

Università degli Studi di Padova

Comunicazione

SCD

Anno accademico 2004/5
 Corso di Sistemi Concorrenti e Distribuiti

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

Corso di Laurea Specialistica in Informatica, Università di Padova 1/31

Università degli Studi di Padova

Sistemi distribuiti: comunicazione

Evoluzione di modelli

- **Remote Procedure Call (RPC)**
 - Modello cliente-server trasparente rispetto allo scambio messaggi
- **Remote (Object) Method Invocation (RMI)**
 - Modello cliente-server 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

Corso di Laurea Specialistica in Informatica, Università di Padova 2/31

Università degli Studi di Padova

Sistemi distribuiti: comunicazione

Visione a livelli

Corso di Laurea Specialistica in Informatica, Università di Padova 3/31

Università degli Studi di Padova

Sistemi distribuiti: comunicazione

Visione per analogie

Corso di Laurea Specialistica in Informatica, Università di Padova 4/31

Università degli Studi di Padova

Sistemi distribuiti: comunicazione

Scambio messaggi

Corso di Laurea Specialistica in Informatica, Università di Padova 5/31

Università degli Studi di Padova

Sistemi distribuiti: comunicazione

RPC - 1

- Consentire ad un processo su un elaboratore di l'esecuzione di una procedura su un altro elaboratore
- 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

Corso di Laurea Specialistica in Informatica, Università di Padova 6/31

Università degli Studi di Padova Sistemi distribuiti: comunicazione
RPC - 2

❑ **Chiamata di procedura "normale"**

Per convenzione, il linguaggio C pone i parametri sulla *stack* in ordine inverso. (Ogni linguaggio ha la sua propria convenzione.)

Corso di Laurea Specialistica in Informatica, Università di Padova 7/31

Università degli Studi di Padova Sistemi distribuiti: comunicazione
RPC - 3

❑ **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***
 - Le modifiche apportate dal chiamato non hanno effetto sul chiamante
- **Un parametro passato per riferimento fornisce un accesso (un puntatore) ad una variabile nello spazio del chiamante**
 - Le modifiche apportate dal chiamante hanno effetto sul chiamato

Corso di Laurea Specialistica in Informatica, Università di Padova 8/31

Università degli Studi di Padova Sistemi distribuiti: comunicazione
RPC - 4

❑ **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* ed invocabile con le normali convenzioni
- Questa racchiude tutte le azioni necessarie ad effettuare l'effettiva chiamata ed 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

Corso di Laurea Specialistica in Informatica, Università di Padova 9/31

Università degli Studi di Padova Sistemi distribuiti: comunicazione
RPC - 5

❑ **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 ed opposta**

- **Parameter unmarshalling**

Corso di Laurea Specialistica in Informatica, Università di Padova 10/31

Università degli Studi di Padova Sistemi distribuiti: comunicazione
RPC - 6

❑ **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)**

Corso di Laurea Specialistica in Informatica, Università di Padova 11/31

Università degli Studi di Padova Sistemi distribuiti: comunicazione
RPC - 7

❑ **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 resa in forma asincrona e con varie semantiche per il trattamento di errori**

- *At-least-once, at-most-once, exactly-once, etc.*
- In relazione al protocollo di gestione delle richieste e delle risposte

Corso di Laurea Specialistica in Informatica, Università di Padova 12/31

Università degli Studi di Padova **Sistemi distribuiti: comunicazione** **RMI - 1**

- ❑ Il principio base di RPC può essere esteso convenientemente al modello ad oggetti distribuiti
 - Due soluzioni standard di riferimento: CORBA e DCOM
- ❑ La separazione logica tra interfaccia ed 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 implementazione 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

Corso di Laurea Specialistica in Informatica, Università di Padova **13/31**

Università degli Studi di Padova **Sistemi distribuiti: comunicazione** **RMI - 2**

Implementazione di oggetti distribuiti

Corso di Laurea Specialistica in Informatica, Università di Padova **14/31**

Università degli Studi di Padova **Sistemi distribuiti: comunicazione** **RMI - 3**

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

Corso di Laurea Specialistica in Informatica, Università di Padova **15/31**

Università degli Studi di Padova **Sistemi distribuiti: comunicazione** **RMI - 4**

- ❑ **Oggetti persistenti**
 - Continuano ad 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
- ❑ **Oggetti transitori**
 - Cessa di esistere insieme al processo servente che lo contiene
- ❑ **Modelli diversi fanno scelte diverse**

Corso di Laurea Specialistica in Informatica, Università di Padova **16/31**

Università degli Studi di Padova **Sistemi distribuiti: comunicazione** **RMI - 5**

- ❑ **Maggior trasparenza rispetto ad RPC**
 - I riferimenti ad 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++ `Distrc_object`)
- ❑ **Un analogo del *daemon* di RPC agisce da mediatore tra il cliente ed il nodo ove risiede il servente dell'oggetto richiesto**
 - Riferimento ad oggetto distribuito (scarsamente scalabile): <indirizzo di rete del *daemon*>, identificatore di livello sistema del servente->

Corso di Laurea Specialistica in Informatica, Università di Padova **17/31**

Università degli Studi di Padova **Sistemi distribuiti: comunicazione** **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

Corso di Laurea Specialistica in Informatica, Università di Padova **18/31**

Università degli Studi di Padova

Sistemi distribuiti: comunicazione

RMI - 7

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

Client code with RMI to server at C (proxy)

Machine A: Local object O1, Local reference L1

Machine B: Remote object O2, Remote reference R1

Machine C: Copy of O1, Copy of R1 to O2, Server code (method implementation)

Remote invocation with L1 and R1 as parameters

New local reference

Remote reference R1

Copy of R1 to O2

Corso di Laurea Specialistica in Informatica, Università di Padova

19/31

Università degli Studi di Padova

Sistemi distribuiti: comunicazione

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)

Corso di Laurea Specialistica in Informatica, Università di Padova

20/31

Università degli Studi di Padova

Sistemi distribuiti: comunicazione

Scambio messaggi - 2

- **6 possibili combinazioni reali**
 - **Comunicazione persistente ed asincrona**
 - Esempio: posta elettronica
 - **Comunicazione persistente e sincrona**
 - Il mittente viene bloccato fino alla copia (**garantita**) del messaggio presso il destinatario
 - **Comunicazione transitoria ed 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 asincrono**
 - 3: mittente bloccato fino alla ricezione di un messaggio di risposta dal destinatario → **RPC standard ed RMI**

Corso di Laurea Specialistica in Informatica, Università di Padova

21/31

Università degli Studi di Padova

Sistemi distribuiti: comunicazione

Scambio messaggi - 3

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

Messaging interface

Sending host: Application, OS

Communication server: Routing program, OS

Receiving host: Application, OS

Local buffer, Local network, Internet network, Incoming message

Buffer independent of communicating hosts

To other (remote) communication server

Corso di Laurea Specialistica in Informatica, Università di Padova

22/31

Università degli Studi di Padova

Sistemi distribuiti: comunicazione

Scambio messaggi - 4

- **Middleware orientato a messaggi**
 - **Applicazioni distribuite comunicano tramite inserzione di messaggi in specifiche code → modello a code di messaggi**
 - Eccellente supporto alle comunicazioni persistenti ed 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 implementativo 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 utilizzeremo una risorsa protetta per rappresentare la coda, con metodo Put di tipo procedure e metodo Get di tipo entry
 - Applicando il concetto di *proxy* e *skeleton* di coda presso mittente e destinatario

Corso di Laurea Specialistica in Informatica, Università di Padova

23/31

Università degli Studi di Padova

Sistemi distribuiti: comunicazione

Scambio messaggi - 5

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

Sender: Queuing layer, Local OS

Receiver: Queuing layer, Local OS

Network

Look-up transport-level address of queue

Queue-level address

Address look-up database

Transport-level address

Corso di Laurea Specialistica in Informatica, Università di Padova

24/31

Università degli Studi di Padova

Sistemi distribuiti: comunicazione

Scambio messaggi - 6

Il **middleware** realizza una rete logica che sovrappone a quella fisica (**overlay network**) ma ha topologia propria

- Ciò richiede un 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

Corso di Laurea Specialistica in Informatica, Università di Padova 25/31

Università degli Studi di Padova

Sistemi distribuiti: comunicazione

Scambio messaggi - 7

L'architettura generale di una rete logica scalabile richiede un insieme di nodi/processi specializzati nel servizio di instradamento

Corso di Laurea Specialistica in Informatica, Università di Padova 26/31

Università degli Studi di Padova

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 ed 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)

Corso di Laurea Specialistica in Informatica, Università di Padova 27/31

Università degli Studi di Padova

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**

Corso di Laurea Specialistica in Informatica, Università di Padova 28/31

Università degli Studi di Padova

Sistemi distribuiti: comunicazione

Comunicazioni a flusso continuo - 3

Le **stream** possono essere composite, ossia internamente strutturate, con requisiti temporali tra le parti → **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)

Corso di Laurea Specialistica in Informatica, Università di Padova 29/31

Università degli Studi di Padova

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

Corso di Laurea Specialistica in Informatica, Università di Padova 30/31

Università degli Studi di Padova	Sistemi distribuiti: comunicazione Comunicazioni a flusso continuo - 5
<p>□ Un esempio di sincronizzazione di <i>stream</i></p> <ul style="list-style-type: none">○ MPEG (<i>Motion Picture Expert Group</i>)<ul style="list-style-type: none">● Un insieme di algoritmi standard per la compressione di video ed audio● Consente di combinare un insieme illimitato di <i>stream</i> distinti, sia discreti che continui in un singolo <i>stream</i><ul style="list-style-type: none">• Ciascuno <i>stream</i> originario viene trasformato in un flusso di unità dati (pacchetti) la cui sequenza è determinata da un'etichetta temporale generata da un orologio <i>video</i> con caratteristiche fissate (90 MHz)• I pacchetti di ciascuno <i>stream</i> vengono combinati mediante <i>multiplexing</i> in una sequenza composta di pacchetti a lunghezza variabile, ma con propria etichetta temporale• A destinazione si ricompongono gli <i>stream</i> originali usando l'etichetta temporale per riprodurre la sincronizzazione tra ciascuna unità dati al suo interno	
Corso di Laurea Specialistica in Informatica, Università di Padova	31/31