CORSO PAS
Laboratorio di RETI

Nicola Drago
nicola.drago@univr.it
Dipartimento di Informatica
Università di Verona

Le origini della comunicazione: Socket

In un primo tempo nasce in ambiente UNIX

- Negli anni ‘80 la Advanced Research Project Agency (ARPA) finanziò l’università di Berkeley per implementare la suite TCP/IP nel sistema operativo UNIX
- I ricercatori di Berkeley svilupparono il set originario di funzioni che fu chiamato *interfaccia socket*, creando quindi delle API aggiuntive
- I socket sono stati progettati per supportare vari protocolli di rete (Unix Domain, Xerox NS Domain) ma col tempo hanno predominato i protocolli dell’architettura Internet basati sull’Internet Protocol(IP): TCP e UDP.

Indirizzamento

- Ogni computer della rete è identificato da un *hostname* e da un indirizzo IP unico a 32 bit
  es: “www.javasoft.com” ⇗ 204.160.241.98
- L’indirizzo IP contiene sia l’identificativo della rete che quello della macchina
- Per verificare l’associazione fra nomi e indirizzi si puo’ utilizzare il comando nslookup che offre la lista di tutti i server alias per un indirizzo.
  Es. >nslookup www.javasoft.com

Come identificare i processi

- Si assegna ad ogni processo un numero di porta
- I numeri di porta possono essere
  - assegnati in genere a servizi di sistema (porte da 0-1023)
  - dinamici o privati (1024 - 65535)
- I server/daemon usano di solito porte ben note
  - es HTTP = 80 , Telnet = 25, FTP = 21
- I client di solito usano porte dinamiche
- Questa associazione viene usata dai protocolli a di livello transport come TCP o UDP

Modello client - server

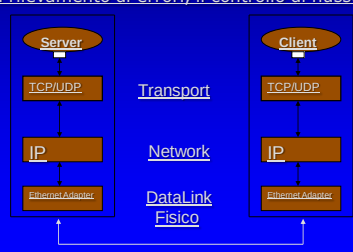
- Il server viene programmato per poter fornire un servizio a chi ne fa richiesta
- Il client inoltra richieste al server e ne elabora le risposte.
- Si deve adottare un meccanismo che consenta la comunicazione fra le due entità.

  - come indirizzare il messaggio in modo che arrivi al server destinatario?

  - come specificare il processo destinatario del messaggio?

Architettura di protocolli

I protocolli implementano quelle funzionalità che consentono lo scambio di messaggio sul mezzo fisico disposto fra due macchine, (ad esempio la suddivision in quadri, il rilevamento di errori, il controllo di flusso ecc.)



Concetto di Socket

- Per attivare una connessione fra due processi si utilizza dunque lo schema Host-Port
  - il server è eseguito su un host e aspetta richieste su una determinata porta
  - il client viene eseguito su un host e richiede l'uso di una porta per effettuare richieste ad un server
- Specificando ora anche il protocollo per la comunicazione riusciamo a descrivere univocamente la connessione con una quintupla detta Association:

(protocollo,ind locale, proc locale, ind remoto, proc remoto)
es: (TCP,123.23.4.221,1500,234.151.124.2,4000)

Concetto di Socket

- La quintupla viene in realtà costituita a partire da due elementi simmetrici, un'associazione locale ed una remota:
  - Protocollo, Ind. locale, porta locale (TCP,123.23.4.221,1500)
  - Protocollo, Ind. remoto, porta remota (TCP, 234.151.124.2,4000)
- Ciascuna si queste parti è detta **socket**
- Nota: protocolli differente possono usare la stessa porta;
  - 1040 per TCP ≠ 1040 UDP

Concetto di Socket

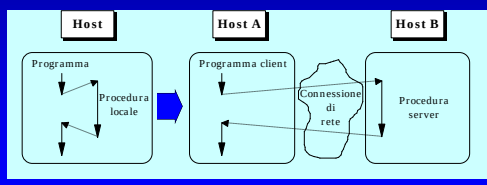
- Un socket è un descrittore di risorsa che consente ad una applicazione di effettuare operazioni di lettura / scrittura verso un particolare dispositivo di I/O.
- Astrazione di un canale di comunicazione tra processi distribuiti in rete
- Interfaccia per la suite di protocolli TCP/IP
- Punto di arrivo nella comunicazione in rete

Creazione di un socket

- Lato server
  - Creazione del socket
  - Bind ad una porta
  - Listen, predisposizione a ricevere sulla porta
  - Accept, blocca il server in attesa di una connesione
  - Lettura - scrittura dei dati
  - Chiusura
- Lato client
  - Creazione del socket
  - Richiesta di connessione
  - Lettura - scrittura dei dati
  - Chiusura

La comunicazione Sun RPC

- Possibilità di invocare una procedura non locale → operazione che interessa un nodo remoto e ne richiede un servizio
- RPC – Remote Procedure Call



Sun RPC Proprietà e Requisiti

- Eterogeneità: dati diversi su macchine diverse
  - semantica diversa del modello locale esteso in modo remoto
- Efficienza: implementazioni efficienti del modello
  - reti locali ad elevata velocità di trasferimento dati
- Autenticazione: controllo di accesso da parte di utenti autorizzati
  - Possibilità di invocare una procedura non locale → operazione che interessa un nodo remoto e ne richiede un servizio
- Il supporto scambia messaggi per consentire
  - identificazione dei messaggi di chiamata e risposta
  - identificazione unica della procedura remota
  - autenticazione del chiamante e del servitore
  - amministrazione della rete e gestione di alcuni tipi di errori dovuti alla distribuzione
    - implementazione errata
    - errori dell'utente
    - roll-over (ritorno indietro)

Sun RPC
Proprietà e Requisiti

- Il meccanismo RPC deve gestire i seguenti eventi anomali:
  - incongruenze di protocollo RPC
  - incongruenze fra versioni di programmi
  - errori di protocollo (ad esempio parametri errati)
  - autenticazione fallita sulla procedura remota e identificazione del motivo del rifiuto dell'accesso
  - altre ragioni per cui la procedura remota non viene chiamata o eseguita

Sun RPC
Semantica di Interazione

- A fronte di possibilità di guasto, il cliente può controllare o meno il servizio
  - maybe
  - at least once (SUN RPC)
  - at most once
  - exactly once
- Per il parallelismo e la sincronizzazione possibile
  - operazioni per il servitore
    - sequenziali (SUN RPC)
    - paralleli
  - operazioni per il cliente
    - sincrone (SUN RPC)
    - asincrone

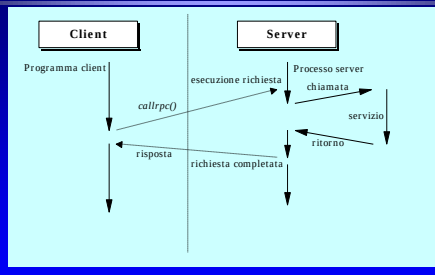
Sun RPC
Implementazione

- RPC meccanismo classificabile in due categorie secondo il grado di trasparenza
  - Open Network Computing (Sun Microsystems)
  - Network Computing Architecture (Apollo)

Open Network Computing (ONC)

- chiamata RPC Sun (diversa dalla locale)
- primitiva callrpc()
  - parametri necessari
    - nome del nodo remoto
    - identificatore della procedura da eseguire
    - specifiche di trasformazione degli argomenti
    - (eXternal Data Representation XDR)

Sun RPC
Implementazione ONC

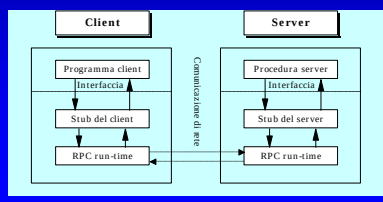


- Il meccanismo usa una tecnica nota come wrapping: uso una chiamata *callrpc()* → il programmatore sa che sta facendo una rpc → non c'è trasparenza

Sun RPC
Network Computing Architecture (NCA)

- Ai due lati della comunicazione, client e server
  - routines stub per ottenere la trasparenza

Chiamate locali allo stub: gli stub sono forniti dall'implementazione generati automaticamente → *Le parti di programma sono "del tutto" inalterate ci si dirige verso lo stub che nasconde le operazioni*

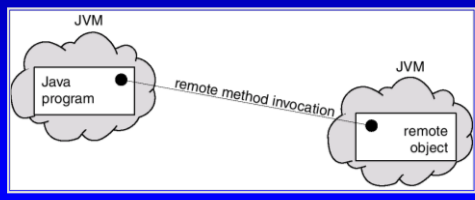


Sun RPC vs Java RMI

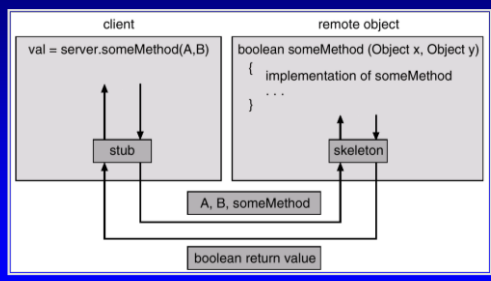
- L' infrastruttura Java consente di ottenere riferimenti ad oggetti remoti e richiedere operazioni ad oggetti che non sono residenti localmente
- Il fatto di avere Java e le classi rende necessario integrare il concetto di classi e di riferimenti remoti
- Un oggetto Java può riferire oggetti remoti solo se questi hanno predefinito una interfaccia Remota
  - Dalla parte cliente, un proxy capace di portare le richieste dall'altra parte
  - Dalla parte server, uno stub che passa la richiesta al remoteObject
- Se si passa un oggetto che possiede una InterfacciaRemota, si passa un riferimento e si creano skeleton e stub
  - La generazione degli skeleton e stub può avvenire automaticamente
- Uso normale: richiesta ad un gestore di nomi
  - Sistema di Binding:
    - un Registry per ogni nodo che registra le possibilità di servizio locale e che risponde alle richieste remote
- Trasporto:
  - uso di unica connessione TCP tra nodi per trasportare richieste e operazioni
  - efficienza, sicurezza???;

Remote Method Invocation

- Remote Method Invocation (RMI) è di fatto un meccanismo JAVA simile alle RPC.
- RMI consente a un programma JAVA in una macchina di invocare un metodo di un oggetto remoto.



Marshalling Parameters



Differenze tra RPC e RMI

1. RPC supporta programmi procedurali dove solo procedure o funzioni remote possono essere richiamate. RMI è orientato agli oggetti: egli supporta l'invocazione di metodi di oggetti remoti.
2. I parametri delle procedure remote sono delle ordinarie strutture per le RPC; Con RMI è possibile passare degli oggetti ai metodi remoti.

- Se i parametri marshaled sono oggetti locali (non remoti), essi saranno passati per copia utilizzando la tecnica conosciuta di serializzazione.
  - Object serialization consente di scrivere lo stato di un oggetto come un byte stream.

Introduzione a RMI

- RMI
  - Registra un metodo come remotamente accessibile
    - Client può ricercare il metodo e ottenere l'interfaccia
    - Ha un riferimento per richiamare il metodo
    - Sintassi molto simile a un normale invocazione di metodo
  - Marshalling of data
    - E' in grado di trasferire anche oggetti
    - Class `ObjectOutputStream` converte `Serializable` oggetti in stream di bytes
      - Trasmette attraverso la rete
    - Class `ObjectInputStream` ricostruisce gli oggetti
  - NON è necessaria una Interface Definition Language
    - Usa direttamente l'interfaccia java

Modelli e Tecnologie
Paradigmi di Coordinamento: Client/Server

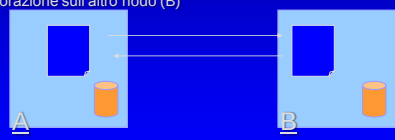
Il modello Client/Server è un modello di coordinamento, evidenzia le modalità di interazione tra entità attive

- Modello di base su cui si fonda qualsiasi interazione
- Ogni configurazione anche complessa può essere ricondotta ad uno schema Client/Server
- Per realizzare una applicazione di rete:
  - Identificazione delle identità che devono collaborare
  - Definizione dei Ruoli: Clienti e Servitori
  - Sviluppo del protocollo di interazione tra Clienti e Servitori

Modelli e Tecnologie
Paradigmi di Coordinamento: Client/Server

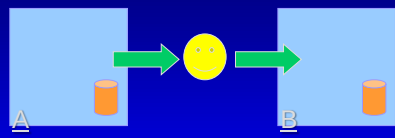
Il modello Client/Server in configurazioni particolarmente complesse crea delle strutture molto difficili da sviluppare e da gestire

La divisione in entità è dettata non solo dalla logica del servizio, ma dal concetto di "località". Se devo intraprendere una azione che richiede due risorse dislocate su due nodi remoti sono costretto a dividere l'azione tra due entità, una che effettua la elaborazione su un nodo (A) e chiede all'altra l'elaborazione sull'altro nodo (B)



Modelli e Tecnologie
Paradigmi di Coordinamento: Gli Agenti Mobili

E se l'entità potesse ***muoversi*** da un nodo all'altro per accedere alle diverse risorse ?



Si parlerebbe in questo caso di **Agente** perché non sarebbe classificabile ne come Cliente ne come Servitore

Non sarebbe più necessario lo sdoppiamento in due entità distinte per la sola necessità di località alla risorsa

Questi Agenti si dicono infatti **Mobili** perché sono in grado di svincolarsi dal limite della località. Questo non significa che non viene applicato più il modello cliente servitore, viene semplicemente utilizzato al di sopra della struttura di rete, non più vincolato dalla struttura fisica sottostante

Modelli e Tecnologie
Modelli di Computazione Distribuita

CORBA è uno standard aperto, garantisce:

- Language neutral:
- Platform independent:
- Versatilità, eterogeneità e flessibilità

La sua estrema ampiezza costa però in termini di **Complessità realizzativa**; non sono previste infatti astrazioni forti di ausilio allo sviluppo (Gestione Transazioni, Sicurezza, Replicazione).

Le tecnologie più diffuse per la computazione distribuita sono:

- **DCOM:** tecnologia proprietaria Microsoft basata sul modello a Componenti, nata dalla evoluzione del concetto di **DLL**:
  - Al di fuori degli standard della computazione distribuita dettati da CORBA
  - Realizzabile solo attraverso tool di sviluppo microsoft
  - Dipendenza dalla piattaforma WinXX
- **EJB** Enterprise Java Bean: è l'implementazione della interfaccia CORBA in Java, per rendere aperte le applicazioni java-based
  - Java based
  - Platform independent:
  - Verstile e potente
  - Semplicità realizzativa: implementa i cosiddetti "Distributed Services", grazie ai quali è possibile astrarre lo sviluppo