Comunicazione in Distribuito

SCD

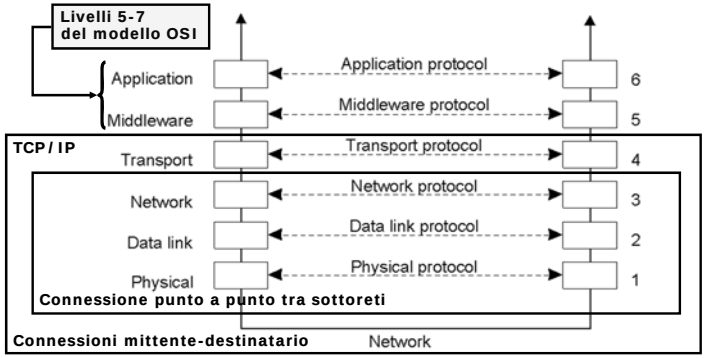
Anno accademico 2015/16
Sistemi Concorrenti e Distribuiti

Tullio Vardanega, tullio.vardanega@math.unipd.it

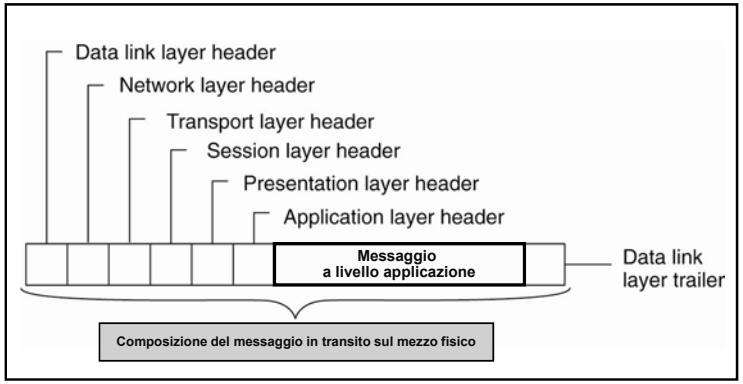
Evoluzione di modelli

- ***Remote Procedure Call*** **(RPC)**
  - Trasparente rispetto allo scambio messaggi necessario per supportare l'interazione cliente-servente a livello applicazione
- ***Remote (Object) Method Invocation*** **(RMI)**
  - Interazione a livello applicazione attraverso oggetti distribuiti
- **Scambio messaggi a livello *middleware***
  - Con paradigmi espliciti a livello applicazione
- ***Stream*** **o comunicazioni a flusso continuo**
  - Flusso di dati che richiedono continuità temporale
  - Ma di queste non tratteremo

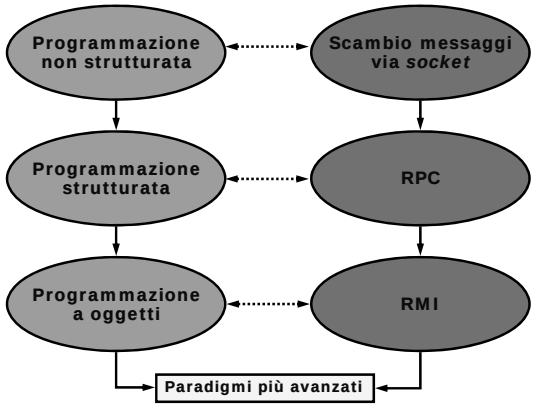
Visione a livelli – 1



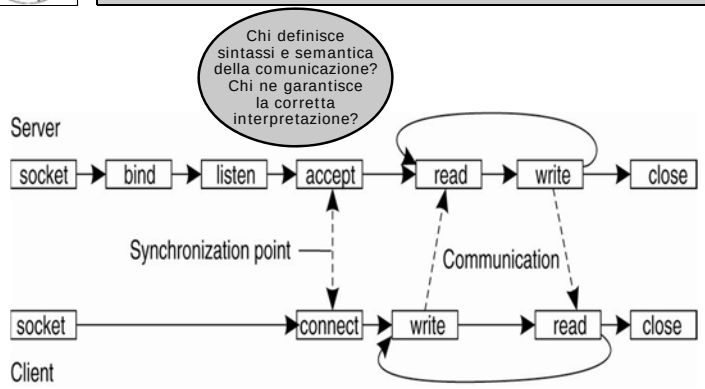
Visione a livelli – 2



Visione per analogie



Scambio messaggi



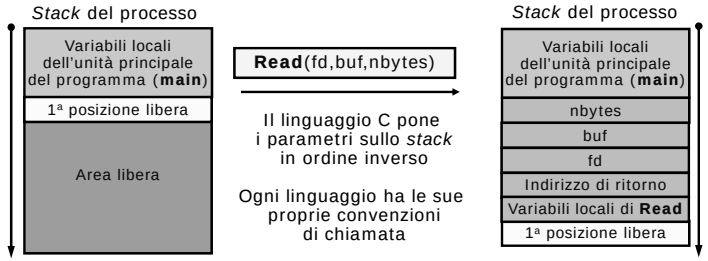
Tratto da: Tanenbaum & Van Steen, *Distributed Systems: Principles and Paradigms*, 2e, (c) 2007 Prentice-Hall, Inc.

RPC – 1

- **Consentire a un processo C residente su un nodo E1 di invocare ed eseguire una procedura P residente su un nodo E2**
- **Durante l'invocazione il chiamante viene sospeso**
  - **I parametri di ingresso viaggiano da chiamante a chiamato**
  - **I parametri di ritorno viaggiano da chiamato a chiamante**
- **Chiamante e chiamato <u>non</u> sono coinvolti nello scambio di messaggi sottostante**
  - **Trasparenza!**

RPC – 2

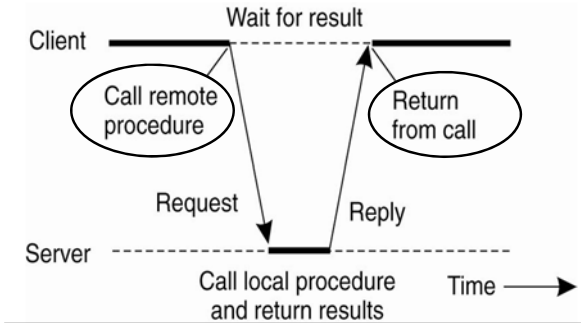
- **Chiamata di procedura locale**



RPC – 3

- **I parametri di procedura locale possono essere inviati per valore (*call-by-value*) o per riferimento (*call-by-reference*)**
  - **Un parametro inviato per valore viene semplicemente copiato sullo *stack* del chiamato**
    - Le modifiche apportate dal chiamato non hanno effetto sul chiamante
  - **Un parametro passato per riferimento permette accesso (via puntatore) allo spazio di memoria del chiamante**
    - Le modifiche apportate dal chiamante hanno effetto sul chiamato
  - **La variante *call-by-value-return* produce effetto sul chiamante solo al ritorno**

RPC – 4



Tratto da: Tanenbaum & Van Steen, *Distributed Systems: Principles and Paradigms*, 2e, (c) 2007 Prentice-Hall, Inc.

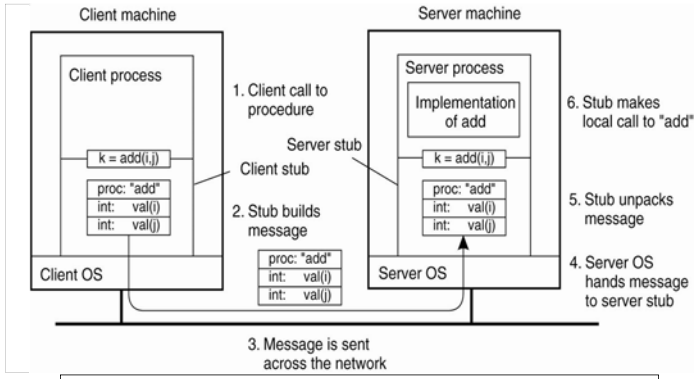
RPC – 5

- **Le procedure remote nello spazio del chiamante sono descritte da una procedura fittizia detta *client stub* invocabile con le convenzioni locali**
  - **Questa procedura svolge le azioni necessarie per effettuare la chiamata remota e riceverne il ritorno**
    - Inoltro chiamata e attesa ritorno
  - **Tali azioni avvengono tramite scambio messaggi in modo trasparente all'applicazione**

RPC – 6

- **L'arrivo del messaggio nello spazio del chiamato attiva una procedura fittizia detta *server stub***
  - **Questa procedura trasforma il messaggio in chiamata locale alla procedura invocata, ne raccoglie l'esito e lo inoltra al chiamante come messaggio**
- **In questo modo il chiamante e il chiamato hanno reciproca transparenza di locazione**

- **Il *client stub* trasforma la chiamata in una sequenza di messaggi da inviare sulla rete**
  - ***Parameter marshaling***
    - Relativamente agevole con parametri passati per valore
    - Occorre solo assicurare trasparenza di accesso
      - Rappresentazione dei valori secondo le convenzioni di chiamante e chiamato
    - Molto più difficile con parametri passati per riferimento
- **Il *server stub* esegue una trasformazione analoga e opposta**
  - ***Parameter un-marshaling***



Tratto da: Tanenbaum & Van Steen, *Distributed Systems: Principles and Paradigms*, 2e, (c) 2007 Prentice-Hall, Inc.

- **Tre aspetti chiave caratterizzano lo specifico protocollo RPC**
  - **Il formato dei messaggi scambiati tra gli *stub***
  - **La rappresentazione dei dati attesa da chiamante e chiamato**
    - *Encoding*
  - **La modalità di comunicazione su rete**
    - P.es.: TCP, UDP, …

- **Un servente si rende noto ai suoi clienti tramite registrazione del suo nodo di residenza presso un'anagrafe pubblica**
- **Il cliente <u>prima</u> localizza il nodo di residenza del servente e <u>poi</u> il processo servente (la sua porta)**
  - ***Binding***
  - **In ascolto sulla porta del servente può trovarsi un *daemon***

RPC – 11

- **RPC di base è sincrona**
  - **Ma la variante senza parametri *out* può essere asincrona**
- **L'eventualità di errori trasmissivi produce una tassonomia di semantiche**
  - ***At-least-once***
  - ***At-most-once***
  - ***Exactly-once***
- **Determinata dal protocollo *request-reply* in uso tra gli *stub***

Semantica della comunicazione – 1

- **Il protocollo di *request-reply* alla base di RPC combina 3 meccanismi aggiuntivi che si basano sull'attesa di conferma**
  - **Lato cliente: *Request Retry – RR1***
    - Il cliente prova fino a ottenere risposta o certezza del guasto del destinatario
  - **Lato servente: *Duplicate Filter – DF***
    - Il servente scarta gli eventuali duplicati provenienti dallo stesso cliente
  - **Lato servente: *Result Retransmit – RR2***
    - Il servente conserva le risposte per poterle ritrasmettere senza ricalcolarle
    - Fondamentale per calcolo non idempotente (**!**)

Semantica della comunicazione – 2

- **Semantica *best effort***
  - **Nessun meccanismo in uso**
  - **Il cliente non può sapere quante volte la sua richiesta sia stata eseguita**
- **Semantica *at least once***
  - **Il lato cliente usa RR1, il lato servente niente**
  - **Se la risposta arriva il cliente non sa quante volte sia stata calcolata dal servente**

Semantica della comunicazione – 3

- **Semantica *at most once***
  - **Tutti i meccanismi in uso**
  - **Se la risposta arriva il cliente sa che è stata calcolata una sola volta**
  - **La risposta non arriva solo a causa di guasti permanenti del servente**
- **Semantica *exactly once***
  - **Ha bisogno di meccanismi supplementari per tollerare guasti di lato servente**
    - P.es. replicazione trasparente

RMI – 1

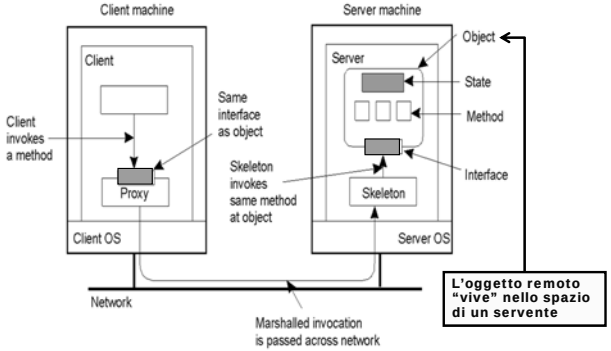
- **Il paradigma RPC può essere facilmente esteso al modello a oggetti distribuiti**
- **Soluzioni storiche**
  - **CORBA (Common Object Request Broker Architecture)**
    - OMG
  - **DCOM (Distributed Component Object Model) poi .NET**
    - Microsoft
  - **J2EE (Java 2 Platform Enterprise Edition) poi Enterprise Java Beans**
    - Sun Microsystems ora Oracle

RMI – 2

- **La separazione logica tra interfaccia e oggetto facilita la distribuzione**
  - **L'interfaccia di un oggetto può essere distribuita senza che lo sia il suo stato interno**
  - **Al *binding* di un cliente con un oggetto distribuito, una copia dell'interfaccia del servente (*proxy*) viene caricata nello spazio del cliente**
    - Il ruolo del *proxy* è analogo a quello del *client stub* in ambiente RPC
  - **La richiesta in arrivo all'oggetto remoto viene trattata da un "agente" (*skeleton*) del cliente localmente al servente**
    - Il ruolo dello *skeleton* è analogo a quello del *server stub* in ambiente RPC

RMI – 3

Realizzazione di oggetti distribuiti



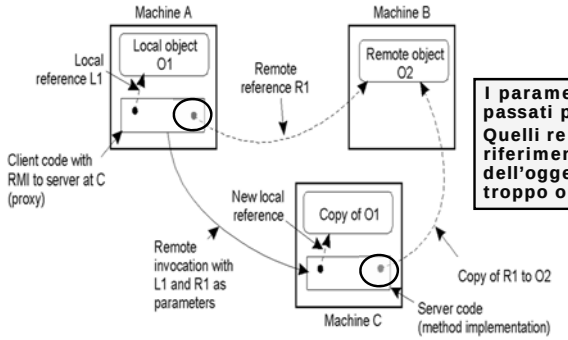
RMI – 4

- **Vi sono oggetti di tipo *compile-time***
  - **La cui realizzazione è completamente determinata dal linguaggio di programmazione**
    - Ambiente e protocollo d'uso noti e uniformi ma non *inter-operable*
- **E oggetti di tipo *run-time***
  - **Quando ciò che si vuole far apparire come oggetto non lo è necessariamente nella sua concreta realizzazione**
    - L'entità concreta (più spesso solo la sua interfaccia) viene incapsulata in un ***object wrapper*** che appare all'esterno come un normale oggetto distribuito
    - In questo modo si ottiene *inter-operability*

- **Vi sono oggetti persistenti**
  - **Che continuano a esistere anche al di fuori dello spazio di indirizzamento del processo servente**
    - Lo stato persistente dell'oggetto distribuito viene salvato in memoria secondaria e da lì ripristinato dai processi servente delegati a farlo
- **E oggetti transitori**
  - **Che cessano di esistere insieme al processo servente che li contiene**
- **Modelli RMI diversi fanno scelte diverse**

- **RMI offre maggiore trasparenza di RPC**
  - **I riferimenti a oggetti distribuiti hanno *scope* globale possono essere liberamente scambiati a livello sistema**
  - **Un riferimento poco scalabile usa un analogo del *daemon* RPC per interconnettere cliente e servente dell'oggetto**
    - <indirizzo di rete del *daemon*; identificatore del servente>
- **Modalità di riferimento**
  - ***Explicit binding***
    - Il cliente deve passare attraverso un registro che restituisce un puntatore al ***proxy*** dell'oggetto servente (**Java RMI**)
  - ***Implicit binding***
    - Il linguaggio risolve direttamente il riferimento (**C++** `Distr_object`)

- **L'invocazione remota può essere statica**
  - **Quando è nota al compilatore che predispone l'invocazione del *proxy* dal lato cliente**
    - L'interfaccia del servizio deve essere noto al programmatore del cliente
    - Se cambia l'interfaccia deve cambiare anche il cliente (nuova compilazione)
- **Oppure dinamica**
  - **Quando viene costruita a tempo d'esecuzione**
    - Sia l'oggetto distribuito che il metodo desiderato sono parametri assegnati dal programma (ignoti al compilatore)
    - Cambiamenti nell'interfaccia non hanno impatto sul codice del cliente



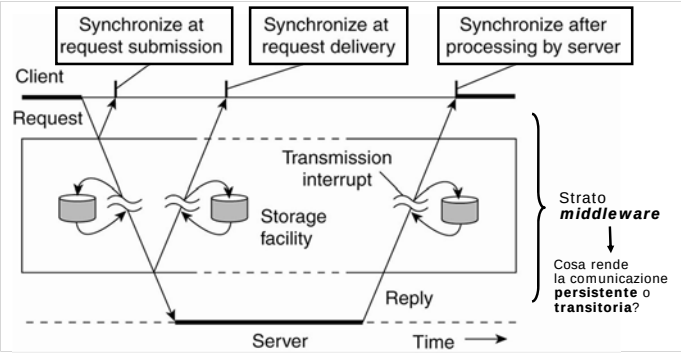
I parametri locali vengono passati per valore
Quelli remoti per riferimento: la copia dell'oggetto può essere troppo onerosa!

Scambio messaggi – 1

- **Comunicazione persistente**
  - **Il messaggio del mittente viene trattenuto dal MW fino alla consegna al destinatario**
- **Comunicazione transitoria**
  - **Non garantisce consegna del messaggio al destinatario perché è fragile rispetto ai possibili guasti (temporanei o permanenti)**
    - Analogo al modello di servizio del protocollo UDP

- **Comunicazione asincrona**
  - **Il mittente attende solo fino alla presa in carico del messaggio da parte del MW**
- **Comunicazione sincrona**
  - **Il mittente attende fino alla ricezione del destinatario o del suo MW**

Scambio messaggi – 2

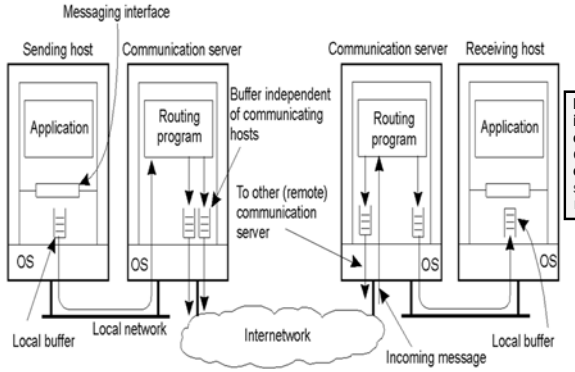


Tratto da: Tanenbaum & Van Steen, *Distributed Systems: Principles and Paradigms*, 2e, (c) 2007 Prentice-Hall, Inc.

Scambio messaggi – 3

- **Comunicazione persistente e asincrona**
  - **Come per la posta elettronica**
- **Comunicazione persistente e sincrona**
  - **Mittente bloccato fino alla ricezione garantita del destinatario**
- **Comunicazione transitoria e asincrona**
  - **Mittente non attende ma messaggio può andare perso** → UDP
- **Comunicazione transitoria e sincrona**
  1. **Mittente bloccato fino all'arrivo del messaggio nel MW del destinatario**
  2. **Mittente bloccato fino alla copia (<u>non garantita</u>) del messaggio nello spazio del destinatario** → RPC asincrona
  3. **Mittente bloccato fino alla ricezione di un messaggio di risposta dal destinatario** → RPC standard **e** RMI

Scambio messaggi – 4

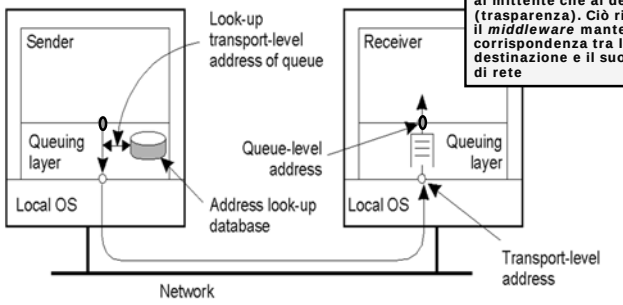


Lo scambio messaggi in ambiente distribuito comporta problemi di persistenza della comunicazione e di sincronizzazione tra mittente e destinatario

Scambio messaggi – 5

- ***Middleware* orientato a messaggi**
  - **Applicazioni distribuite comunicano tramite inserzione di messaggi in specifiche code → modello a code di messaggi**
    - Eccellente supporto a comunicazioni persistenti e asincrone
    - Nessuna garanzia che il destinatario prelevi il messaggio dalla sua coda
  - **Di immediata realizzazione tramite**
    - **`Put`** non bloccante (asincrona → come trattare il caso di coda piena?)
    - **`Get`** bloccante (sincrona rispetto alla presenza di messaggi in coda)
      - Un meccanismo di ***callback*** separa la coda dall'attivazione del destinatario
    - Una risorsa protetta realizza la coda con metodo **`Put`** di tipo **`P`** e metodo **`Get`** di tipo **`E`**
    - Realizzando coda *proxy* presso mittente e coda *skeleton* presso destinatario

Scambio messaggi – 6



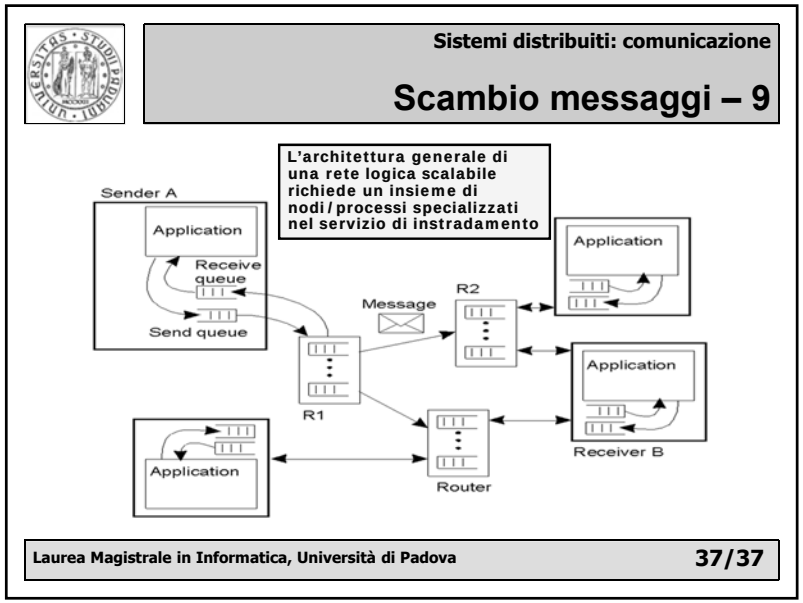
La coda appare <u>locale</u> sia al mittente che al destinatario (trasparenza). Ciò richiede che il *middleware* mantenga la corrispondenza tra la coda destinazione e il suo indirizzo di rete

Scambio messaggi – 7

- **Il *middleware* realizza una rete logica sovrapposta a quella fisica (*overlay network*) con topologia propria e distinta**
  - **Ciò richiede un proprio servizio di instradamento (*routing*)**
  - **Una sottorete connessa di instradatori conosce la topologia della rete logica e si occupa di far pervenire il messaggio del mittente alla coda del destinatario**
  - **Topologie complesse e variabili (scalabili) richiedono gestione dinamica delle corrispondenze coda-indirizzo di rete, in totale analogia con quanto avviene nel modello IP**

Scambio messaggi – 8

- **Un *broker* fornisce trasparenza di accesso a messaggi il cui formato aderisce a standard di trasporto diversi nel suo percorso**
  - **Servizio analogo a quello offerto dai *gateway* delle reti**
- **La natura del *middleware* è di essere adattivo e non intrusivo rispetto all'ambiente ospite**


Sistemi distribuiti: comunicazione
Scambio messaggi – 9
L'architettura generale di una rete logica scalabile richiede un insieme di nodi/processi specializzati nel servizio di instradamento
Sender A
Application
Receive queue
Send queue
Message
R2
Application
Application
R1
Application
Receiver B
Router
Laurea Magistrale in Informatica, Università di Padova
37/37