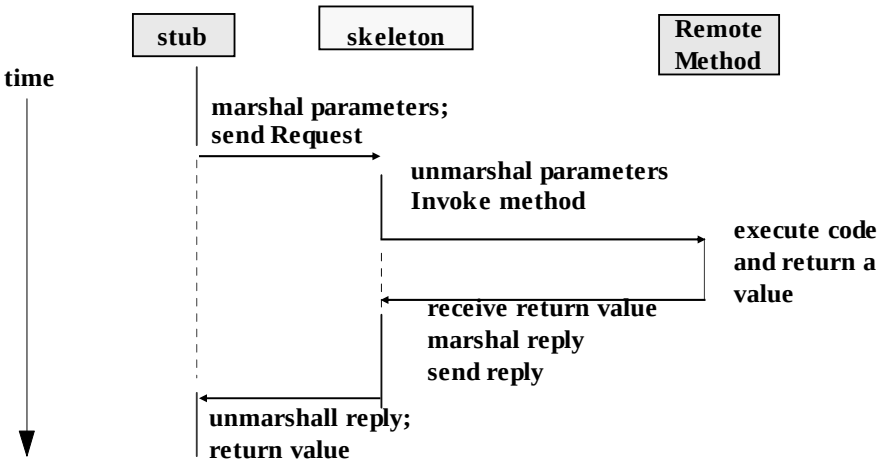
Java RMI - Architettura



Oggetto Registry

- Le API RMI permettono di usare diversi tipi di directory service per registare un oggetto distribuito
- Noi useremo un semplice directory service chiamato RMI registry, rmiregistry, il quale è fornito direttamente con Java Software Development Kit (SDK)
- L' RMI Registry è un servizio, che se attivo, deve essere in esecuzione sulla macchina host su cui risiede l'oggetto server (oggetto che pubblica l'oggetto remoto), e per default riceve sulla porta 1099
- Un alternativa ad rmiRegistry è Java Naming and Directory Interface (**JNDI**), il quale è più generale di RMI registry, nel senso che può essere utilizzato da applicazioni che non usano RMI

Interazioni tra stub e skeleton



(based on http://java.sun.com.marketing/collateral/javarim.html)

Le API per Java RMI

- L'Interfaccia Remota
- Il Software Server-side
  - Implementazione dell' Interfaccia Remota
  - Generazione dello Stub
  - L' Oggetto Server
- Il Softawe Client-side

L’Interfaccia Remota

- Un’Interfaccia Remota è un’ interfaccia che estende l’interfaccia java.rmi.Remote
  - Tale interfaccia non contiene alcun metodo ma serve solo come “marcatore”, cioè che la classe è utilizzabile per accedere ad un oggetto remoto(cioè invocarne i metodi) e protegge l’applicazione da anomalie derivanti dall’utilizzo di risorse remote.
- Oltre che estendere l’interfaccia Remote, ogni metodo dell’interfaccia remota deve poter generare una Remote Exception, ma la sintassi dell’interfaccia remota è la stessa di una interfaccia Java locale

Un esempio di interfaccia remota

```
import java.rmi.*
public interface SomeInterface extends Remote {
    public String someMethod1( )
        throws java.rmi.RemoteException;

    public int someMethod2( float f) throws
              java.rmi.RemoteException;

   [signature of other remote methods may follow]
}
```

Un esempio di interfaccia remota - 2

- L'eccezione java.rmi.RemoteException deve essere inserita nella clausola *throw* di ogni metodo dell'interfaccia
- Questa eccezione viene sollevata quando si verificano errori nella chiamata remota
- Causa di queste eccezioni possono essere errori durante la comunicazione, come il fallimento di una connessione, o errori dovuti al marshalling e quindi allo stub

Implementazione dell' Interfaccia Remota

```
import java.rmi.*;
import java.rmi.server.*;

public class SomeImpl extends UnicastRemoteObject
     implements SomeInterface {
   public SomeImpl() throws RemoteException {
      super( );
   }
   public String someMethod1( ) throws RemoteException {
      // code to be supplied
   }
   public int someMethod2( float f )
          throws RemoteException {
      // code to be supplied
   }
}
```

Generare la classe Stub

- In RMI, ogni oggetto distribuito richiede un proxy sia per l'oggetto server che per l'oggetto client
- Questo proxy è generato a partire dall' implementazione dell'interfaccia remota usando un tool fornito con la JDK: RMI compiler *rmic*
  - **rmic –v1.2 SomeImpl**
- Come risultato della compilazione, verrà generato la classe proxy, il cui nome avrà come prefisso il nome della classe che implementa il servizio:
  - **SomeImpl_Stub.class**

L'Oggetto Server

L'oggetto server crea un'istanza dell' oggetto che implementa l'interfaccia remota ed esporta lo stesso sul Registry

```
import java.rmi.*;

public class SomeServer  {
   public static void main(String args[]) {
    […]
    try{
      SomeImpl exportedObj = new SomeImpl();
      Naming.rebind("some", exportedObj);
          System.out.println("Some Server ready.");
       }
    […]
    }
}
```

L’Oggetto Server - 2

- La classe Naming fornisce metodi per memorizzare e ottenere riferimenti ad oggetti remoti dal registro
  - Il metodo rebind (String name, Remote obj): associa un nome (servizio) ad un oggetto remoto, sostituendo qualsiasi associazione esistente.
  - Il metodo rebind sovrascriverà il nome se nel registro ne esiste uno uguale
    - Per evitare ciò, si può anche usare anche il methodo bind
  - Il nome dell’host dovrebbe essere il nome del server o semplicemente localhost. Il nome di riferimento dell’ oggetto remoto dovrebbe essere unico nel registro

RMI Registry

- L’RMI Registry (per motivi di sicurezza) deve essere eseguito sull’host dove risiede il server che esporta (pubblica) l’oggetto remoto
- Esso può essere attivato usando l’utility *rmiregistry* fornita con la JDK, nel seguente modo:
  rmiregistry <port number>
  dove <port number> rappresenta il numero di porta
- Se non viene specificato nessun numero di porta, allora rmiregistry userà la porta di default 1099
- Il registro rimarrà in esecuzione in modo continuato. (Per stoppare si può usare CTRL-C)

L’Oggetto Server - 3

- Quando un oggetto server viene eseguito, la pubblicazione dell’oggetto distribuito fa si che l’oggetto server inizi ad essere in ascolto aspettando connessioni da parte dei client che richiedono l’esecuzione dei metodi
- Un oggetto server RMI è un server concorrente: ogni richiesta da un oggetto client è servita usando un thread separato
- Nota che se un processo client invoca più chiamate a metodi remoti, queste chiamate verranno eseguite in modo concorrente, a meno che non vengano presi provvedimenti dal processo client per sincronizzare le chiamate

Software Client-side

- La programmazione della classe client è simile alla programmazione di una normale classe Java
- La sintassi necessaria per la richiesta RMI
  - localizzazione dell’ RMI Registry nel server host
  - Eseguire il looking up del riferimento all’oggetto remoto
  - Si esegue il casting del riferimento rispetto all’interfaccia remota. Su di essa può allora essere invocato il metodo remoto

Software Client-side - 2

```
import java.rmi.*;

public class SomeClient {
   public static void main(String args[]) {
       […]
       try {
       String registryURL =
         "rmi://localhost:" + portNum + "/some";
       SomeInterface h =
         (SomeInterface)Naming.lookup(registryURL);
       String result = h.someMethod1();
       […]
       }
       catch (Exception e) {[…]}
   }
}
```

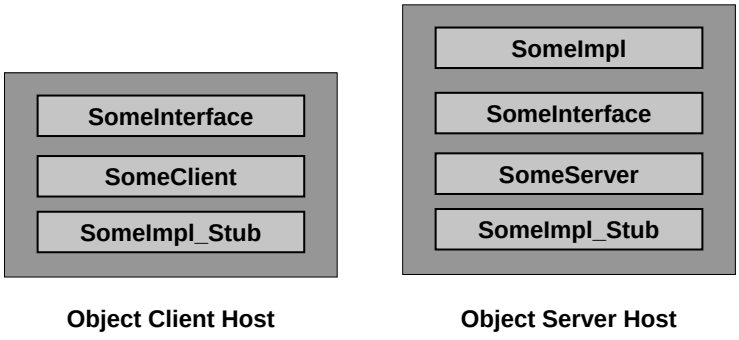
Invocare un Metodo Remoto

- La sintassi per l'invocazione di metodi remoti è la stessa dei metodi locali
- Un errore comune è il casting dell'oggetto prelevato dal registry alla classe dell'oggetto che implementa l'interfaccia o dell'oggetto server
- Si deve castizzare rispetto all' interfaccia

Stub dell’oggetto remoto

- Il file Stub dell’oggetto, così come il file dell’interfaccia remota, deve essere condiviso con l’oggetto client in quanto questi file sono richiesti dall’oggetto client in fase di compilazione
- La copia di questi file deve essere fornito all’oggetto client manualmente
- In alternativa, Java RMI ha una caratteristica chiamata stub downloading il quale permetta ad un file Stub di essere ottenuto dal client dinamicamente

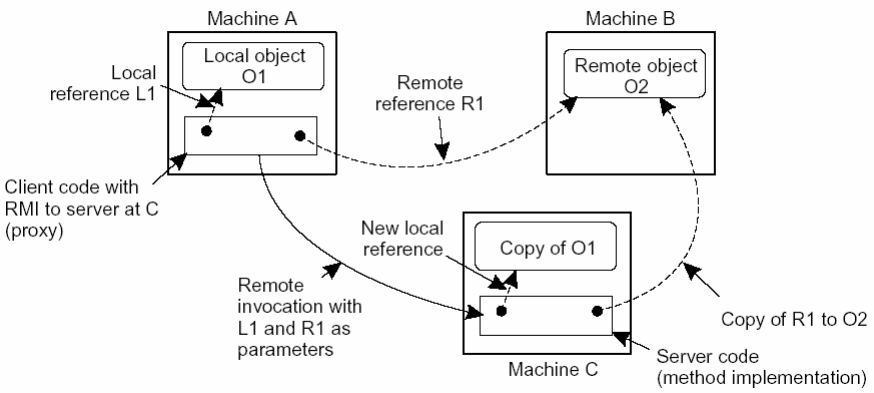
Disposizione delle classi per una applicazione RMI



RMI - Passaggio dei Parametri

- **Tipi primitivi**
  - **passati per valore;**
- **Oggetti remoti**
  - **passati per riferimento;**
- **Oggetti non remoti**
  - **passati per valore;**
  - **si usa la Java Object Serialization (Serializzazione).**

Riferimenti Locali e Remoti



RMI e Sockets

- Remote Method Invocation può essre usato in luogo dei Socket in una applicazione di rete
- Alcune differenze tra le RMI e i Socket :
  - La comunicazione esplicita tra socket è più efficiente, ma richiede più tempo per la programmazione (classi proxy)
  - RMI garantisce un livello di astrazione maggiore con una conseguente aumento di leggibilità. Quando la complessità dell'applicazione aumenta può divenire indispensabile
  - Con RMI tutti i componenti di una applicazione distribuita devono essere necessariamente scritti in Java