Comunicazione

SCD

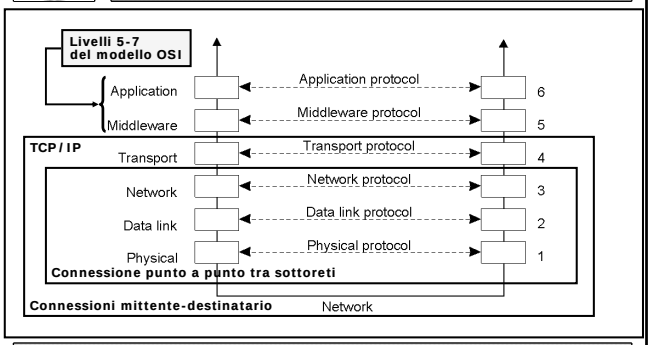
Anno accademico 2010/11
Sistemi Concorrenti e Distribuiti

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

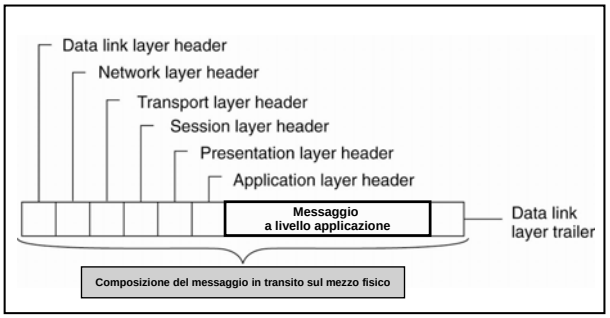
Evoluzione di modelli

- ***Remote Procedure Call*** **(RPC)**
  - Trasparente rispetto allo scambio messaggi necessario per supportare l'interazione cliente-servente a livello applicazione
- ***Remote (Object) Method Invocation*** **(RMI)**
  - L'interazione a livello applicazione avviene 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

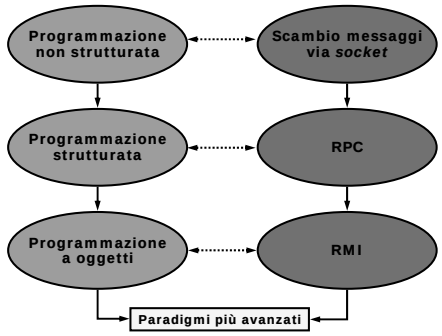
Visione a livelli – 1



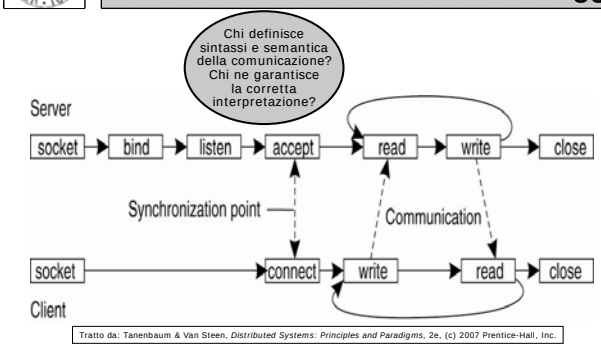
Visione a livelli – 2



Visione per analogie



Scambio messaggi



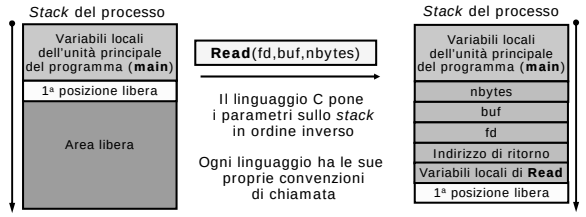
Tratto da: Tanenbaum & Van Steen, *Distributed Systems: Principles and Paradigms*, 2e, (c) 2007 Prentice-Hall, Inc.

RPC – 1

- Consentire a un processo C residente su un elaboratore (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 non sono coinvolti nello scambio di messaggi sottostante
  - Trasparenza!

RPC – 2

- Chiamata di procedura locale

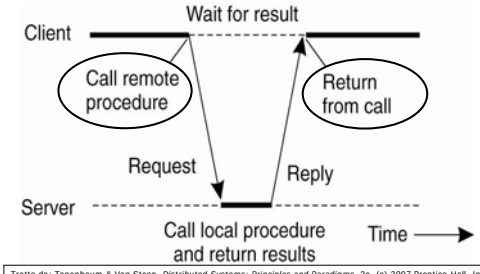


RPC – 3

- I parametri di procedura locale possono essere inviati per valore (*call-by-copy*) 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 fornisce un accesso (puntatore) a una variabile nello spazio del chiamante
    - Le modifiche apportate dal chiamante hanno effetto sul chiamato
  - La variante *call-by-value-return* produce questo effetto solo al ritorno

RPC – 4

- Chiamata di procedura remota



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

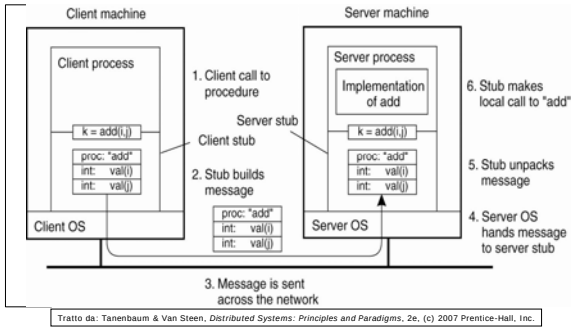
RPC – 5

- Trasparenza di locazione di chiamante e chiamato
  - Le procedure invocabili in remoto nello spazio del chiamante sono descritte da una procedura fittizia detta *client stub* invocabile con le convenzioni locali
    - Questa procedura svolge le azioni necessarie (invio chiamata e acquisizione ritorno) per effettuare la chiamata remota e riceverne il ritorno
    - Tali azioni avvengono tramite scambio messaggi trasparente all'applicazione
  - 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 invia al chiamante come messaggio

RPC – 6

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

RPC – 7



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

RPC – 8

- **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, ...

RPC – 9

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

RPC – 10

- **RPC di base è sincrona**
- **Ammette però variante asincrona**
- **Può avere semantica diversa in presenza di errori di rete**
  - ***At-least-once***
  - ***At-most-once***
  - ***Exactly-once***
- **Determinata dal protocollo di richiesta e risposta in uso tra gli *stub***

Semantica della comunicazione – 1

- **Il protocollo di *request-reply* usa una combinazione di 3 meccanismi di base**
  - **Lato cliente: *Riprova (Request Retry) – RR1***
    - Il cliente prova fino a ottenere risposta o certezza del guasto del destinatario
  - **Lato servente: *Filtra i duplicati (Duplicate filtering) – DF***
    - Il servente scarta gli eventuali duplicati provenienti dallo stesso cliente
  - **Lato servente: *Ritrasmetti le risposte (Result Retransmit) – RR2***
    - Il servente conserva le risposte per poterle ritrasmettere senza ricalcolarle
    - Fondamentale per calcolo non idempotente (**!**)

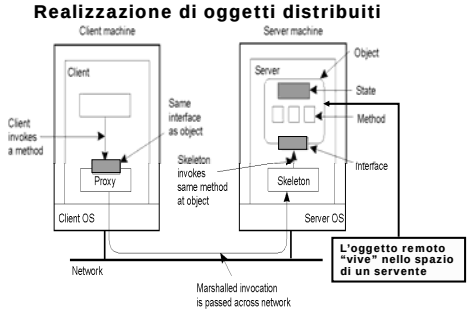
Semantica della comunicazione – 2

- **Semantica "*maybe*"**
  - **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 "*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**

- **Il paradigma RPC può essere facilmente esteso al modello a oggetti distribuiti**
- **Soluzioni storiche – standard di riferimento**
  - **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

- **La separazione logica tra interfaccia e oggetto facilita la distribuzione**
  - **L'interfaccia di un oggetto può essere distribuito 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**
    - Ruolo del tutto 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**
    - Ruolo del tutto analogo a quello del ***server stub*** in ambiente RPC

Realizzazione di oggetti distribuiti



- **Oggetti di tipo *compile-time***
  - **Realizzazione completamente determinata dal linguaggio di programmazione**
    - Ambiente e protocollo d'uso noti e uniformi
- **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

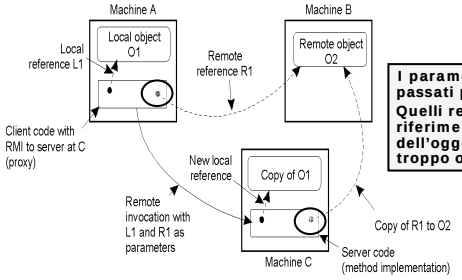
- **Oggetti persistenti**
  - **Continuano a esistere anche al di fuori dello spazio di indirizzamento del processo servente**
    - Lo stato persistente dell'oggetto distribuito viene salvato in memoria secondaria e da lì ripristinato dai processi servente delegati a farlo
- **Oggetti transitori**
  - **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`**)

RMI – 7

- **Invocazione statica**
  - **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)
- **Invocazione dinamica**
  - **Deve essere 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

RMI – 8



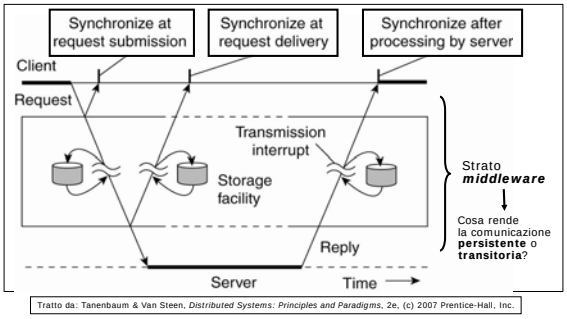
I parametri locali vengono passati per valore
Quelli remoti per riferimento: la copia dell'oggetto può essere troppo onerosa!

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

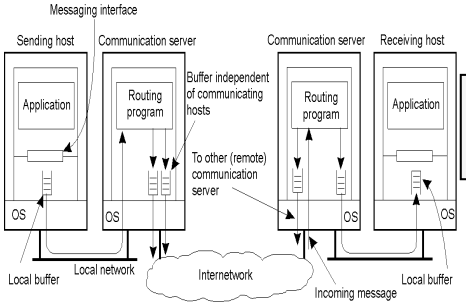


Tratto da: Tanenbaum & Van Steen, *Distributed Systems: Principles and Paradigms*, 2e, (c) 2007 Prentice-Hall, Inc.

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 (non garantita) 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

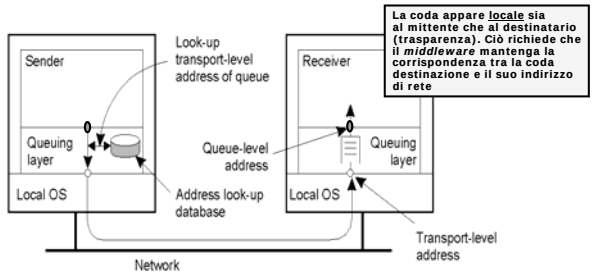


Lo scambio messaggi in ambiente distribuito comporta problemi di persistenza della comunicazione e di sincronizzazione tra mittente e destinatario

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

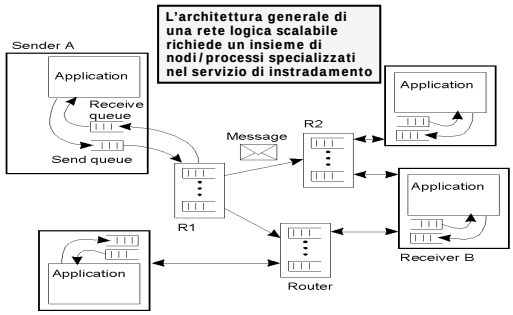
Scambio messaggi – 6



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
- **Un adattatore (*broker*) fornisce trasparenza di accesso a messaggi il cui formato aderisce a standard di trasporto diversi nel suo percorso**
  - **La natura del *middleware* è adattativa e non intrusiva**

Scambio messaggi – 8



Comunicazioni a flusso continuo – 1

- **Il contenuto delle comunicazioni scambiate tramite RPC, RMI e messaggi non dipende dal tempo di ricezione**
  - **Mezzo trasmissivo detto "a rappresentazione discreta"**
    - Adatto a testo, fotografie digitali, *file* oggetto, eseguibili
- **Vi sono comunicazioni il cui contenuto presenta dipendenze temporali forti**
  - **Mezzo trasmissivo detto "a rappresentazione continua"**
    - Adatto a video, audio → ***data stream*** : una sequenza di unità dati collegate
  - **Requisiti temporali espressi come "Qualità del Servizio" (QoS)**

Comunicazioni a flusso continuo – 2

- **L'invio di *data stream* è facilitato dal mezzo trasmissivo a rappresentazione continua, ma non ne è dipendente**
  - **Le *pipe* di UNIX e le connessioni di TCP/IP forniscono un mezzo trasmissivo a rappresentazione discreta (orientato a [gruppi di] *byte*)**
- **Vincoli temporali sulla modalità trasmissiva**
  - **Asincrona**
    - Preserva l'ordinamento, non la distanza temporale tra unità dati
  - **Sincrona**
    - Preserva l'ordinamento e garantisce un tempo massimo di trasmissione di ogni unità dati
  - **Isocrona**
    - Aggiunge la garanzia di un tempo minimo di trasmissione → **bounded (delay) jitter**

Comunicazioni a flusso continuo – 3

- Gli *stream* possono essere compositi (internamente strutturati) con requisiti temporali tra le parti che li compongono
  - Problema di sincronizzazione
- *Stream* come connessione virtuale tra sorgente e destinazione
  - *Multicast* : più destinatari di uno stesso *stream*
  - La connessione deve essere configurata in termini di risorse fisiche e logiche dedicate (QoS)

Comunicazioni a flusso continuo – 4

- QoS espressa come specifica di flusso
  - Capacità di trasporto (*bandwidth*), frequenza di trasmissione, ritardi di trasmissione, ecc.
- Il *Token Bucket Algorithm* fissa il contributo dello *stream* al traffico di rete
  - Un controllore produce gettoni che "comprano" un diritto d'uscita
  - Ogni unità dati in ingresso viene inviata sulla rete se può consumare un gettone
  - Altrimenti resta in coda (o viene scartata a coda piena)
  - Le unità dati vengono emesse sulla rete con la frequenza di generazione del gettone
    - Indipendentemente da possibili irregolarità di lato mittente

Comunicazioni a flusso continuo – 5

- Un esempio di sincronizzazione di *stream*
  - MPEG (*Motion Picture Expert Group*)
    - Un insieme di algoritmi standard per la compressione di video e audio
    - Per combinare in un singolo *stream* un insieme illimitato di *stream* distinti sia discreti che continui
      - Ciascuno *stream* originario viene trasformato in un flusso di unità dati (*frame*) la cui sequenza è determinata da un'etichetta temporale generata da un orologio unico con caratteristiche fissate (90 MHz)
      - I pacchetti di ciascuno *stream* vengono combinati mediante *multiplexing* in una sequenza composita di pacchetti a lunghezza variabile ma con propria etichetta temporale
      - A destinazione si ricompongono gli *stream* originari usando l'etichetta temporale per riprodurre la sincronizzazione tra ciascuna unità dati al suo interno