Comunicazione

SCD

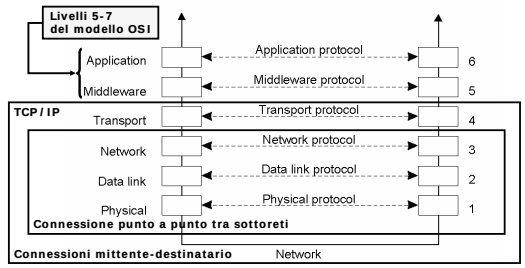
Anno accademico 2007/8
Corso di Sistemi Concorrenti e Distribuiti

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

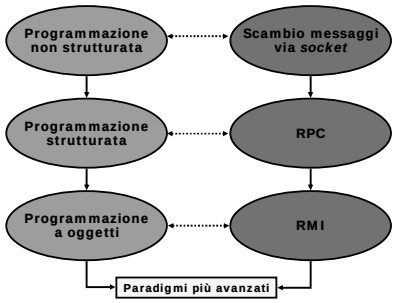
Evoluzione di modelli

- ***Remote Procedure Call*** **(RPC)**
  - Modello cliente-servente trasparente rispetto allo scambio messaggi
- ***Remote (Object) Method Invocation*** **(RMI)**
  - Modello cliente-servente basato su oggetti distribuiti
- **Scambio messaggi a livello *middleware***
  - Modelli più avanzati e protocolli più potenti e specializzati di quelli di rete
- ***Stream*** **o comunicazioni a flusso continuo**
  - Flusso di dati che richiedono continuità temporale

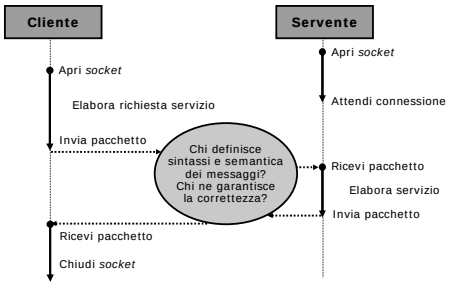
Visione a livelli



Visione per analogie



Scambio messaggi

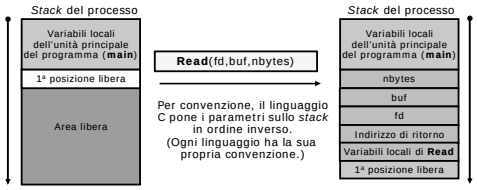


RPC – 1

- **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!

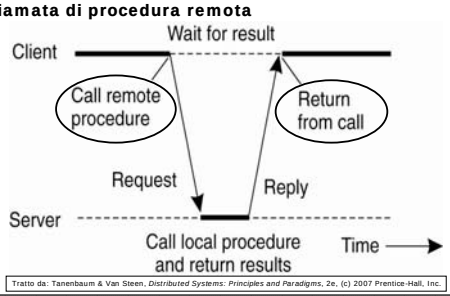
RPC – 2

- **Chiamata di procedura "normale"**



RPC – 3

Chiamata di procedura remota



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

RPC – 4

- **I parametri di procedura "normale" 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 (un puntatore) a una variabile nello spazio del chiamante**
    - Le modifiche apportate dal chiamante hanno effetto sul chiamato

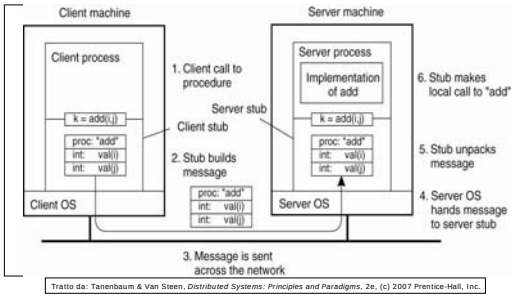
RPC – 5

- **Trasparenza rispetto alla non località di chiamante e chiamato**
  - **Le procedure invocabili in remoto sono descritte, nello spazio del chiamante, da una procedura fittizia detta *client stub* invocabile con le normali convenzioni**
  - **Questa racchiude tutte le azioni necessarie per effettuare l'effettiva chiamata e a riceverne l'esito**
    - Invio della richiesta e ricezione dei valori di ritorno avvengono tramite scambio messaggi
  - **L'arrivo del messaggio nello spazio del chiamato attiva una procedura fittizia detta *server stub***
  - **Questa trasforma il messaggio in una chiamata locale alla procedura invocata, ne raccoglie l'esito e lo invia al chiamante come messaggio sulla rete**

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 corretta dei valori rispetto alle convenzioni del chiamante e del chiamato
    - Molto più arduo 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

- **Il formato dei messaggi scambiati sulla rete è un aspetto del protocollo di RPC**
- **Altri aspetti includono la rappresentazione dei dati (*encoding*) attesa da chiamante e chiamato, sia per i tipi elementari che per le strutture**
- **Ultimo aspetto da definire è la modalità di comunicazione su rete (p.es. TCP, UDP)**

RPC – 9

- **Un servente si rende noto ai suoi possibili clienti tramite registrazione del suo nodo di residenza presso un'anagrafe pubblica**
- **Un cliente deve prima localizzare il nodo di residenza del servente e poi il processo servente (normalmente una porta)**
  - ***Binding***
  - **In ascolto sulla porta, al posto del servente, può trovarsi un *daemon***
- **RPC ha natura sincrona ma può essere reso in forma asincrona e con varie semantiche per il trattamento di errori**
  - ***At-least-once, at-most-once*, *exactly-once***
  - **In relazione al protocollo di gestione delle richieste e delle risposte**

Semantica della comunicazione – 1

- **Il protocollo di *request-reply* determina la semantica della comunicazione in caso di errori**
- **Combinando 3 meccanismi di base**
  - **Lato cliente: *Riprova (Request Retry – RR1)***
    - Il cliente continua a provare fino a ottenere risposta o la certezza del guasto del destinatario
  - **Lato servente: *Filtra i duplicati (Duplicate filtering – DF)***
    - Il servente scarta gli eventuali duplicati di richieste provenienti dallo stesso cliente
  - **Lato servente: *Ritrasmetti le risposte (Result Retransmit – RR2)***
    - Il servente conserva le risposte per poterle ritrasmettere senza ricalcolarle
      - Fondamentale ove il calcolo non fosse idempotente!

Semantica della comunicazione – 2

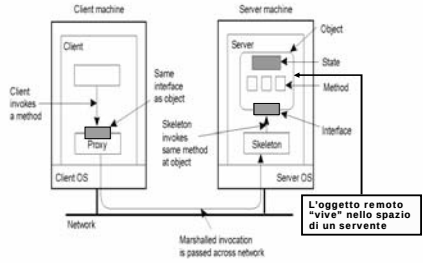
- **Semantica "*maybe*"**
  - **Nessun meccanismo in uso**
  - **Il cliente non può sapere quante volte la sua richiesta sia stata eseguita**
- **Semantica "*at least once*"**
  - **Il lato cliente usa RR1 ma il lato servente non usa né DF né RR2**
  - **Quando arriva una risposta il cliente non sa quante volte sia stata calcolata dal servente e dunque non conosce per certo lo stato del servente**
- **Semantica "*at most once*"**
  - **Tutti i meccanismi in uso**
  - **Se la risposta arriva il cliente sa che è stata calcolata una sola volta dal servente**
    - Non arriva soltanto in presenza di guasti permanenti del servente
- **Semantica "*exactly once*"**
  - **Ha bisogno di ulteriori meccanismi (p.es. replicazione trasparente) per tollerare guasti di lato servente**

RMI – 1

- **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*) → .NET – Microsoft
    - **J2EE** (*Java 2 Platform, Enterprise Edition*) → *Enterprise JavaBeans* – Sun Microsystems
- **La separazione logica tra interfaccia e oggetto consente anche separazione fisica**
  - **Lo stato interno di un oggetto non viene distribuito!**
  - **Al *binding* di un cliente con un oggetto servente 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" del cliente, locale al servente (*skeleton*)**
    - Ruolo del tutto analogo a quello del ***server stub*** in ambiente RPC

RMI – 2

Realizzazione di oggetti distribuiti



RMI – 3

- **Oggetti di tipo *compile-time***
  - **La loro realizzazione è completamente determinata dal linguaggio di programmazione**
    - Ambiente e protocollo d'uso noti e uniformi
- **Oggetti di tipo *run-time***
  - **Ciò che si vuole far apparire come oggetto senza che la sua concreta realizzazione lo sia effettivamente**
    - Occorre incapsulare l'entità concreta, spesso solo l'interfaccia, con un "adattatore" (***object wrapper***) capace di interagire con l'esterno come un normale oggetto distribuito

RMI – 4

- **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 diversi fanno scelte diverse**

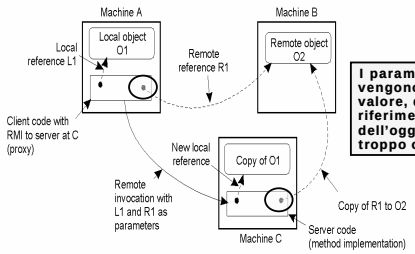
RMI – 5

- **Maggior trasparenza rispetto a RPC**
  - **I riferimenti a oggetti distribuiti solo validi e possono essere scambiati a livello sistema**
    - ***System-wide* (in RMI)** *vs.* ***scoped* (in RPC)**
- **2 modalità di utilizzo dei riferimenti**
  - ***Explicit binding***
    - Il cliente deve passare attraverso una fase di registrazione che gli restituisce un valore puntatore al ***proxy*** dell'oggetto servente (**Java RMI**)
  - ***Implicit binding***
    - Il linguaggio risolve direttamente il riferimento del cliente all'oggetto distribuito (**C++** `Distr_object`)
- **Un analogo del *daemon* di RPC agisce da mediatore tra il cliente e il nodo ove risiede il servente dell'oggetto richiesto**
  - Riferimento a oggetto distribuito (scarsamente scalabile): <indirizzo di rete del *daemon*; identificatore di livello sistema del servente>

RMI – 6

- **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 sulla struttura del cliente

RMI – 7

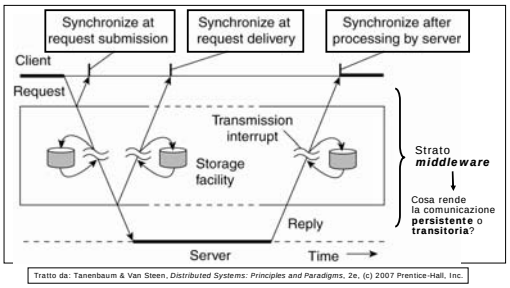


I parametri locali in RMI vengono passati per valore, quelli remoti per riferimento → la copia dell'oggetto può essere troppo onerosa!

Scambio messaggi – 1

- **Comunicazione persistente**
  - **Il messaggio inviato dal mittente viene trattenuto e preservato dalla infrastruttura di comunicazione fino alla sua consegna al destinatario**
- **Comunicazione transitoria**
  - **L'infrastruttura di comunicazione è fragile rispetto ai possibili guasti (temporanei o permanenti) e non garantisce la consegna del messaggio al destinatario**
    - Il modello di servizio offerto dal protocollo UDP di Internet
- **Comunicazione asincrona**
  - **Il mittente attende solo fino alla prima memorizzazione del messaggio**
- **Comunicazione sincrona**
  - **Il mittente attende fino alla ricezione del destinatario (o del suo nodo di residenza)**

Scambio messaggi – 2

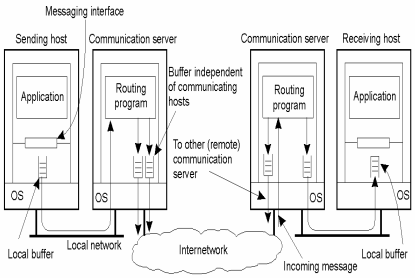


Tratto da: Tanenbaum & Van Steen, *Distributed Systems: Principles and Paradigms*, 2e, (c) 2007 Prentice-Hall, Inc.

Scambio messaggi – 3

- **6 possibili combinazioni reali**
  - **Comunicazione persistente e asincrona**
    - Esempio: posta elettronica
  - **Comunicazione persistente e sincrona**
    - Il mittente viene bloccato fino alla copia (garantita) del messaggio presso il destinatario
  - **Comunicazione transitoria e asincrona**
    - Il mittente prosegue senza attendere ma il messaggio viene perso se il destinatario non è attivo all'arrivo del messaggio → **UDP**
  - **Comunicazione transitoria e sincrona**
    - **1**: mittente bloccato fino alla copia del messaggio nel nodo 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

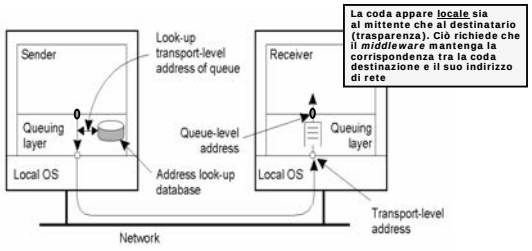


Lo scambio messaggi in ambiente distribuito comporta problemi di persistenza della comunicazione e di sincronizzazione tra mittente e destinatario

Scambio messaggi – 5

- ***Middleware* orientato a messaggi**
  - **Applicazioni distribuite comunicano tramite inserzione di messaggi in specifiche code → modello a code di messaggi**
    - Eccellente supporto alle 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)
      - Il meccanismo realizzativo che separa la coda dall'attivazione del destinatario quando la condizione di risveglio viene soddisfatta viene spesso detto ***callback***
    - Applicando il modello concorrente esaminato a lezione utilizzeremmo una risorsa protetta per rappresentare la coda, con canali tipati `Put` di tipo `P` e `Get` di tipo `E`
    - Applicando il concetto di *proxy* e *skeleton* di coda presso mittente e destinatario

Scambio messaggi – 6



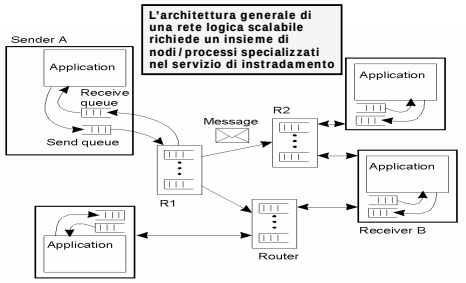
La coda appare locale sia al mittente che al destinatario (trasparenza). Ciò richiede che il *middleware* mantenga la corrispondenza tra la coda destinazione e il suo indirizzo di rete

Scambio messaggi – 7

- **Il *middleware* realizza una rete logica che si sovrappone a quella fisica (*overlay network*) ma ha 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 debba riflettere standard di trasporto diversi nel percorso**
  - **La natura del *middleware* è adattativa e non intrusiva**

Sistemi distribuiti: comunicazione

Scambio messaggi – 8



Sistemi distribuiti: comunicazione

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)**

Sistemi distribuiti: comunicazione

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**

Sistemi distribuiti: comunicazione

Comunicazioni a flusso continuo – 3

- **Le *stream* possono essere composite, ossia internamente strutturate, con requisiti temporali tra le parti**
  - **Problema di sincronizzazione**
- **Una *stream* può essere vista come una connessione virtuale tra sorgente e destinazione**
  - ***Multicast* : più destinatari di uno stesso *stream***
  - **La connessione deve dunque essere configurata in termini di risorse fisiche e logiche dedicate (QoS)**

Sistemi distribuiti: comunicazione

Comunicazioni a flusso continuo – 4

- **La QoS può essere espressa come una specifica di flusso**
  - **Capacità di trasporto (*bandwidth*), frequenze di trasmissione, ritardi di trasmissione, etc.**
- **Il *Token Bucket Algorithm* determina il contributo dello *stream* al traffico di rete**
  - **Un controllore produce gettoni con periodo T, che pagano il costo d'uscita; ogni unità dati in ingresso viene inviata sulla rete se vi è un gettone per lei, altrimenti resta in coda (o viene scartata se la coda fosse piena)**
    - L'effetto è che, a regime, le unità dati sono inviate sulla rete con la frequenza di generazione del gettone, indipendentemente da possibili irregolarità di generazione

Sistemi distribuiti: comunicazione

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 ed audio
    - Consente di 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 (pacchetti, *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