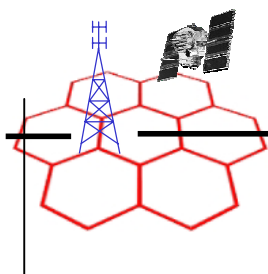




Università degli Studi di Roma “La Sapienza”

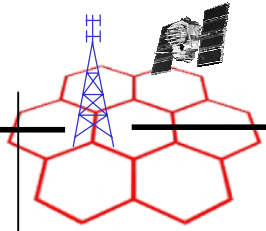
Dipartimento di Informatica e Sistemistica



Sviluppo di applicazioni distribuite

**Progetto di Reti di Calcolatori, a.a. 2003/2004**

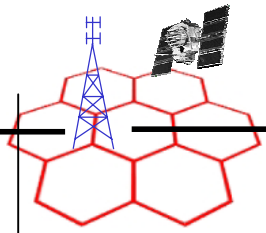
Emiliano Trevisani, Andrea Vitaletti



# Overview

---

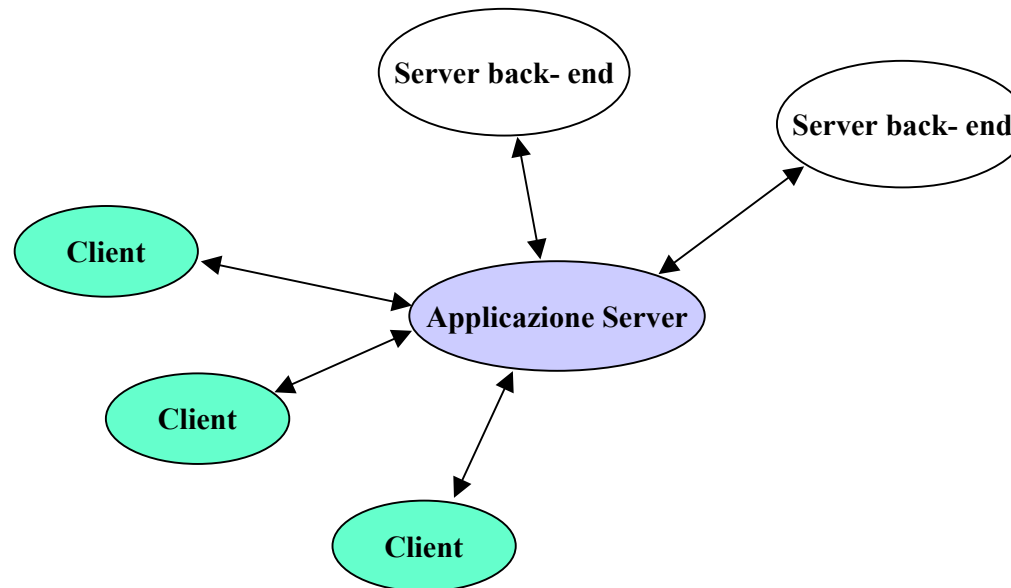
- ❑ Un'applicazione distribuita è un'applicazione la cui esecuzione è distribuita tra più host di una rete;
- ❑ Vantaggi:
  - Maggiore efficienza attraverso un uso ottimale delle risorse;
  - Elevato parallelismo;
  - Tolleranza ai guasti;
  - Bilanciamento del carico;
- ❑ Approcci allo sviluppo di applicazioni distribuite:
  - **Client / Server** [Internet like]
  - **DCE** [Distributed Computing Environment]
  - **DCOM** [Distributed Component Object Model]
  - **CORBA** [Common Object Request Broker Architecture]
  - **Java RMI** [Remote Method Invocation]

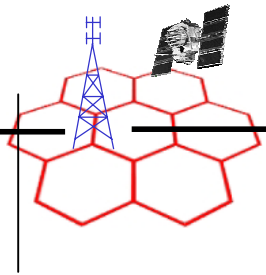


# Client / Server

---

- ❑ L'utilizzo diffuso dello stack TCP/IP ha consentito la crescita delle applicazioni distribuite eseguite nel supporto di rete client – server offerto da Internet;
- ❑ Questa prima generazione di applicazioni distribuite prevede la comunicazione attraverso protocolli **specifici** delle particolari applicazioni [HTTP, FTP,...];
- ❑ Un'applicazione client è tipicamente eseguita su un host ed utilizza, in generale, il protocollo TCP per aprire un canale di comunicazione con un'applicazione server in esecuzione su un host remoto;
- ❑ Oggetto della comunicazione è la richiesta di servizi ed il risultato dell'elaborazione lato server viene inoltrata verso il client;

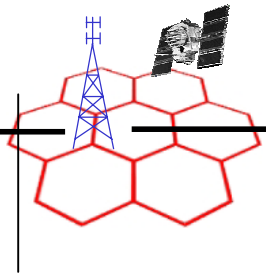




# DCE

---

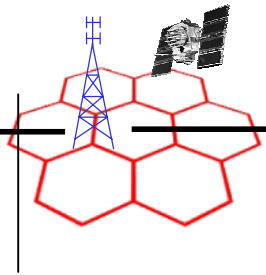
- ❑ Modello: i sistemi distribuiti sono organizzati in **celle**;
- ❑ Una cella è costituita da **un gruppo di risorse di calcolo**;
- ❑ I **servizi DCE** di una cella sono utilizzati per **realizzare applicazioni distribuite**;
- ❑ Uno dei servizi più comune è il servizio **RPC [Remote Procedure Call]** che sostituisce i socket TCP come meccanismo di base per la comunicazione client – server; in effetti le RPC sono implementate come livello costruito sopra quello di trasporto di TCP e gestiscono in modo trasparente le connessioni e gli eventi specifici del protocollo;
- ❑ DCE **non è un modello orientato agli oggetti** ma piuttosto alle **procedure**; questo non si adatta bene ad un ambiente orientato agli oggetti come è, ad esempio, JAVA;
- ❑ Al DCE ci si riferisce spesso come **middleware**: non si tratta di un prodotto a sé stante ma piuttosto una raccolta di servizi integrati in un ambiente operativo;
- ❑ I servizi DCE sono utilizzati come approccio alternativo per lo sviluppo di applicazioni distribuite rispetto all'approccio precedente;



# DCOM

---

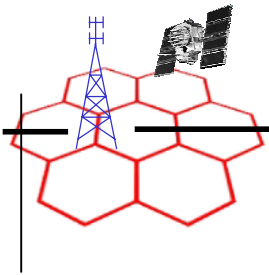
- ❑ La tecnologia DCOM consente ad oggetti in esecuzione su un host locale di **creare** oggetti su host remoti e di **invocarne i metodi**;
- ❑ La localizzazione dell'host remoto è **trasparente** a DCOM mentre **gli oggetti remoti sono utilizzati esattamente come oggetti locali**;
- ❑ Affinché un oggetto remoto sia utilizzabile da un oggetto locale, la classe dell'oggetto remoto deve essere registrata nel registro locale della macchina locale;
- ❑ L'astrazione è tale che l'oggetto locale ignora la localizzazione dell'oggetto del quale sta invocando i metodi;
- ❑ DCOM è un prodotto Microsoft ed una delle sue principali caratteristiche è il supporto della sicurezza;
- ❑ Sebbene basato su RPC **offre il supporto alla programmazione ad oggetti**;



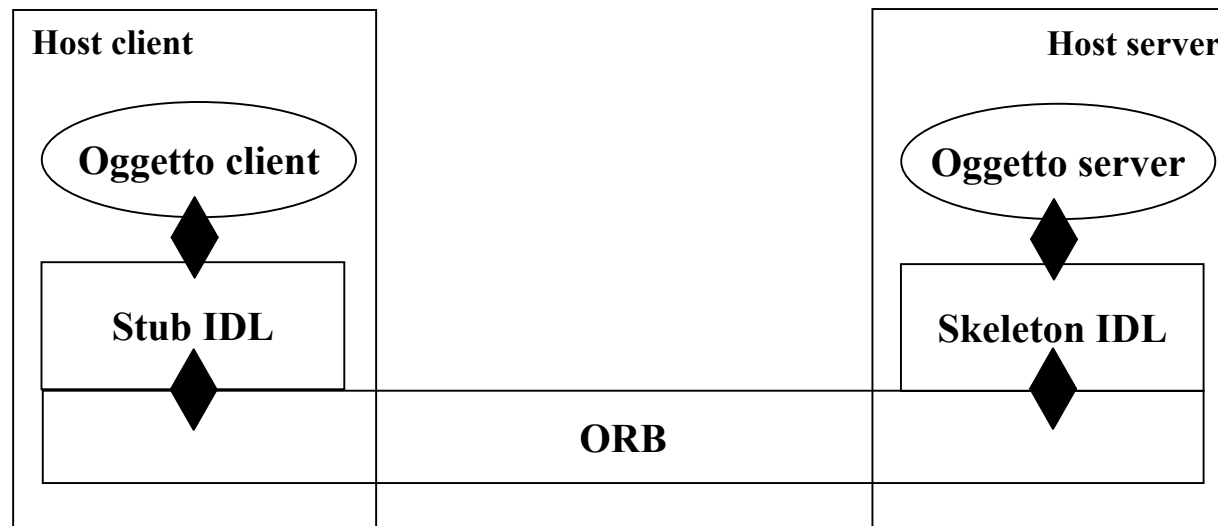
## CORBA 1/2

---

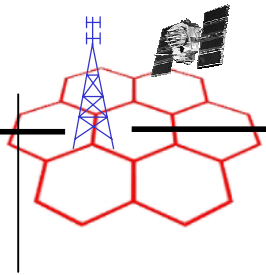
- ❑ CORBA offre un approccio **generale** ed **orientato agli oggetti** per lo sviluppo di applicazioni distribuite;
- ❑ Consente ad oggetti in esecuzione su di un host locale di richiamare metodi di oggetti in esecuzione su oggetti remoti;
- ❑ Diversamente da altri approcci **non è legato ad un particolare sistema operativo o linguaggio di programmazione**;
- ❑ CORBA utilizza oggetti accessibili attraverso ORB (Object Request Broker);
- ❑ Un ORB collega oggetti tra di loro sulla rete: un oggetto locale richiama metodi su oggetti remoti attraverso ORB;
- ❑ L'interfaccia del **client** verso l'**ORB** è uno **stub** scritto in **IDL [Interface Definition Language]**;
  - uno **stub agisce come un proxy locale per un oggetto remoto**;
  - **IDL offre un meccanismo “language independent” utilizzabile per descrivere i metodi di un oggetto**;
- ❑ L'interfaccia **dell'ORB** verso il **server** passa anch'essa per una struttura IDL (**skeleton**); **questo offre all'ORB un meccanismo “language independent” per accedere all'oggetto remoto**;
- ❑ Chiamata di un metodo remoto con CORBA:
  - Oggetto client chiama il metodo dell'oggetto remoto utilizzando lo stub IDL;
  - Lo stub IDL comunica la chiamata del metodo all'ORB e questi richiama il metodo sullo skeleton IDL;
  - Lo skeleton richiama il metodo sull'oggetto remoto e propaga a ritroso l'output fino all'oggetto locale;



## CORBA 2/2



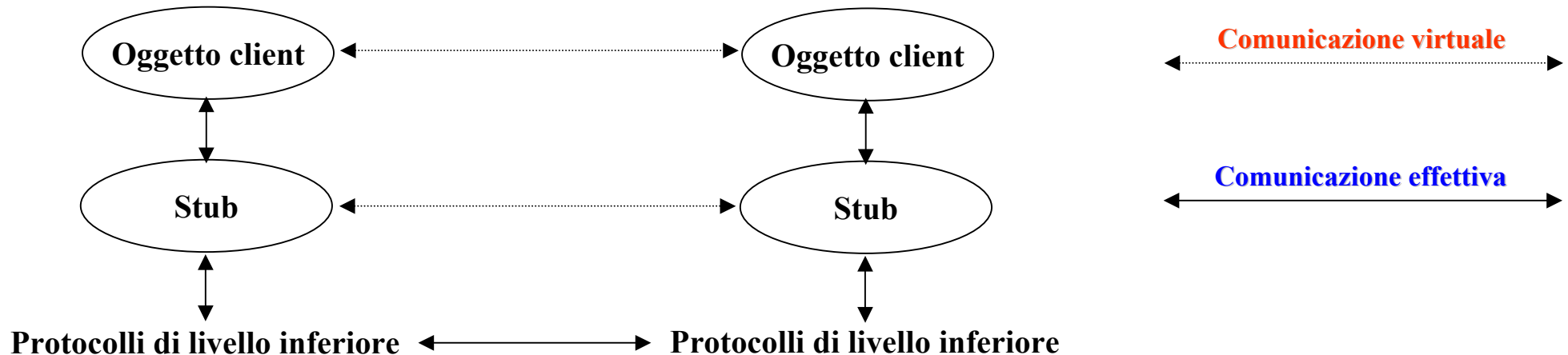
- ❑ Invece di creare server e client monolitici (come nel caso di browser e server web), CORBA consente di distribuire sia i client che i server su diversi host;
- ❑ L'indipendenza dal linguaggio ed il pieno supporto della programmazione ad oggetti hanno fatto in modo che CORBA sia riconosciuto come standard internazionale e venga supportato in pratica da tutti i produttori SW;

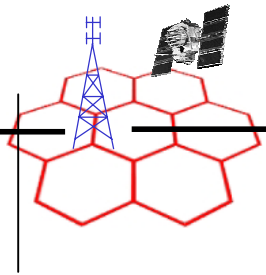


# JAVA RMI 1/4

## □ Il modello a oggetti distribuiti di Java:

- Permette ad oggetti in esecuzione su una JVM di richiamare metodi di oggetti in esecuzione su altre JVM;
- Le JVM possono essere in esecuzione come un processo separato sullo stesso computer o su altri computer remoti;
- **L'oggetto client** effettua la chiamata remota dei metodi **dell'oggetto server**;
- Un oggetto client non fa mai riferimento diretto all'oggetto remoto ma si riferisce ad **un'interfaccia remota implementata dall'oggetto remoto**;
- È una soluzione JAVA 100%: **efficienza** vs **dipendenza dal linguaggio**;



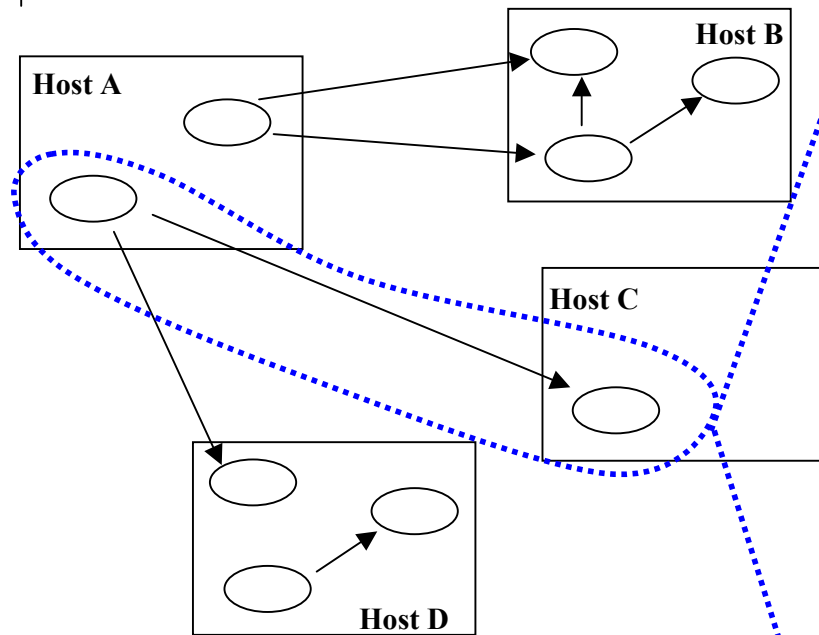
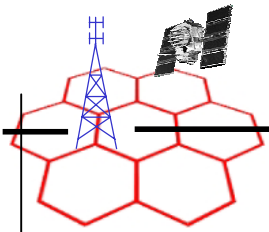


## JAVA RMI 2/4

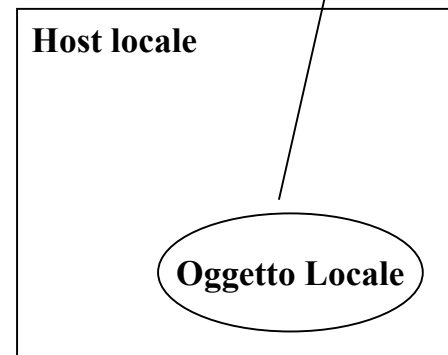
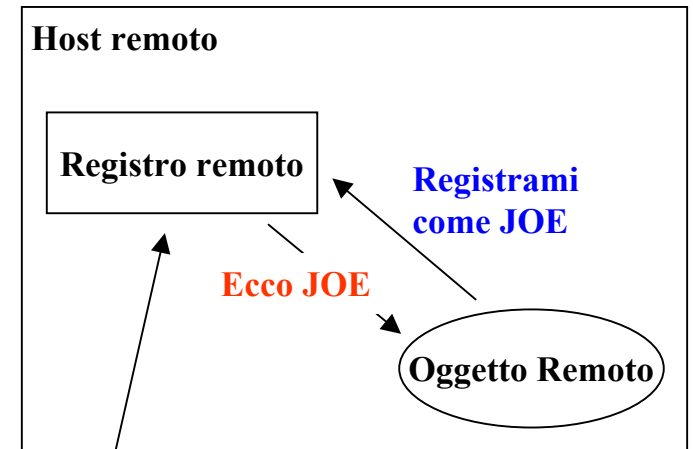
---

- Elementi base di un applicazione che utilizza RMI [**package java.rmi;**]:
  1. Creazione e compilazione **dell'interfaccia** dell'oggetto remoto; l'interfaccia estende la classe **Remote** e **definisce** (non implementa) i metodi richiamabili da remoto. Tali metodi devono sollevare una *RemoteException*;
  2. Creazione e compilazione della classe che **implementa** l'interfaccia dell'oggetto remoto [PUNTO 1] ed estende la classe **UnicastRemoteObject**;
  3. Creazione dello **stub** e dello **skeleton** per l'oggetto remoto [**rmi compiler**];
  4. La classe dell'oggetto remoto, la sua interfaccia e la classe struttura devono essere nel CLASSPATH dell'host remoto;
  5. Attivazione il demone del **registro RMI** sull'host remoto [**rmiregistry**];
  6. Un'istanza dell'oggetto remoto deve essere creata e registrata nel registro remoto [metodo **rebind()** della classe **Naming**];
  7. Copia dell'interfaccia **compilata** e dello **stub** sull'host client;
  8. Creazione e compilazione dell'oggetto locale [client];
  9. Esecuzione dell'oggetto locale che utilizza l'oggetto remoto invocandone i metodi;

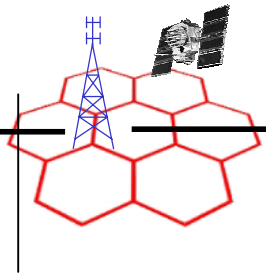
# JAVA RMI 3/4



○ Oggetto  
→ Chiamata di metodo



Lasciami accedere a JOE



## JAVA RMI 4/4

---

- Il riferimento ad oggetti remoti avviene mediante URL RMI la cui struttura è la seguente:
  - **//host:port/nomeOggetto;**  
[default host: localhost; default port: 1099]
  
  - **Registrazione dell'oggetto nel registro RMI**  
Naming.rebind("//"+hostname+"/Server",instance)
  
  - **Richiamare l'oggetto dal registro remoto**  
Server instance=(Server) Naming.lookup("//"+hostname+"/Server");