

REMOTE PROCEDURE CALL (RPC)

estensione del normale meccanismo di chiamata a procedura, adatta per il modello cliente servitore

IDEA per uniformare
programmi concentrati e distribuiti

APPROCCIO linguistico:

il cliente invia la richiesta ed attende fino alla risposta del servitore stesso

A differenza della *chiamata a procedura locale*

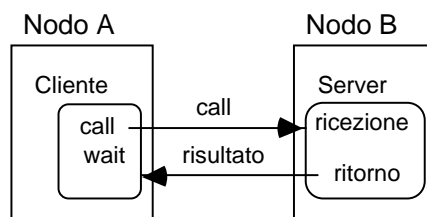
- i processi **non** condividono lo spazio di indirizzamento
- i processi hanno vita **separata**
- possono accadere **malfunzionamenti**, sia ai nodi, sia alla interconnessione

RPC

consente di considerare anche il **type checking** da livello di linguaggio del cliente al servitore

trattamento automatico degli argomenti di ingresso e di uscita dal cliente al servitore, e vice versa

marshalling



RPC

Birrel Nelson (1984)

usate in Xerox, Spice, Sun, HP, etc.

PROPRIETÀ

trasparenza approccio locale e remoto

uniformità totale è impossibile (guasti)

type checking e parametrizzazione

lo stesso controllo di tipo e dei parametri

controllo concorrenza e eccezioni

binding distribuito

possibile trattamento degli orfani

recovery in caso di fallimento: *orfano* chi resta

RPC come astrazione dello scambio messaggi

CLIENTE

SERVITORE

send

get-request <operazione>

wait

send-reply

Progetto Athena MIT, ITC CMU, ...

PRIMITIVE

dalla parte del cliente

call servizio (argomenti, risultato)

dalla parte del servitore

due possibilità:

- il servizio svolto da un unico **processo sequenziale**
- il servizio è un **processo indipendente**, generato per ogni richiesta (approccio implicito)

RELIABILITY

Malfunzionamenti

- perdita di messaggio di richiesta o di risposta
- crash del nodo del cliente
- crash del nodo del servitore

In caso di crash del servitore, prima di fornire la risposta il **cliente** può:

- aspettare per sempre;
- **time-out e riportare una eccezione al cliente;**
- **time-out e ritrasmettere (uso identificatori unici);**

Operazioni idempotenti

che si possono eseguire un numero qualunque di volte con lo stesso esito

Semantica

may-be	time-out per il cliente
at-least-once	time-out e ritrasmissioni
at-most-once	tabelle delle azioni effettuate
exactly-once	e l'azione fatta fino alla fine

In caso di crash del **cliente**, si devono trattare **orfani**

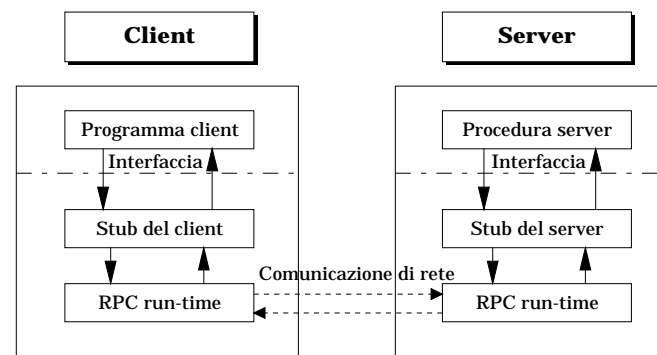
- **sterminio**: ogni orfano risultato di un crash viene distrutto
- **terminazione a tempo**: ogni calcolo ha una scadenza, oltre la quale è automaticamente abortito
- **reincarnazione (ad epoche)**: tempo diviso in epoche; tutto ciò che è relativo alla epoca precedente è obsoleto

Primo modello con trasparenza

Birrel Nelson: uso di stub

ossia di **interfacce per la trasparenza**

che trasformano la richiesta da locale a remota



Lo sviluppo prevede **trasparenza**

Il **cliente** invoca uno **stub**, che si incarica del trattamento dei parametri e della richiesta al supporto run-time, per il **trasporto** della richiesta

Il **servitore** riceve la richiesta dallo **stub** relativo, che si incarica del trattamento dei parametri dopo avere ricevuto la richiesta pervenuta dal **trasporto**
Al completamento del servizio, lo stub rimanda il risultato al cliente

modello asimmetrico a molti clienti/ un solo servitore

PASSAGGIO DEI PARAMETRI

passaggio per valore e per riferimento

nel caso di riferimento ==>

uso di oggetti confinati che sono identificati in modo unico nell'intero sistema

Trattamento dei parametri

impaccamento dei parametri e disimpaccamento

marshalling ==> dipende dal linguaggio utilizzato
unmarshalling

Exception handling

problemi tipici dipendenti dalla distribuzione e loro gestione

In un caso generale, una RPC può:

- può produrre il servizio *con successo*
- può produrre *insuccesso* e determinare una **eccezione**

In genere, si specifica la azione per il trattamento anomalo in un opportuno **gestore della eccezione**

Si possono prevedere più gestori a secondo dell'evento anomalo. Inoltre, si può anche inserire l'eccezione nello scope di linguaggio

CLU (Liskov) a livello di invocazione della RPC

MESA (Cedar) a livello di messaggio

FASI

compilazione

binding

trasporto

controllo

rappresentazione dei dati

problemi in ambiente eterogeneo

Le scelte sono diverse

scelta pessimistica e statica

La compilazione potrebbe risolvere ogni problema e forzare un **binding statico**

scelta ottimistica e dinamica

Il **binding dinamico** consente di ridirigere le richieste sul gestore più scarico o presente in caso di sistema dinamico

L'uso della **comunicazione** è intrinseco

tanto più veloce, tanto meglio

Il **controllo** consente anche di usare gli stessi strumenti per funzioni diverse, con maggiore asincronicità e maggiore complessità

necessità di **traslazione** dei dati

tanto più veloce, tanto meglio

bilanciata con

la ridondanza che viene ritenuta necessaria

TRATTAMENTO DEL BINDING

legame tra cliente e servitore

STATICO vs. DINAMICO

due fasi:

- **servizio**, il cliente specifica a chi vuole essere connesso, come nome del servizio (NAMING)
- **indirizzo**, il cliente deve essere collegato al servitore che fornisce il servizio (ADDRESSING)

NAMING

si può risolvere attraverso un numero associato staticamente alla interfaccia del servizio

ADDRESSING DINAMICO

1) si può risolvere con un multicast o broadcast attendendo solo la prima risposta e non le altre

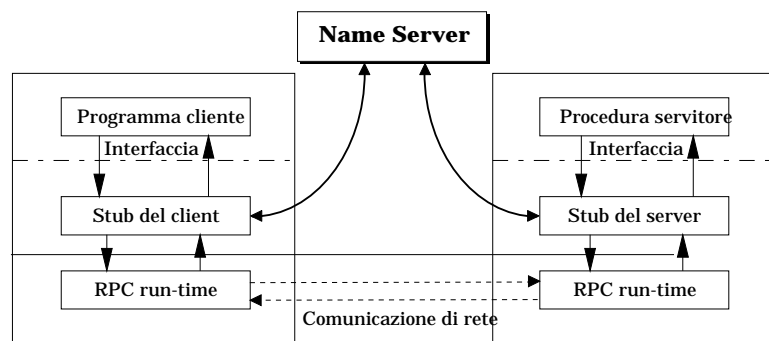
ESPLICITAMENTE dai processi

2) uso di un **name server** che registra tutti i servitori

IMPLICITAMENTE dall'agente esterno

Azioni sulle tabelle di binding

registrazione, aggiornamento, eliminazione



BINDING DINAMICO

La chiamata può andare a buon fine dopo un collegamento statico o dinamico

il binding può avvenire meno frequentemente delle chiamate stesse

*in genere, **si usa lo stesso binding per molte richieste e chiamate allo stesso server***

Binder (Broker, Name Server, etc.)

entità che consente il collegamento

tra **clienti e servitori**

possibilità di tenere conto dei **servizi**

operazioni per un binder

possono consentire anche agganci più liberi

lookup (servizio, versione, &servitore)

register (servizio, versione, servitore)

unregister (servizio, versione, servitore)

Il **nome del servitore** (servitore) può essere dipendente dal nodo di residenza o meno: se dipendente, allora una variazione deve essere comunicata al binder

BINDING come servizio coordinato

Uso di **binder multipli** per limitare overhead

Inizialmente i clienti usano un broadcast per trovare il binder più conveniente

Uso di **cache** ai singoli clienti o ai singoli nodi

RPC IMPLEMENTAZIONE

- si invia una richiesta alla volta (**stop & wait**)
- si inviano una serie di richieste alla volta (**burst**)
e se non sono arrivate
anche richieste di ripetizione selettive
(**selective repeat**)

dipendentemente dal vincolo

RPC SEMPLICI E COMPLESSE

protocollo di trasporto differenziato (UDP e TCP)
con dimensioni massime di pacchetto

RPC semplici chiamate frequenti, di breve durata,
argomenti in pacchetti corti

RPC complesse altri casi

RPC semplici

pacchetto di **invocazione** e pacchetto di **ritorno**
con ritrasmissioni e time-out (massimo numero volte)

RPC complesse

per chiamate di **lunga durata** o
per **argomenti** che eccedono un pacchetto

Il messaggio viene diviso su più pacchetti

ricostruiti dalla parte del server

In caso di perdita, o si ritrasmette tutto,
o solo le parti mancanti

Il cliente invia periodicamente dei **probe messaggi**

Il server deve mandare un acknowledgement

Interface Definition Language

Definizione di **linguaggi astratti**
per la specifica del **servizio**

Un servizio deve consentire:

- identificazione (unica) del **servizio** tra quelli possibili
uso di **nome astratto** del servizio
versioni diverse del servizio
- definizione astratta **dei dati** da trasmettere in input ed
output
uso di **un linguaggio astratto** di definizione dei dati
(uso di interfacce, con operazioni e parametri)

possibili estensioni:

linguaggio con **ereditarietà**
ambiente con **binder** ed altre **entità**

Dalla definizione del linguaggio IDL

strumenti per lo **sviluppo automatico** di parte dei
programmi direttamente dalla **specifica astratta**

OSF **DCE IDL**
SUN **XDR**
ANSA **ANSAware**
HP **NCS IDL**
CORBA **IDL**

Confronto RPC e message passing

ci sono casi anche molto simili

V-kernel message passing con **send-reply** RPC e rendez-vous

Ambienti di linguaggi a parte

Message Passing

flessibilità
performance
difficoltà di documentazione

RPC

astrazione
leggibilità
strumenti di generazione automatica di codice stub

Confronto tra

Charlotte (message passing) e **STARMOD** (RPC)
più costoso il message passing

dipendenza dalle scelte di Charlotte:

- linguaggio di implementazione
- protocollo a livelli costoso
- strutture di kernel molto diverse

RPC MOLTI A MOLTI - AZIONI DI GRUPPO

- un servitore può fornire più servizi allo stesso tempo
- un cliente può richiedere più servizi allo stesso tempo

Servitore multiplo

Si prevedono un certo numero predefinito di attività di servizio e questi sono disponibili contemporaneamente

possibilità statica, già presenti

possibilità dinamica, creati quando è il caso

È possibile anche non limitare il numero delle attività

overhead di ogni servizio (creazione attività)

tempo di servizio

tempo di attesa

Cliente multiplo

Uno stesso cliente può richiedere più servizi contemporaneamente

spesso uso di broadcast per la trasmissione

Attesa delle risposte e trattamento delle stesse

si introduce l'idea della replicazione
e del broadcast/multicast della richiesta

USO DELLA REPLICAZIONE

Alcuni dei servizi possono essere coordinati e fornire una astrazione replicata dello stesso servizio

Troupe (Cooper 85)

Replicazione completa

- i server possono essere multipli e fornire tutti servizi dello stesso tipo
- i clienti sono tutte copie e richiedono gli stessi servizi a server identici

SEMANTICA

exactly-once

su tutti i componenti della troupe

copie tutte attive

replicazione con copie tutte attive ed in esecuzione

==> coordinamento ad ogni messaggio

Questo modello di replicazione con tutte le copie attive che si devono coordinare ad ogni azione è molto costoso

RPC asincrona

il cliente non si blocca ad aspettare il server

Possibilità di modalità asincrona a livello di ambienti di uso (a volte si tratta di modi non bloccanti)

- maggiore parallelismo ottenibile
- processi leggeri per il server

problema fondamentale

==> come ottenere il **risultato**
soluzione possibile
non averlo per niente

due punti di vista di progetto

RPC bassa latenza vs. RPC alto throughput

le prime tendono a mandare un messaggio di richiesta ed a trascurare il risultato (**bassa latenza**)

le altre tendono a differire l'invio delle richieste per raggrupparle (**throughput elevato**)

IMPLEMENTAZIONE

si possono usare supporti UDP o TCP

e

si ottengono semantiche diverse

Athena

UDP e bassa latenza
semantica may-be

SUN

invio di una serie di RPC asincrone e di una finale
TCP ed eleva throughput
semantica at-most-once

Batching

in questi casi *non c'è il valore di ritorno*

con *valore di ritorno*

Chorus

bassa latenza
uso di una variabile che contiene il valore

Mercury

alto throughput
uso di stream per le richieste tenendo conto di
azioni asincrone e sincrone

TRANSAZIONI

Possibilità di computazioni affidabili nel distribuito

mantenere la consistenza globale

con possibilità di

ridondanza delle risorse

Una sola comunicazione non provvede l'intera semantica
di interazione necessaria

accesso a dati memorizzati su ***nodi diversi***

necessità di coordinamento con il

sistema di comunicazione

Distributed Transaction

Spector 1986

basate sulla comunicazione

Primitive per le transazioni

nel concentrato

beginTransaction
endTransaction

nel distribuito

richiesta == beginTransaction
risposta == endTransaction

NOME UNICO della transazione

TRANSAZIONI

proprietà (ACID)

- **Atomicità** (a fronte di guasti)
- **Consistency** (solo stati corretti)
- **Isolation** effetto come in modo sequenziale (**serializzabilità**)
- **Durability** o permanenza

PRIMITIVE

sendTransRequest (header, argomenti, risultato)

header: sender, receiver, servizio, TID, altro

getTransRequest (sorgente, listaargomenti)

sorgente, controllo di accettazione

lista ottenuta dal messaggio

replyTrans (header, listarisultati)

lista trasferita ai risultati

POSSIBILI errori

- messaggi TransReq o TransReply perduta
- nodo in crash o non più raggiungibile

AZIONI di RECOVERY conseguenti

sender: time-out e ritrasmissioni

receiver: uso di TID per non rifare azioni già fatte

SEMANTICA delle azioni conseguenti

at-least-once == > azioni idempotenti

exactly-once == > azioni non idempotenti

Operazioni IN CASCATA

Transazioni Innestate

innestamento gerarchico

- maggiore concorrenza (allo stesso livello)
- minori legami tra i diversi livelli: un livello può arrivare al commit anche se un livello inferiore fallisce

Decisioni semantiche in caso di azioni innestate

anche ripetizione di transazioni interne

o via alternative

IMPLEMENTAZIONE

Uso di strumenti di comunicazione a basso livello

costrutti privati ai processi

possibilità di canali virtuali o datagrammi

attribuzione dinamica e statica delle socket

su queste primitive e costrutti

==> progetto del protocollo **CCR**

Commitment, Concurrency control & Recovery

Necessità di un protocollo per **non** incorrere in un tempo infinito per il completamento della azione o nella necessità di mantenere informazioni per sempre

CCR Commitment, Concurrency and Recovery
Problemi

- guasto di nodo
- necessità di coordinare più nodi in una azione

CCR implementazione

un **master** e gli **slave** che devono seguire
il protocollo CCR
con possibilità di innestamento

eventi significativi:

- inizio di azione atomica (**C-begin**)
- fase di scambio di messaggi di applicazione
- il master conclude la azione
esplicitamente invia una **C-prepare**
implicitamente indica il completamento
- gli slave decidono
commitment con **C-ready**
rifiuto con **C-refuse**
- il master ordina con conferma o il completamento o il
ritorno allo stato precedente
commitment con **C-commit**
disfare (roll-back) con **C-rollback**
- gli slave rispondono ed il master può dimenticare
l'azione

Two-phase commit protocol

Protocollo

Master

Slave (N altri)

C-begin

send (request1)
send (request2)

send (requestN)

receive (request1)

C-prepare

C-prepare

if (OK) { lock degli oggetti;
memorizza stato e richiesta;
C-ready }
else **C-refuse**

if (tutti gli slave C-Ready)

C-commit

else **C-rollback**

attesa delle risposte dagli slave

C-rollback

if (**C-commit**) {
esegui la azione;
sblocca gli oggetti }
else {**abort** dell'azione;
libera oggetti; }

Durata del protocollo

Terminazione con insuccesso

Two phase commit protocol



ANCHE ALTRE POSSIBILITÀ DI REALIZZAZIONE DEL PROTOCOLLO

Heuristic commitment

uno slave può, dopo un crash, decidere unilateralmente di fare una azione di recovery o di commit

Euristica, nel capire cosa vale la pena di fare, pagando in inconsistenza la decisione sbagliata
Tutto il protocollo CCR fallisce

CONTROLLO DELL'ACCESSO

Per la proprietà delle transazioni, l'accesso alle informazioni deve essere regolato

all'estremo **serializzato**
o con gli **stessi effetti** ma **maggiore concorrenza**

Uso di strategie diverse: **pessimiste ottimiste**

pessimiste

si previene una qualsiasi interferenza impedendo cambiamenti scorretti

ottimiste

si effettuano i cambiamenti fino alla fase finale; sono però attuati solo al termine se non ci sono problemi, altrimenti vengono scartati

LOCK

protocolli a due fasi per gestire lock

proprietà e granularità dei lock

con possibilità di distinguere anche tipi di lock diversi:

esclusivi o distinti (read/write)

lock con time-out

gerarchie di lock

Problema del **deadlock** (nel **distribuito**)

prevention

acquisizione di tutte le risorse all'inizio

detection/recovery

si mantengono dei grafi di richieste e disponibilità da verificare dinamicamente

timestamp

Uso del **tempo** per dirimere i conflitti tra azioni di accesso

timestamp per ogni transazione
spesso associati all'uso di lock

timestamp

con un tempo definito per ogni transazione
ordinamento deciso all'inizio della transazione
abort immediato in caso di conflitto

lock a due fasi

ordinamento deciso ad ogni possibile punto di conflitto
attesa in caso di conflitto

metodi ottimistici

Kung

scarsa probabilità di conflitto

- 1) Fase di azione di **tentativo** sulle informazioni
- 2) Fase di **validazione** delle operazioni
- 3) Fase di copia dei valori **tentativo** sulle informazioni

In caso di interferenza, si deve ripristinare uno stato di consistenza

definizione di una politica
costo della implementazione

SINCRONIZZAZIONE

vincolo sull'ordine temporale degli eventi di un sistema

necessaria per provvedere una **vista consistente** del sistema alla **globalità** dei processi che comunicano

Comunicazione e sincronizzazione sono correlate

Ad esempio:

- *sincronizzazione sender receiver* di un messaggio
- controllo di attività *cooperanti*
- *serializzazione di accessi* a risorse condivise

N processi ed una risorsa (concurrency control)

quindi, uso di **ordinamento degli eventi significativi**

Ordinamento dei tempi di Lamport

uso di timestamp (indicatori di tempo) per etichettare gli eventi ed ordinarli ==>
clock logici e fisici e relazione "happened before"

Token passing

token che viene passato in un anello logico e consente di ordinare gli eventi

Ordinamento degli eventi sulla base della priorità

uso della priorità dei processi per ordinare gli eventi correlati
sistemi real-time

RELAZIONE "HAPPENED_BEFORE" →

Ordinamento degli eventi per un insieme di processi che comunicano attraverso message passing derivato dalla causalità

- 1) Se a e b sono eventi dello stesso processo ed a è eseguito prima di b , allora $a \rightarrow b$
- 2) Se a è l'evento di invio e b l'evento di spedizione di un messaggio, allora $a \rightarrow b$
- 3) Se $a \rightarrow b$ e $b \rightarrow c$, allora $a \rightarrow c$

La relazione \rightarrow introduce un **ordinamento parziale** degli eventi di un sistema

Due eventi **concorrenti** se **non** $a \rightarrow b$ e **non** $b \rightarrow a$

Non esiste un unico orologio globale (**tempo globale**), ma un insieme di clock locali (**tempi locali**)

È possibile un ritardo di trasmissione, variabile e più grande di ogni evento osservabile nel sistema

==> Necessità di un **ordinamento globale o totale** per la sincronizzazione

Clock fisici

Unico tempo se assumiamo di avere

- un unico clock disponibile a tutti i nodi oppure
- un clock per ogni nodo e che i clock siano perfettamente sincronizzati

purtroppo,

- orologi globali non esistono
- garantire la sincronizzazione di orologi comporta un notevole overhead e un margine di errore

Uso di trasmissione radio dell'ora

Clock logici

Il clock assegna un numero ad ogni evento significativo

La funzione $TS(i)$ assegna un valore ad ogni evento

happened_before → è solo parziale

Se $a \rightarrow b$, allora il timestamp logico degli eventi è tale che $TS(a) < TS(b)$

Condizione di clock

C. Per $\forall a$ e b , se $a \rightarrow b$, allora $LC(a) < LC(b)$

La funzione di clock logico globale **LC** è tale che per ogni evento a che appartenga al processo i, si ha $LC(a) = LC_i(a)$

CONDIZIONI

- C1. Per $\forall a$ e b , se $a \rightarrow b$ nel processo P_i , allora $LC_i(a) < LC_i(b)$
- C1. Per $\forall a$ e b , se a è l'invio di un messaggio nel processo P_i e b la ricezione nel processo P_j , allora $LC_i(a) < LC_j(b)$
- I1. Ogni processo P_i incrementa C_i tra due eventi
- I2. Per $\forall a$ invio di un messaggio nel processo P_i , il messaggio contiene il clock come timestamp
 $TS = LC_i(a)$
- I3. Per $\forall b$ ricezione di un messaggio nel processo P_j , il processo mette il clock logico al valore maggiore del clock corrente e del timestamp
 $LC_j = \max(TS_{ricevuto}, LC_{corrente}) + 1$

Questo produce una relazione d'ordine parziale

È necessario introdurre una **relazione di ordine** totale tra tutti i processi del sistema

Le condizioni garantiscono un sistema di **clock logici** che consente di definire una relazione di **ordine globale** tra tutti gli eventi nel sistema \Rightarrow

relazione d'ordine totale \Rightarrow

Se a è un evento in un processo P_i e b un evento in un processo P_j , allora $a \Rightarrow b$ se e solo se

- R1) $LC_i(a) < LC_j(b)$ oppure
R2) $LC_i(a) = LC_j(b)$ e $P_i < P_j$

Processo coordinatore

Un unico processo coordina tutti gli altri

Approccio completamente centralizzato

Ogni processo invia a tutti gli altri le proprie richieste e il processo coordinatore decide gli accessi alla risorse in mutua esclusione

Tre messaggi per ogni sezione critica

- 1) un processo che vuole richiedere la risorsa invia un messaggio di richiesta (**request**)
- 2) Il coordinatore, a fronte delle richieste decide, se la risorsa è libera di rispondere ad una richiesta per volta (**reply**)
- 3) al ricevimento del reply, il processo usa la risorsa e al termine, la libera mandando un messaggio di rilascio al coordinatore (**release**)

VANTAGGI

- mutua esclusione automatica
- non conoscenza reciproca
- lo scheduling è FIFO, secondo l'accodamento al coordinatore

SVANTAGGI

- carico di messaggi
- ritardo nel servizio
- il fallimento del coordinatore, se ne deve eleggere uno nuovo

SINCRONIZZAZIONE

Un protocollo di sincronizzazione con uso dei clock logici

Un insieme di N processi che devono accedere ad una risorsa singola in mutua esclusione

La risorsa deve essere assegnata ad un processo per volta, che la rilascia alla fine dell'uso

*Richieste diverse devono essere servite **in ordine***

Assunzioni:

- messaggi diversi da un processo ad un altro devono arrivare nell'ordine di generazione
- i messaggi possono essere ritardati ma non persi
- la connessione tra i processi è completa e diretta

Ogni processo ha una coda locale dei messaggi, a cui i messaggi sono accodati, che contiene inizialmente, per tutti i processi, il messaggio $T_0:P_0$, inferiore di ogni clock del sistema

AZIONI

- 1) Il processo P_i manda il messaggio $T_m:P_i$ ad ogni altro processo (anche alla propria coda) per segnalare l'intenzione di accedere alla risorsa
- 2) Alla ricezione del messaggio $T_m:P_i$ il processo P_j (nella propria coda) invia una risposta con il proprio timestamp
- 3) la risorsa è data al processo P_j se sono verificate le condizioni:
 - C'è una richiesta $T_m:P_i$ nella sua coda delle richieste ordinata prima di ogni altra (relazione \Rightarrow)
 - Il processo P_j ha nella sua coda delle richieste un messaggio da ogni altro processo con Timestamp superiore a T_m
- 4) Il rilascio avviene rimuovendo il messaggio dalla propria coda e inviando un messaggio di rilascio con il proprio timestamp ad ogni processo
- 5) Un processo P_j riceve la richiesta di rilascio e rimuove il messaggio di richiesta dalla propria coda

L'algoritmo è un algoritmo con *nessuna centralizzazione* un algoritmo **completamente distribuito**

Numero di messaggi $3 * (N-1)$
per ogni azione sulla sezione critica

ALTRO PROTOCOLLO

Con un **Numero di messaggi** $2 * (N-1)$
per ogni azione sulla sezione critica

- 1) Il processo P_i manda il messaggio di richiesta ad ogni altro processo per accedere alla risorsa
- 2) Alla ricezione del messaggio $T_m:P_i$ il processo P_j invia
 - un reply immediato se non necessita la risorsa o il richiedente ha priorità superiore alla sua
 - un reply ritardato se sta usando la risorsa

Ci sono quindi $N-1$ messaggi dal richiedente ed $N-1$ da tutti gli altri

L'algoritmo è **completamente distribuito**
libero da deadlock a da starvation

PROBLEMI

- tolleranza ai guasti
- necessità di conoscenza reciproca

Token passing

Token che circola tra i processi membri
Solo chi ha il token può entrare nella sezione critica

Architettura logica
Anello logico (**Ring**)

Ring logico

Ogni processo è organizzato con un precedente ed un successivo

Chi riceve il token,

- verifica che sia diretto a lui e
- (dopo un eventuale uso del token stesso)
- si prepara ad indirizzarlo al successivo

Un solo processo accede alla risorsa
non è possibile starvation,
se il ring viene percorso in un verso solo

Numero di messaggi $N-1$
per il passaggio del token

Se ogni processo può tenere il token solo per un intervallo limitato (si fissa il massimo), il ricevimento del token avviene entro un tempo predefinito dipendente solo da N

GUASTI (FAULT)

Un guasto può fare perdere il token
==> si vuole evitare la situazione senza token
o con più di un token circolante

PROTOCOLLO DI ELEZIONE (Lelann)

Ogni nodo ha un **timer**, con un tempo di attivazione che dipende dal numero dei nodi (N) e dal tempo di permanenza massimo del token per nodo

Il **timer** viene attivato quando il token è inviato al vicino

- Al risveglio, il processo P crea un token di elezione (**ET**) con il nome del processo ed entra in stato di elezione
- se P riceve il normale **token** prima che l'ET generato sia tornato, la elezione era inutile ed è terminata (ET distrutto al ritorno)
- se ogni altro processo riceve un ET, lo registra in una lista di elezioni, insieme con l'identità del processo che l'ha generato e lo passa nel ring
Se ha già generato un token ET, verifica la priorità statica assunta e assente o meno
- se P riceve il suo ET ancora in fase di elezione, lo rimuove e verifica la lista di registrazione.
Si sceglie di generare il nuovo token, se il nodo P è il nodo di indice minimo nella lista di registrazione

Anche il massimo del tempo di **rigenerazione** del token può essere calcolato

Naturalmente, si devono anche controllare i vicini, per non incorrere in blocchi

ALTRI PROTOCOLLI DI ELEZIONE

Sono necessari protocolli di elezione ogni volta che si deve trovare un accordo tra un insieme di partecipanti

in caso di fault e di recovery in un gruppo

Algoritmo BULLY

supponiamo che esista un ordinamento statico tra i partecipanti

Ogni partecipante P_i che rilevi la necessità di fare una elezione (evento locale a ciascuno) o in recovery
messaggi elezione **Election**

risposta **Answer** annuncio **IAMCoordinator**

1) invia un messaggio di elezione ai processi più prioritari

1bis) in caso di messaggio di **elezione**, si risponde e si fa partire una nuova elezione

2) dopo un certo **tempo**,
se riceve **Answer** aspetta messaggi di coordinamento superiori
se non arriva **nessun** messaggio da quelli prioritari, diventa il coordinatore (messaggio **IAMCoordinator**)

3) invio del messaggio **IAMCoordinator** ai nodi di priorità inferiore

Sequencer

Il token può anche portare un distributore di indicatori di ordinamento (**ticket**)

I ticket rappresentano un modo di ordinare globalmente tutti gli eventi nel sistema

ticket (S)

ritorna la chiamante un valore intero, e poi incrementa il sequenziatore

Spesso usato con EVENTCOUNT

Oggetto astratto per garantire l'ordinamento direttamente ai processi, senza passare attraverso la mutua esclusione

==> tiene conto degli eventi rilevanti nel sistema

primitive

advance (E)

aumenta il valore dell'oggetto E

il processo che ha fatto la advance diventa il segnalante

read (E)

riporta il valore dell'oggetto E

await (E, v)

sospende il processo che la esegue finché il valore di E raggiunge v

Le primitive possono avvenire insieme: una advance può avvenire insieme con read ==>

Le read possono riportare il valore precedente o successivo

NON si usa mutua esclusione

Valutazioni

Criteria di valutazione di uno

strumento di sincronizzazione

fairness e convergenza

non si deve produrre nessuna forma di starvation ed i processi devono avere gli stessi diritti

comportamento deterministico

la sincronizzazione deve essere sempre garantita (non probabilistica)

resiliency

capacità di sopravvivenza ai guasti (in numero) e di recovery da errori

dinamicità

possibilità di inserimento di nuovi nodi e di variazioni

tempo di risposta e throughput

traendo vantaggio dalla distribuzione

overhead

costo limitato, in comunicazione, processing e memoria

connettività

ipotesi di località nella connessione dei nodi e dei processi

comprensibilità e facilità di uso

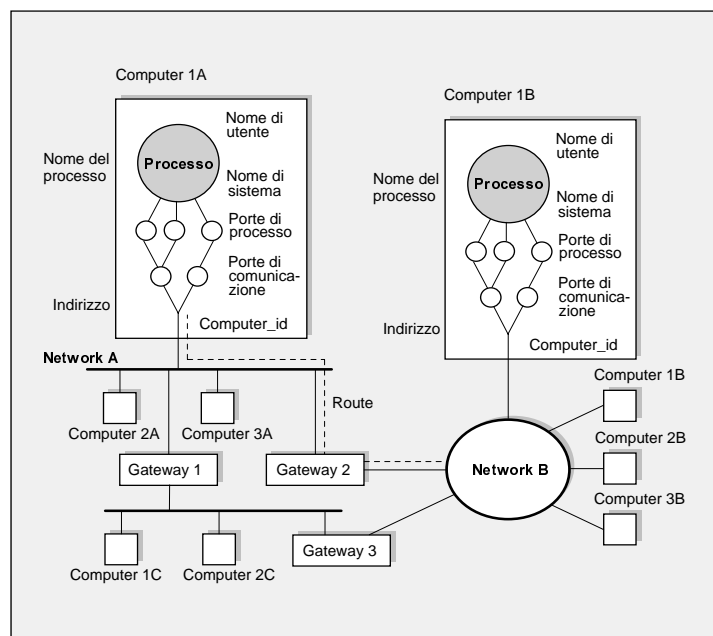
facili prove di correttezza e di uso anche da parte di utenti inesperti

Soprattutto

*come si garantisce la **robustezza delle realizzazioni** a fronte di **guasti differenziati***

SISTEMA DI NOMI

- **generalità**
varietà di nomi
- **definizioni multiple** della stessa entità
varietà di nomi per lo stesso oggetto con mapping capace di traslare
- **distribuibilità**
uso di direttori partizionati o replicati
- **user-friendliness**



Presenza di molti livelli di nomi

CARATTERISTICHE DEL SISTEMA DI NAMING

Contesti di definizione dei nomi

- *Uso* generalizzato del nome nei diversi contesti e strumenti
- livelli di nomi diversi e relative funzioni di *mapping*
- minimizzazione dei servizi di naming
- interazione con *contesti* di validità dei nomi
- *replicazione*
nomi che consentano la replicazione di oggetti
- *possibilità di riallocazione (bilanciamento)*
uso di nomi che consentano il movimento di oggetti
- *multicast e broadcast*
definizione di gruppi di oggetti e di azioni sui gruppi

Esempi di nomi

uso **esplicito** del nome o dell'indirizzo:

oggetto X o all'indirizzo X

per **contenuto**, ossia per l'oggetto con valori di attributi: *oggetto il cui campo Z vale Y*

(o in relazione ad un valore: $Z < 34$)

uso di una **forma esplicita**: *tutti i miei files*

identificatore **broadcast**: *tutti gli oggetti*

identificatore di **gruppo**: *tutti il gruppo Z*

identificatore di **cammino** o **route**:

oggetto che si trova seguendo il cammino K

in relazione ad alcune delle **specifiche precedenti**:

tutti gli oggetti determinati sopra

NAMING

Problema fondamentale nel **distribuito**
necessità di ritrovare (cioè **identificare**) le altre entità
nel **sistema**
Complessità del problema (anche concentrato) e
difficoltà di soluzioni generali

entità diverse eterogenee =>
livelli diversi di nomi

più sistemi di naming presenti
più livelli di naming nel sistema

con **contesti di visibilità**
più funzioni di trasformazione da nome all'entità
nel sistema

NOMI

identificatore come

- stringa di caratteri o
- numero binario

nomi esterni (nomi di utente)

nomi interni (nomi di sistema)

oggetti

nome di utente (significativi)
nome di sistema

LIVELLI DI NOMI

NOME organizzato(Shock)
in **tre possibili livelli**

- **nome** LOGICO esterno
- **indirizzo** FISICO
- **route** sistema per la raggiungibilità

nome

specifica *quale oggetto* (entità) si riferisce
denota la entità

indirizzo

specifica *dove* l'oggetto risiede
riferisce dopo un collegamento (**binding**)

route

specifica *come* raggiungere l'oggetto

Funzioni di corrispondenza **MAPPING**

nomi ==> indirizzi
indirizzi ==> route

nomi scelti **dall'utente**
indirizzi assegnati **dal sistema**

NAMING: problemi

ambiguità con una sola forma di indirizzo

nomi di gruppo ==> non associati alla allocazione
con possibilità di cambiamento della allocazione alle
variazioni del gruppo

indirizzi assoluti ==> non collegati alla allocazione
indirizzi Ethernet

necessità di definire il mapping

tra livelli di nomi diversi

Indirizzi come tramite tra nomi e route

nomi ==> **statici**

route ==> **dinamici**

indirizzi consentono il mapping

NOME oggetto linguistico che distingue un oggetto in
una collezione di oggetti (dominio) **denotazione**

Proprietà

NOMI GLOBALI O LOCALI

NON AMBIGUITÀ

nomi unici

nomi multipli per lo stesso oggetto (alias)

NOMI STATICI E DINAMICI

NOMI E CARDINALITÀ

nomi specifici

nomi di gruppo

SPAZIO DEI NOMI

piatto (flat)

nessuna struttura

partizionato

gerarchia e contesti (DNS)

deis33.cineca.it

descrittivo

con riferimento ad una struttura di **oggetto**

uso di attributi (OSI X.500)

username e password

attributi con liste di valori (rigidi o meno)

Nomi di gruppo

un valore di attributo identifica una lista di nomi

Esempio:

sistema di naming piatto statico e globale ==> **assoluto**

per sistemi con indipendenza dalla comunicazione
si cambiano le **tabelle di indirizzamento**

COMPONENTI DI UN SERVIZIO DI NAMING

- **Name Server** (anche più di uno)
- **Comunicazione** con il name server
- Gestione **tabelle** e coordinamento
- Gestione dei **nomi** veri e propri (coordinamento)

Name server

Oggetti con tuple di attributi con operazioni

Query

ricerca un oggetto

AddTuple / DeleteTuple

aggiungi/togli una tupla dal server

ModifYTuple

modifica una tupla

Enumerate

lista tutte le tuple, una alla volta

Realizzazione con

UNICO SERVIZIO VS. AGENTI MULTIPLI

Il servizio può essere centralizzato, distribuito (replicato)

COMUNICAZIONE

considerata tra servitori/agenti (e tra loro)

- datagrammi
- connessioni
- RPC

GESTIONE TABELLE E COORDINAMENTO

problemi di

- consistenza
- reliability

Gestione dei nomi veri e propri

Due temi fondamentali

- **Distribuzione** dei nomi
- **Risoluzione** dei nomi

Nomi

dipendenti dalla locazione

dipendenti dalla autorità (uso di domini)

organizzati in gerarchia

(uso di un albero unico riconosciuto di domini)

liberi da struttura

uso di un insieme di attributi e del loro valore

Distribuzione dei nomi

I nomi sono mantenuti in oggetti che hanno la responsabilità ==> **autorità**

partizionamento tra i server responsabili

Come dividere la gestione e il mantenimento?

Clustering

Algoritmico (hash)

Sintattico (pattern matching)

Basato su Attributi (tuple)

Risoluzione dei nomi

- trovare la autorità corretta
- verificare le autorizzazioni alla operazione
- eseguire la operazione

Ogni nodo specifica i name server noti

Limitazione delle comunicazioni tra i server

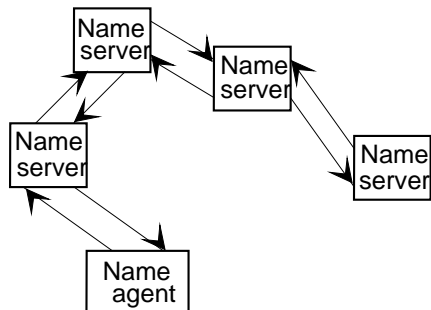
IMPLEMENTAZIONE (VEDI DNS)

USO DI CONTESTI E LOCALITÀ

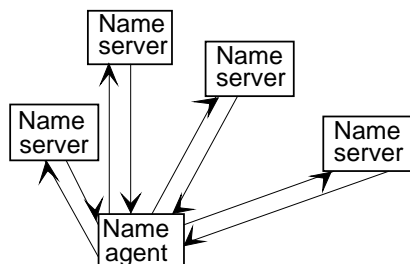
Si distribuisce e risolve nel contesto locale

Si ricorre ad altri contesti solo in caso sia necessario

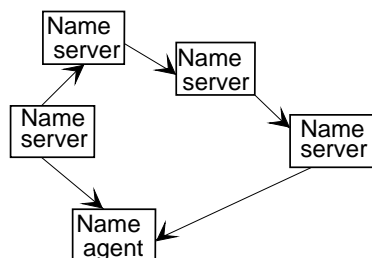
STRATEGIE DI COORDINAMENTO TRA SERVER



Risoluzione ricorsiva



Risoluzione iterativa



Risoluzione transitiva

Altri sistemi di nomi

OSI X.500 - Servizio di Direttorio e di Nomi

partizionato e decentralizzato

standard e omogeneo

CCITT definisce X500 come "una collezione di sistemi aperti che cooperano per mantenere un database logico di informazioni sugli oggetti del mondo reale. Gli utenti della Directory possono leggere o modificare l'informazione, o parte di essa, solo se hanno i privilegi necessari"

Per superare i limiti di DNS

CCITT	ISO	TITLE
X.500	9594-1	Overview of Concepts, Models and Services
X.501	9594-2	Models
X.509	9594-8	Authentication Framework
X.511	9594-3	Abstract Service Definition
X.518	9594-4	Procedures for Distributed Operation
X.519	9594-5	Protocol Specifications
X.520	9594-6	Selected Attribute Types
X.521	9594-7	Selected Object Classes
X.525	9594-9	Replication

X.500 organizza un albero logico per il Directory Information Base DIB, chiamato Directory Information Tree (DIT)

L'albero è costruito in base al **valore di attributi**

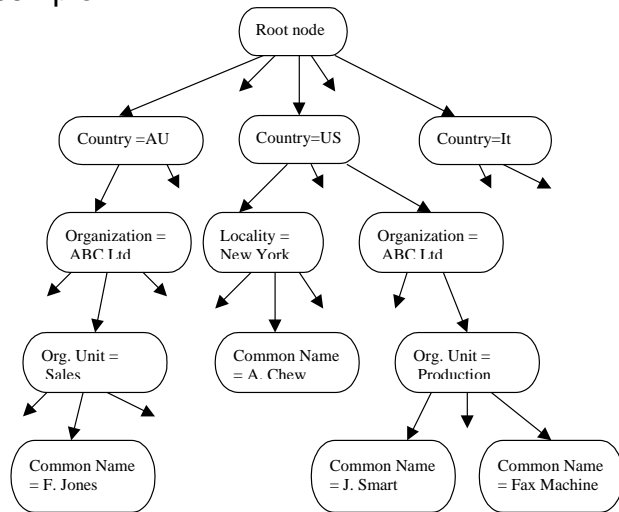
country, organization, organization unit, common name

ATTRIBUTI DEL TUTTO LIBERI

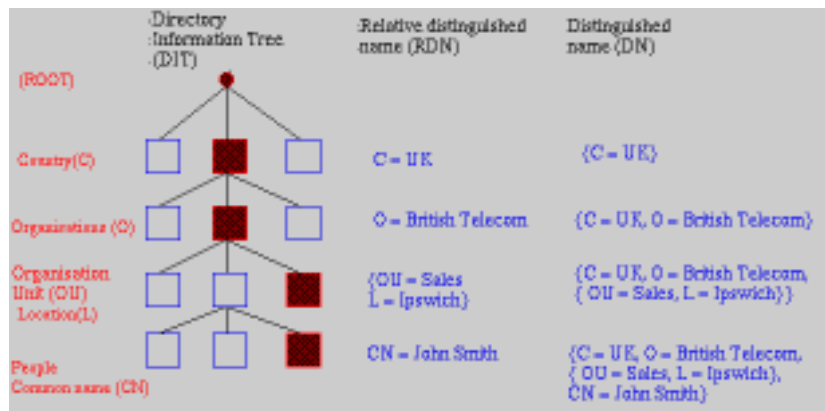
RICERCHE ANCHE SU VALORI DEGLI ATTRIBUTI

ORGANIZZAZIONI BASATE SUL CONTENUTO

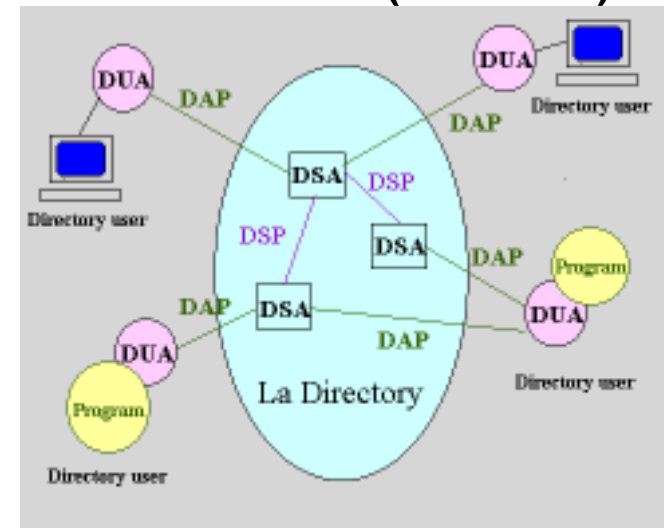
Ad esempio



Ogni entry si ritrova attraverso diverse notazioni
Distinguished Name (DN) che identifica univocamente l'oggetto all'interno del DIT
Relative Distinguished Name (RDN) che definisce univocamente un oggetto all'interno di un contesto



Ricerca sul direttorio (OSI X.500)



La ricerca avviene attraverso agenti:
DUA, directory user agent
tramite per fare richieste
DSA, directory system agent
che mantiene informazioni X.500 di un contesto
DSP, directory system protocol
per scambiare informazioni tra DSA

DAP, Directory Access Protocol

protocollo di accesso al direttorio usato da DUA
 si crea una connessione e poi si fanno operazioni di lettura, confronto, ricerca, lista delle entità
 Tecniche di ricerca **ricorsive** ed **iterative**

LDAP, Lightweight Directory Access Protocol usato per infrastrutture di certificazione e verifiche

SVILUPPO DEI SISTEMI DI NOMI

Due forme di evoluzione

Protocolli di Directory

Protocolli di Discovery

Considerando una entità che possa avere attributi con lente variazioni e attributi con variazioni veloci

Directory

soluzioni di **nomi globali**
servizi **completi** e complessi
costo elevato delle operazioni

Discovery

soluzioni di **nomi locali**
servizi **essenziali** e funzioni limitate
costo limitato adatto a variazioni rapide

Ad esempio: Un utente generico
che si muova in un **sistema globale**
vuole avere accesso a

informazioni globali, come
descrizione dei dispositivi,
delle preferenze proprie del suo profilo,
delle sue firme digitali e PKI
delle sue sorgenti di informazioni, ecc.

informazioni locali, come
descrizione delle risorse locali
dei gestori presenti, ecc.

Protocolli di Directory

Un servizio di **directory**

garantisce le proprietà di **replicazione**, **sicurezza**,
gestione multiserver, ...
*supporto per memorizzare le informazioni organizzate
prevedendo molti **accessi in lettura** e poche **variazioni***

UPnP (Universal Plug-and-Play)

Standard vicino alle architetture Microsoft

Servizi di Nomi basati su variazioni di DAP (o LDAP)

Windows2000 propone **Active Directory** come un servizio
di direttori integrato nel e per il sistema operativo

Salutation

Service Location Protocol (RFC 2165)

- si possono registrare servizi diversi
- i servizi vengono divisi in località distinte
- i servizi vengono protetti in diversi modi
- interfacce compatibili con i browser (Web)
- uso di sistemi di nomi URL

Le **operazioni definite e implementate** permettono di fare
ricerche molto evolute sulle informazioni (memorizzate in
modo globale) e compatibili con la maggior parte degli
strumenti e dei sistemi di nomi più diffusi

<http, research, homepage>

<printer, local, postscript>

Ci sono già implementazioni: Cisco, Apple, Novell
area del **Mobile Computing**

Protocolli di Discovery

Computazione distribuita e cooperativa in ambito locale

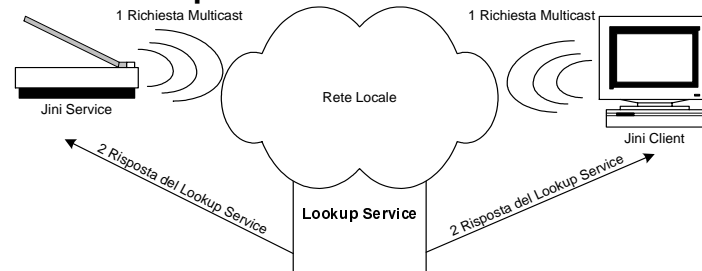
Una unità deve ritrovarne altre, in modo veloce e economico si prevedono azioni come il broadcast e solleciti periodici un servizio di **discovery** definisce e standardizza come si **possano ritrovare** altre entità correntemente visibili (**informazioni di località delle risorse**)

JINI

protocollo Java per il **discovery** (appliances)

Si vuole rispondere alle esigenze di chi arriva in un contesto e vuole operare senza conoscenze predefinite

Protocolli di lookup



Il server può passare (tramite **lookup**) al cliente:

- anche codice (che si può eseguire localmente) integrazione con **codice mobile**
- riferimento al server (da interrogare in remoto - RMI)

Start up con **multicast in ambiente locale**

Il **discovery server** verifica la presenza delle **risorse** ad **intervalli opportuni**

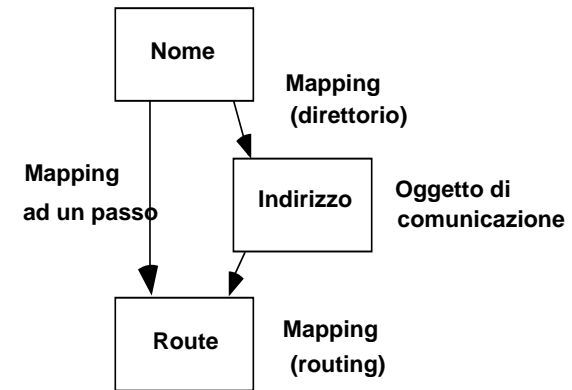
INDIRIZZI (NOMI DI SISTEMA)

forma intermedia tra **nome** e **route**

NOMI ==> entità

INDIRIZZI ==> orientati alla comunicazione con oggetto attraverso il **BINDING** usato per generare la route

Quindi, l'indirizzo serve per la comunicazione con l'entità vedi OSI o comunicazione attraverso porte



INDIRIZZI come per i nomi

- piatti o partizionati
- statici o dinamici
- individuali o di gruppo

MAPPING ad un unico passo, a più passi

routing per nome (by name)

routing per indirizzo (by address)

Internet

Nomi di alto livello, convertiti in nomi DNS

Nomi di IP di dominio convertiti in IP

Routing fatto a livello di IP

NOMI DI SISTEMA

Necessità di ottenere, a livello di sistema, dei nomi che siano **unici** o nomi che consentano un **accesso protetto**

NOMI UNICI

Identificatori **non strutturati** o strutturati

VANTAGGI DELLA NON STRUTTURAZIONE

- Facilità di memorizzazione
- Nomi assoluti ed uniformi
(indipendenza dalla allocazione)
- Composizione di oggetti e tipaggio

SVANTAGGI DELLA NON STRUTTURAZIONE

- difficoltà di creare nomi unici globali
- difficoltà per oggetti con versioni e replicazione

NOMI UNICI in un sistema distribuito

ottenuto con

- **concatenazione gerarchica**
nome_nodo @ nome_oggetto
nome_server @ nome_oggetto (**a server**)
nomi di lunghezza variabile e non trasparenza
- **approccio uniforme (a priori)**
partizione dei nomi globali per fornire ogni nodo con i propri identificatori
scelta locale dei nomi più adatti e riassegnamento
- **concatenazione**
ottenuta con il dominio ed il tempo di generazione
nome_nodo | tempo_fisico

NOMI UNICI

PILOT

Un oggetto è identificato da un intero di 64 bit
Nomi unici in spazio e tempo

COM

sistema di nomi per i componenti con identificatori globali

GUID=globally unique identifiers

{32bb8320-b41b-11cf-a6bb-0080c7b2d682}

GUID intero di 128 bit (16 byte) assegnato per garantire l'unicità nello spazio (48 bit) e nel tempo (60 bit)

V-kernel

Un oggetto è identificato da un suo gestore e da un identificatore all'interno del dominio del gestore

{manager_ID, local_object_ID}

Un processo è composto da

{node_ID, local_unique_ID}

Gli identificatori sono indipendenti dalla allocazione

APPROCCIO A SERVER

Processi gestori sono i server

nome_server @ nome_oggetto

Si usano gli ObjectID locali a piacere, anche replicati

Con l'*approccio a server*, è possibile ottenere anche più implementazioni dello stesso servizio

==> ad esempio, usando una porta intermedia a cui il server è agganciato

variazioni in tempo

Il server crea un **dominio di nomi locali**

IDENTIFICAZIONE DEL SERVER

Si usano porte per ritrovare il server per ogni oggetto di interesse

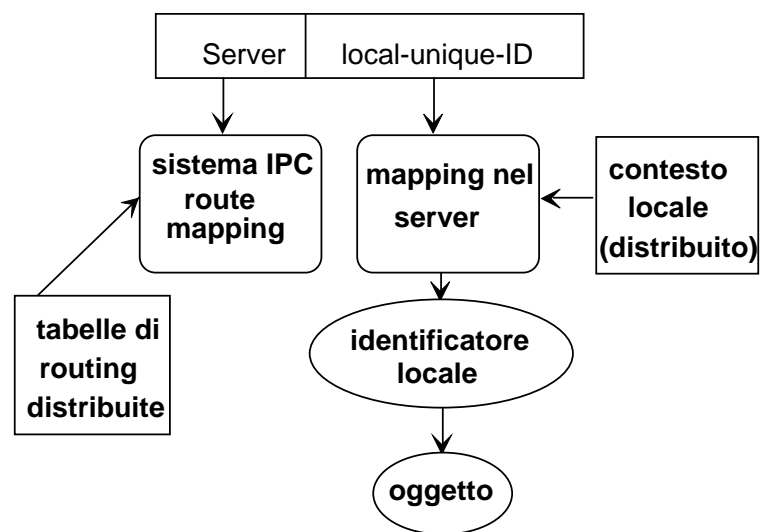
{PortAddress, local_unique_ID}

IDENTIFICAZIONE DELL'OGGETTO

Organizzazione in più fasi di ricerca attraverso
binding statico o name server o broadcast

Prima accesso al server

Poi identificazione dell'oggetto



Uso di **cache**

NOMI DI UTENTE

che vengono mappati in nomi di sistema

PARTI (anche non tutte presenti)

- 1) sintattica unica (anche gerarchica)
 - 2) con attributi espliciti
 - 3) con attributi impliciti (contesto)
- 1 e 2 opzionali*

LOCUS

completa trasparenza: uso di nomi come in sistema unico di direttorio con aggiunta di direttorio di default
Il nome di file viene espanso in *multiple file container*

PROFILE

oggetti identificati da un insieme di attributi
vedi anche direttorio di X.500 di OSI

ANSA

uso di name server per ritrovare i servizi

V-kernel

uso di tre livelli di nomi:

{stringhe, Obj_ID, Entity_ID}

Un contesto interpreta le stringhe

Ogni contesto usa nomi, che sono resi globali dal nome del gestore

Ogni gestore appartiene ad un gruppo di gestori (con un unico nome)

ROUTE e ROUTING

cammino dal sorgente al destinatario e comprende una lista di nomi intermedi

Necessità di mapping efficiente

ROUTING

uso di algoritmi di routing

proprietà

- correttezza
- semplicità
- robustezza (tolleranza ai guasti e variazioni)
- stabilità della soluzione
- ottimalità
- fairness (giustizia)

ROUTING

GLOBALI	/	LOCALI (ISOLATI)
STATICI	/	DINAMICI
ADATTATIVO	/	NON ADATTATIVO
non deterministico		deterministico

attributi

- chi prende le decisioni di routing
- chi attua le decisioni di routing
- il tempo delle decisioni di routing
- controllo del routing adattativo

Chi prende le decisioni di routing

decisioni prese:

al sorgente (source)

specifica l'intero cammino

hop-by-hop decisione ad ogni passo

il sorgente non conosce il cammino

broadcast

ogni oggetto riceve (costoso)

Chi attua le decisioni di routing

decisioni attuate da agenti

centralizzato

gestore unico del routing (ottimale)

distribuito

non esiste un unico luogo di controllo

locali (isolati)

con scambio di informazioni

parzialmente distribuito (intermedia)

si propagano solo le decisioni di variazione

Tempo delle decisioni di routing

statico vs dinamico

statico o fisso o deterministico

algoritmo non cambia

dinamico o **adattativo**

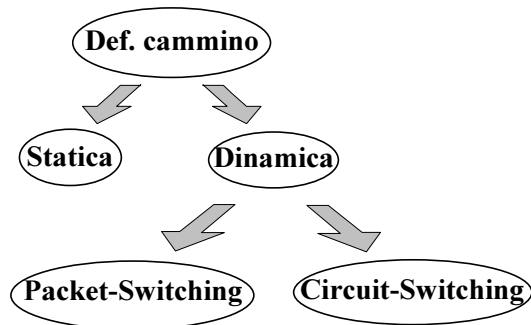
le informazioni di routing sono costantemente adeguate in base allo stato del sistema

Controllo routing adattativo

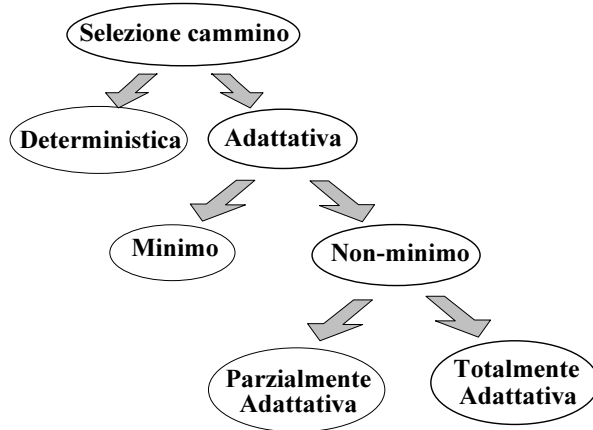
interazione con le risorse impegnate

ROUTING

**STABILIRE
IL CAMMINO
DA SEGUIRE**

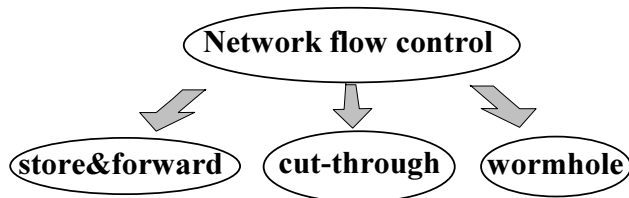


**SCEGLIERE
IL CAMMINO
da seguire
tra i possibili
cammini**



(anche non
minimi)
MISROUTING

**CONTROLLO
DI FLUSSO**



Tecniche di comunicazione

circuit-switching, packet-switching (store&forward)
virtual cut-through (**wormhole**)

TCP/IP

In IP il routing

strategia dinamica e adattativa

le informazioni di routing sono adeguate allo stato del sistema e dipendono da questo
limitando la intrusione

decisioni distribuite

router stabiliscono la strategia
senza un luogo di controllo

con scambio di informazioni in caso di variazione

il numero di partecipanti alla decisione viene tenuto limitato da *organizzazione gerarchica*

routing

hop-by-hop

decisione ad ogni router con la tabella locale

broadcast

limitato ad una rete (sottorete)

al sorgente (source route)

in caso di monitoring e per i sistemisti/router

Algoritmi di ROUTING

Statici	non adattativi	deterministici
Dinamici	adattativi	non deterministici

Algoritmi GLOBALI

in cui si propagano globalmente delle informazioni
(scarsa scalabilità per variazione)

Algoritmo **SHORTEST PATH FIRST** Dijkstra

Ogni nodo costruisce un grafo completo dell'intera interconnessione, stabilendo una metrica di distanza in base a pesi

Con successive iterazioni, si calcolano le distanze minime per ogni nodo

Tutto il traffico di routing segue il cammino più corto determinato

problemi

propagazione tabelle

eventuale uso di tutte le risorse esistenti

Implementazioni

Spanning tree

albero di interconnessione tra tutti i nodi con eliminazione dei cicli

Distance Vector

si mantiene la sola distanza da ogni nodo
problema nella propagazione delle variazioni

Link State (broadcast della variazioni)

CAMMINI MULTIPLI

Algoritmo **MULTIPATH**

Ogni nodo mantiene una tabella propria, con più possibilità per ogni destinazione, considerando **più cammini possibili** per la route tra due nodi
Scelta *random* (probabilistica) del cammino, partendo dai cammini determinati per primi

Bilanciamento del traffico di routing

Reliability in caso di guasti (anche multipli)

Algoritmo **BACKWARD LEARNING**

Ogni messaggio porta l'indicazione del mittente e, quindi, consente di inferire la **distanza** del mittente stesso ad ogni passaggio

I nodi intermedi possono stimare la distanza

Fase di apprendimento dell'algoritmo deve lavorare in base ad una politica, da cui dipendono le stime successive

Conoscenza della topologia della interconnessione completa (globale)

==>

SI POSSONO OTTENERE RISULTATI OTTIMI

non si incorre in cicli o livelock

Algoritmi globali costosi in ambiente dinamico

Dinamicità

Algoritmi isolati ed adattativi

*indipendenti dalla topologia di interconnessione
che si basano su informazioni solo locali (isolati)
o solo di vicinato (distribuiti e locali)*

MANCANO COSTOSE FASI DI COORDINAMENTO

limitato overhead per variazioni, cammini diversi

PROBLEMI PER LA PERDITA DI VISIBILITÀ

possibili cicli o livelock

Algoritmo PATATA BOLLENTE

Un messaggio viene smistato (se non è arrivato)
attraverso la coda di uscita **più scarica** del nodo
*Non si può predire il numero di passi per arrivare a
destinazione e dipende dal traffico*

Nel caso si conosca la topologia, si possono sovrapporre
anche informazioni di direzione (ad es. mesh)
*Si noti che un **algoritmo isolato adattativo** trova il
ricevente anche se questo si muove!*

Algoritmo FLOODING

Un messaggio viene smistato (se non è arrivato)
attraverso tutte le code di uscita del nodo (direzione)
*Uso di contatori per limitare i passi di un messaggio
Uso di identificatori per evitare generazione senza fine*

Algoritmo RANDOM

Ogni messaggio viene smistato, se non a
destinazione, usando una coda di uscita scelta a
caso, a parte quella di arrivo

teorema per MP-RAM

*Algoritmo ottimale in un sistema dinamico con un
numero molto elevato di nodi è una combinazione del
random (M mittente D destinatario)*

si determina in modo random un nodo **R** (diverso da
M e **D**) e si manda il messaggio con cammino
determinato in modo random da M ad R e da R a D

Naturalmente, in caso molto 'dinamico' in cui il ricevente
si sposta, qualunque algoritmo può fallire il proprio
obiettivo

In sistemi **molto dinamici e a rapida variazione**, ma
con **indicazioni di località** per le entità da raggiungere,
si cercano di determinare direzioni di **orientamento o
polarizzazione** che possano orientare le decisioni
non informate attuate da algoritmi locali

Problemi fondamentali

Congestione

entità diverse che devono predisporre risorse
necessità di controllare gli asincronismi

Controllo dei buffer

- si scartano i messaggi successivi
- si inviano indicazioni al mittente (choke)
- si prevede un numero massimo fisso di messaggi circolanti

Deadlock

impegno totale dei buffer con blocco

Soluzioni al deadlock

avoidance

si numerano i buffer in modo statico

prevention

si mantengono buffer per fare scambi in caso di saturazione

recovery

Livelock

messaggi che continuano a permanere nel sistema senza giungere a destinazione

Soluzioni

a priori

si mantiene il percorso e si evitano i loop

a posteriori

si elimina il messaggio oltre un certo numero di passi

Sistemi ad oggetti

Un insieme di **oggetti che interagiscono** per fornire per **più di una applicazione**

eterogeneità

oggetti una risposta alla diversità di piattaforme, di sistemi operativi, etc.

information hiding

autonomia

oggetti una risposta alla diversità di realizzazione di servizi ed alla raggiungibilità di servizi diversi

interfaccia determinata

apertura

oggetti una risposta alle variazioni di comportamenti

dinamicità

Progetti per la gestione di

Sistemi distribuiti aperti

Al momento sono disponibili modi molto diversi per operare a diversi livelli

OLE /COM **Microsoft**

OpenDoc **IBM DSOM, Apple Event**

DOE **SUN**

DOMF **HP**

ORBIX **IONA**

Ogni proprietario vuole il 'proprio' meccanismo

Sistemi ad oggetti APERTI

La possibilità di realizzare relazioni
cliente/servitore nel distribuito
attraverso metodologie Object-Oriented

Obiettivo

***tutte le risorse** (oggetti o meno) devono essere a disposizione di tutti gli utilizzatori, **comunque** siano stati specificate e **da chiunque** siano rese disponibili*

PROGETTI in questa direzione

Open Software Foundation ==>

Distributed Computing Environment
RPC

ANSA

Ansaware - Ambiente distribuito e standard

BBN

CHRONUS - Ambiente di servizi
con ereditarietà

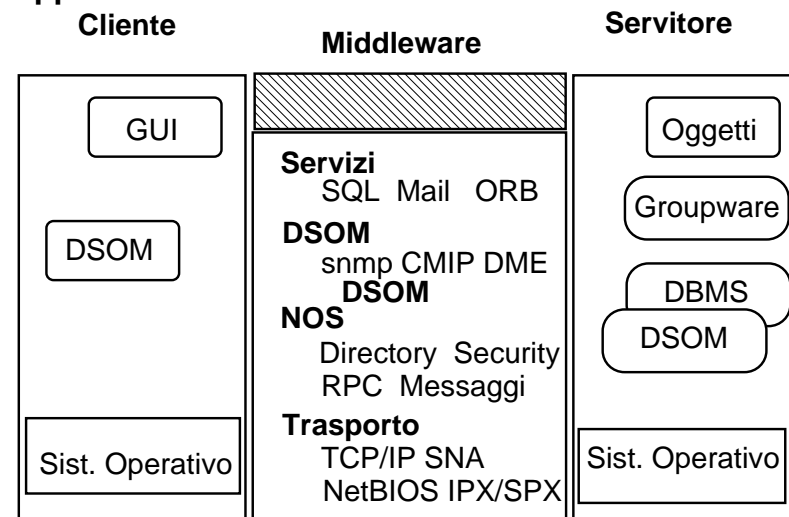
CORBA

ampio standard in fase di accettazione

Possibile infrastruttura

per la interazione tra ambienti diversi

approccio a middleware



per la interazione tra sistemi ad oggetti diversi approccio ad-hoc

molti metodi diversi

- wrapper
- protocollo comune
- funzioni di conversione custom
- conversione in formato generico

Si vorrebbero approcci **trasparenti e completi**

OSF DME

DISTRIBUTED MANAGEMENT ENVIROMENT

Open Software Foundation
per ovviare alla mancanza di

consistenza

scalabilità

interoperabilità

stabilisce un comitato per l'ambiente di programmazione

- Protocolli di **management**
snmp, CMIP, RPC di DCE
- Livello di **servizi di oggetti**
registrazione di oggetti e allocazione
altri servizi custom
- Servizi **applicativi**
installazione, management, stampe, etc.
- Protocolli di **interfaccia utente**
interfacce grafiche

Implementazione iniziata nel 1990

Distributed Object management system

insieme di **nodi clienti** e **nodi servitori**
risorse comuni al sistema distribuito completo

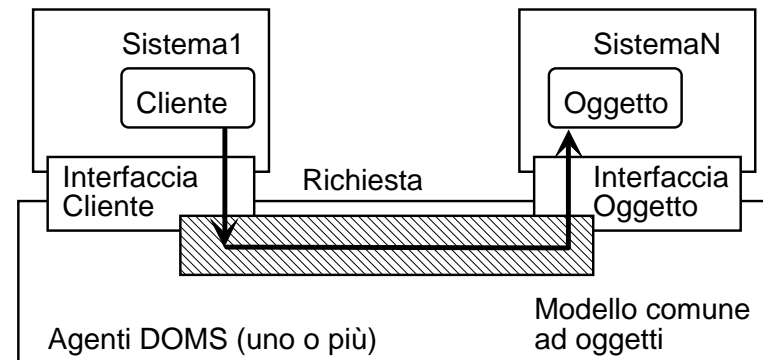
Uso di manager di **oggetti distribuiti**

che rendono i servizi attraverso l'interfaccia

I servitori forniscono gli oggetti per il servizio

Almeno:

- **implementazione** degli oggetti
- **interfacce** degli oggetti per i **clienti**
- strumenti di **scambio messaggi**
- **interfacce** degli oggetti
- **risorse distribuite** di calcolo



STANDARD

SISTEMI APERTI AD OGGETTI

modelli ad **oggetti diversi** con
possibilità di **interazione reciproca**

- definizione di una **interfaccia** per oggetto
- definizione della **interazione** tra sistemi diversi
- **bus ad oggetti** per integrazione

OMG-CORBA

OBJECT MANAGEMENT GROUP

creato nel 1989 con **440 aziende**
Microsoft, Digital, HP, NCR, SUN, OSF, *etc.*

con l'obiettivo di creare un **sistema di gestione** di una
architettura distribuita

CORBA STANDARD

Common Object Request Broker Architecture

v1 ==> 1991, v2 ==> 1992

Orbix SunOS Solaris, Iris, Windows NT, HP/UX,
AIX, OSF/1, UnixWare

DSOM IBM

conflitto

possibili concorrenti

- **COM (Common Object Model)**
proposto da **Digital, Microsoft**
- **Distributed Bus**
proposto da **Novell**

COM (Digital, Microsoft)

uso di **OLE**

ma in generale **non** di un **modello ad oggetti**

CARATTERISTICHE

- non presenta il concetto di oggetto per incapsulare le proprietà (gli oggetti DCOM non hanno stato)
- non ereditarietà tra specifiche
- IDL non compliant né con CORBA né con DCE (Distributed Computing Environment)
- no identificatori unici per gli oggetti
- uso di eventi per comandare le azioni
- necessità di definire le interconnessioni tra oggetti con **delegazione, aggregazione, composizione**

**ci si avvia al superamento dei limiti attraverso
OLE integration**

ENTITÀ

da mettere in relazione

cliente e implementazione di un oggetto per il servizio

oggetto

entità che fornisce servizi

richiesta

meccanismo per manifestare esigenza di un servizio

tipo

entità per la classificazione degli oggetti

interfaccia

descrizione delle operazioni possibili per un insieme di oggetti

operazione

entità con nome che può essere richiesta ad un oggetto

in un contesto comune (CORBA)

Common Object Request Broker Architecture

Il cuore è il gestore dei nomi (**broker**) che consente i collegamenti

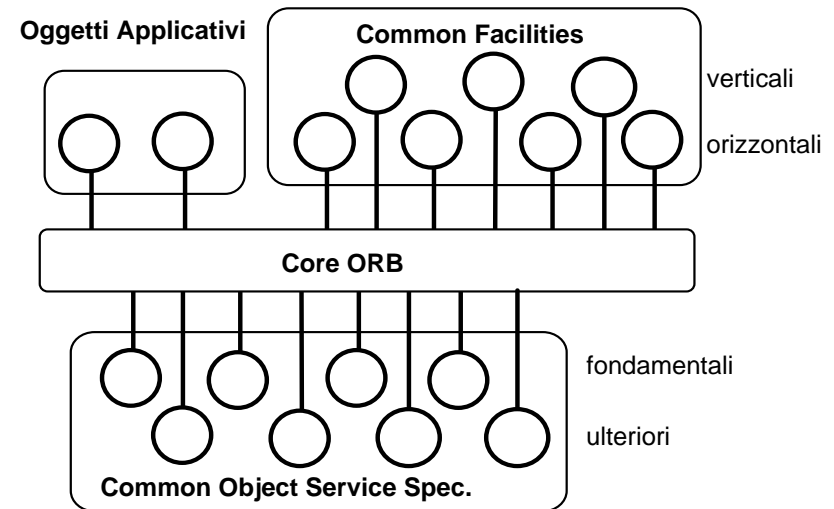
in modo **statico e dinamico**

Object Request Broker (ORB)

controllo **allocazione** e **visibilità** di oggetti
controllo dei **metodi** e della **comunicazione**

Struttura ORBA

ORB come un bus di interconnessione



Sulla via della standardizzazione

si deve standardizzare l'intero ambiente di interconnessione pensando alle specifiche e a come si possono realizzare

Ogni componente può legarsi agli altri, anche non noti, usando il servizio di uno o più ORB (anche non noti a priori)

Object Management Architecture OMA



Common Object Service Specification COSS

1) operazioni fondamentali per oggetti

naming, tempo di vita, eventi, persistenza

2) operazioni ulteriori (o servizi)

relazionali, verso esterno

transazioni, controllo concorrenza

COMMON FACILITIES CF

1) facility orizzontali

Interfaccia utente,

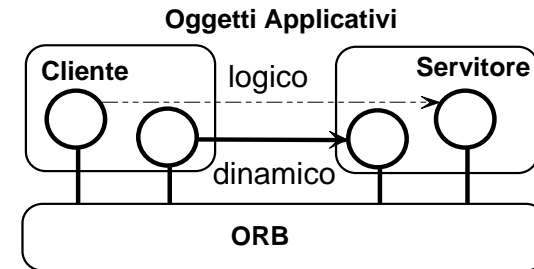
Management Sistema, Informazioni, Task

1) facility verticali

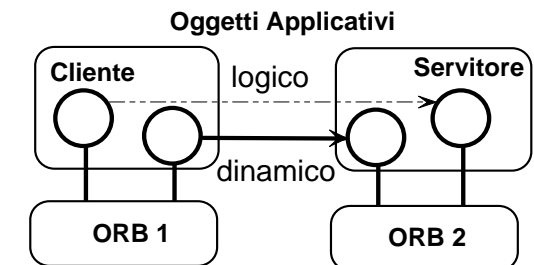
oggetti speciali dedicati

CORBA

Con lo standard anche interoperabilità per il flusso di informazioni tra ambienti eterogenei (uno stesso ORB)



Naturalmente si vogliono anche considerare le possibilità di coordinamento di ORB diversi



Definizione (dalla versione 2) di **Protocolli InterORB**
protocollo per interoperabilità ORB-to-ORB
General Inter-ORB Protocol (GIOP)

Internet Inter-ORB Protocol (IIOP)

interoperabilità per la rete Internet su TCP/I

CORBA

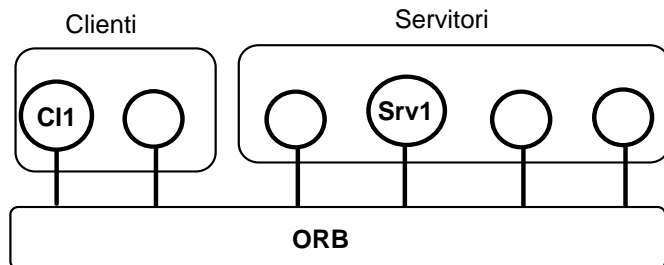
I componenti di CORBA sono

- *Object Request Broker* (ORB)
- *Interface Definition Language* (IDL)
- *Basic Object Adapter* (BOA)
- *Static Invocation Interface* (SII)
- *Dynamic Invocation Interface* (DII)
- *Object Repository* (OR)

Object Request Broker (ORB) deve **coordinare la invocazione di servizi locali e remoti**

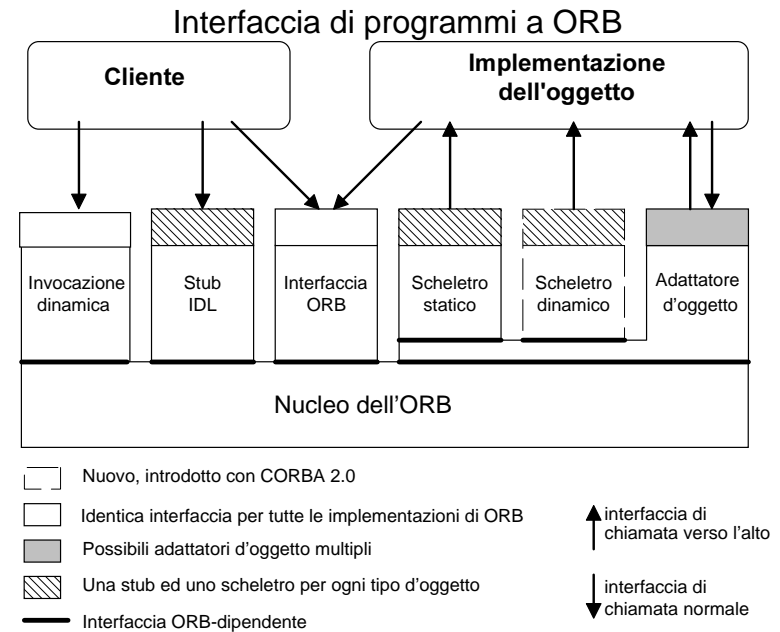
- individuare l'implementazione di un oggetto per fare da servitore ad una richiesta (*object location*)
- preparare il servitore a ricevere la richiesta (*object creation, activation & management*)
- trasferire la richiesta dal cliente al servitore
- restituire la risposta al cliente

Oggetti Applicativi



CORBA

CORBA (Common Object Request Broker Architecture) standard per garantire interoperabilità e portabilità di sistemi object-oriented nel distribuito



Accesso agli oggetti attraverso l'**adattatore** per filtrare e adattare modi di richiesta diversi

CORBA Interface Definition Language

Aggancio con un linguaggio di specifica della interfaccia che consente **binding statico** e **dinamico**

Realizzazione del mapping su linguaggi diversi con proprietà diverse

Binding

Conoscenza degli oggetti sia **statica**, sia **dinamica** con costi diversi

Binding statico

basso costo

uso di stub opportuni e di livello di RPC
le decisioni sono prese prima della esecuzione

Binding dinamico

elevato costo

Ogni richiesta viene passata all'ORB
il messaggio viene passato dall'**ORB** all'**adattatore**
dell'oggetto
Modalità **sincrone** ed **asincrone** di richiesta

Il binding dinamico è possibile tramite il **servizio** di **naming**
detto **object repository**

per ogni richiesta, il cliente trova le **interfacce** per
soddisfarla e può rivolgersi in modo dinamico all'ORB
per ritrovare diverse **implementazioni** del servizio

Object repository può contenere informazioni addizionali,
per **debugging**, **facility**, etc.

Le possibilità di adeguarsi dipendono dai servizi e dai
metodi che si vogliono fornire

CORBA Interface Definition Language

Aggancio con un linguaggio di specifica della interfaccia
che consente **binding statico** e **dinamico**

*Realizzazione del mapping su linguaggi diversi con
proprietà diverse*

INTERFACE DEFINITION LANGUAGE (IDL)

Per garantire flessibilità nella distribuzione su piattaforme
eterogenee

UNIX vs. Windows NT

C++ vs. Java

si usano linguaggi dichiarativi

Interface Definition Language

Molti IDL diffusi sono procedurali

- OSI **ASN.1** / **GMDO**
- ONC **XDR** (SUN RPC)
- **OMG IDL**
- Microsoft **ODL**

CORBA IDL

è un linguaggio **object-oriented**
(*derivato dal C++*)

- **interfacce** come insiemi di *metodi* ed *attributi*
- distinzione tra *definizione* e *implementazione*
- *ereditarietà multipla* delle interfacce
- definizione *eccezioni*
- gestione automatica degli *attributi*
- *mappaggi* per linguaggi diversi ed ambienti diversi

Il compilatore può ottenere automaticamente *stub* per
clienti/servitori anche usando linguaggi diversi


```

module Stock
{exception Invalid_Stock {};
exception Invalid_Index {};
const length = 100;

interface Quoter
{ attribute float quote;
readonly attribute float quotation;
long get_quote (in string stock_name )
raises (Invalid_Stock) ;
};

interface SpecialQuoter: Quoter
{ attribute float quotehistory [length];
readonly int index [length];
long get_next (in string stock_name )
raises (Invalid_Index) ;
long get_first (in string stock_name )
raises (Invalid_Index) ;
};
}

```

Per ogni attributo si mettono a disposizione le funzioni di accesso

```

attribute float quote;
float _get_quote ();
void _set_quote (in float q);
readonly attribute ind index;
float _get_index ();

```

Per ogni eccezione, lo stato completion_status dà informazioni

```

COMPLETED_YES,
COMPLETED_NO,
COMPLETED_MAYBE

```

INVOCATION INTERFACE

Static Invocation Interface (SII)

Il compilatore e gli strumenti risolvono

le chiamate prima della esecuzione

Tutte le invocazioni sono controllate in anticipo e sicure

nessun controllo dinamico

Dynamic Invocation Interface (DII)

necessario per inviare le richieste senza predefinitone

cioè ad oggetti che non esistono alla compilazione

comportamento DINAMICO

- il cliente ottiene a run-time dall'object repository una struttura che descrive l'interfaccia dell'oggetto

Uso di API per la invocazione dinamica

```

interface Request { // Pseudo IDL
    Status add_arg ( // argomento
        in Identifier name, // nome
        in TypeCode arg_type, // tipo
        in void *value, // valore
        in long len, // lunghezza
        in Flag arg_flag // flag
    );
    Status invoke ( in Flags invoke_flags // flag invocazione);
    Status delete ();
    Status send ( in Flags invoke_flags // flag invocazione);
    Status get_response ( in Flags response_flags
        // flag risposta);
}

```

La richiesta viene creata dall'Oggetto con tutte le informazioni di contesto necessarie

Necessità di identificatori tra ambienti diversi

CORBA 1.2 non sono previsti nomi unici

CORBA 2 supporto per nomi unici

Object Reference

Nomi *non unici* associati ad un servizio

ObjectRef possono essere passate da un ambiente ad un altro ed essere utilizzate dovunque

Sistema di nomi che trasla in nomi esterni

Un **ObjectRef** al passaggio viene convertito in un proxy nell'ambiente del ricevente per potere essere utilizzato

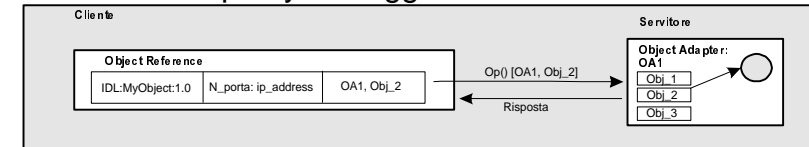
esempio

```
interface Consumatore_Evento
{ oneway void ricevi_evento
  (in Event evento);
  // un evento si è verificato
  oneway void disconnetti (in string motivo);
  // Disconnetti a causa di <motivo>
  // invocata dal Notificatore
};
```

```
interface Notificatore_Evento
{ oneway void invia_evento
  (in Event evento);
  // manda la notifica a tutti i sottoscrittori
  oneway void subscribe
  (in Consumatore_Evento notifica,
   in string criteriodifiltro) ;
  // connessione da parte del consumatore
  // per ricevere indicazioni con filtraggio
  oneway void unsubscribe
  (in Consumatore_Evento notifica);
  // disconnessione da parte del consumatore
};
```

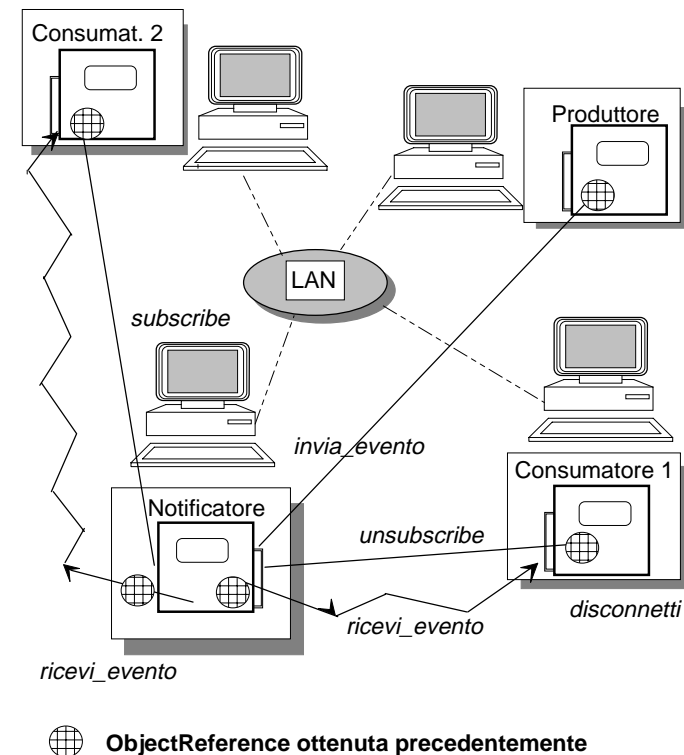
Object Reference

inserita come proxy dell'oggetto interessato



La conoscenza si amplia attraverso le invocazioni

e si mantengono **ObjectReference** agli oggetti non locali



Questi riferimenti possono essere passati da un nodo ad un altro e devono mantenersi validi

Object Adapter (OA)

per provvedere il binding tra objectRef e oggetti associati

Possono esserci più adattatori

Basic Object Adapter (BOA)

Portable Object Adapter (POA)

Object-Oriented Database Adapter (OODA)

...

Basic Object Adapter (BOA)

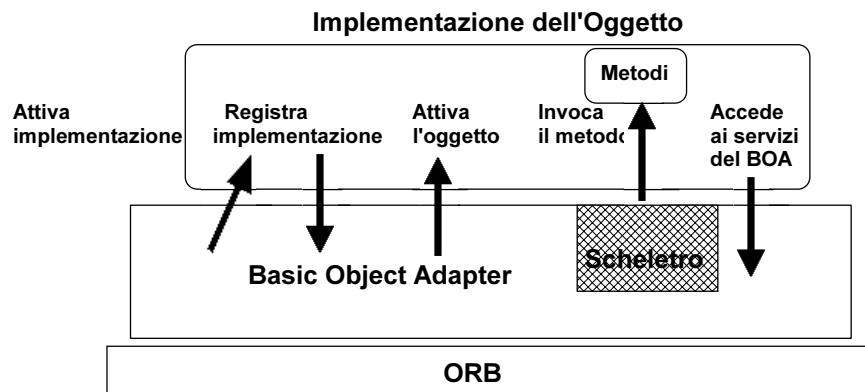
Interfaccia usata per accedere alle funzionalità di ORB

Esporta operazioni per:

- creare i riferimenti agli oggetti
- registrare e attivare le implementazioni degli oggetti
- autenticare le richieste

L'oggetto deve registrarsi per potere essere attivato dall'ORB

Sono possibili molti adattatori diversi secondo le specifiche necessità POA



ATTIVAZIONE

Il cliente deve essere legato al server

binding al server e invocazione di un metodo proxy

Il server deve essere attivato

e potere decidere come servire le richieste

*il server viene o lanciato inizialmente
o lanciato su necessità (e persiste)*

⊖ costo di attivazione del processo

MODI CONCORRENTI DI ATTIVAZIONE NEL SERVER

attivazione per ogni richiesta (thread_per_request)

un processo viene creato nell'oggetto per servizio

⊖ costo di generazione del processo

attivazione iniziale di un pool (pool di thread)

ogni oggetto riceve il suo processo da un pool creato inizialmente senza il costo della creazione

⊖ problemi in caso di traffico elevato

attivazione per sessione

ogni cliente ha un processo dedicato alla interazione

⊖ non si bilancia il carico

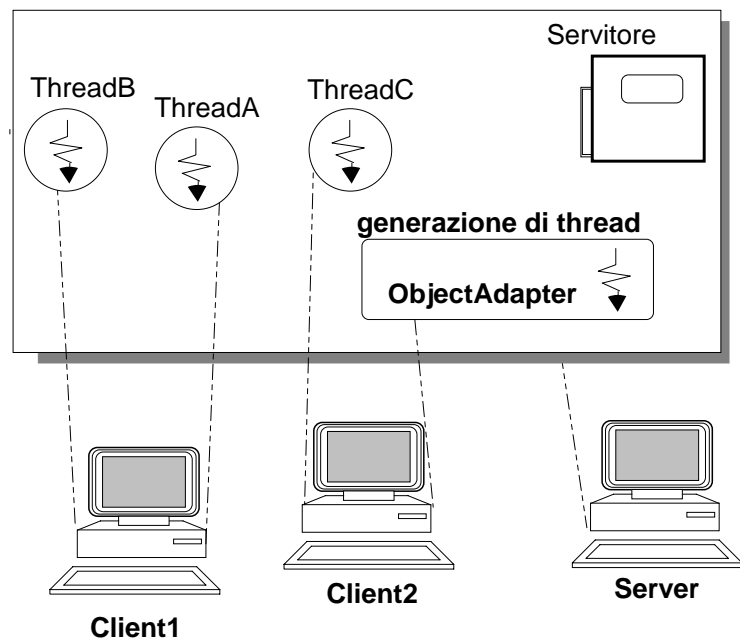
Si pensino anche altre modalità

Ad esempio:

avere una **sol**a attivazione (capacità di risposta)
possibile per più oggetti server (**shared server**)
contemporaneamente

THREAD-PER-REQUEST

Si genera un thread per ogni richiesta di servizio

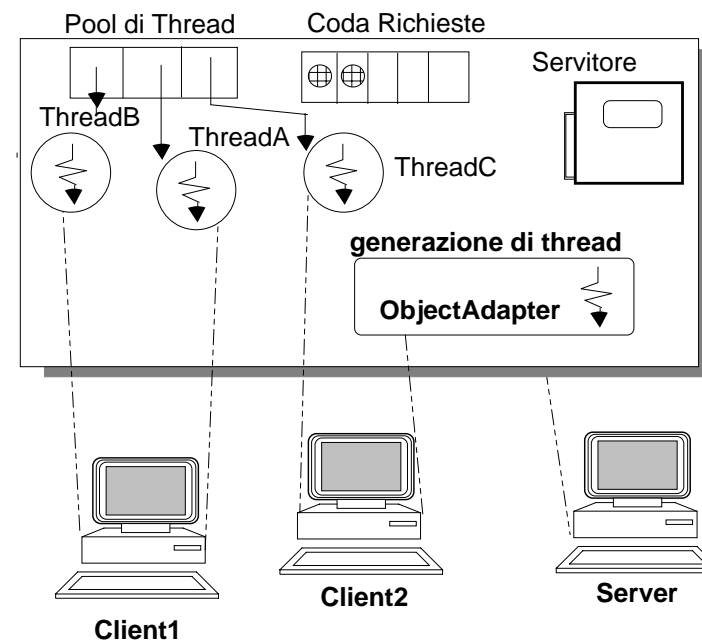


 Capacità di Esecuzione

Costo della generazione di un processo per ogni richiesta

THREAD-POOL

Si genera un pool di thread per le richieste di servizio



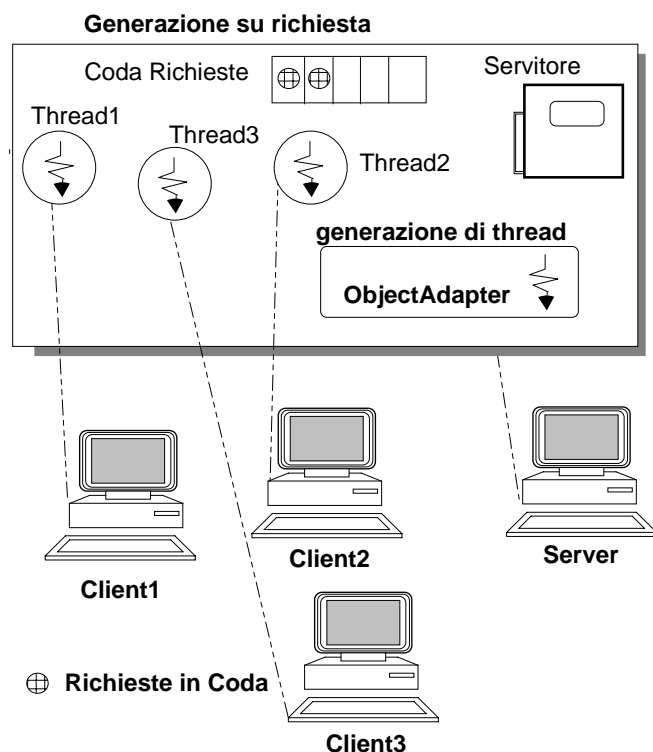
 Capacità di Esecuzione  Richieste in Coda

In questo caso, è necessario disporre di coda per le richieste e di aggancio alle richieste dei thread esistenti e disponibili

In caso non ci siano task liberi, o si attende o si prevede una ulteriore generazione

THREAD-PER SESSIONE

Si genera un thread per ogni cliente e questo si incarica di rispondere a tutte le richieste di quel cliente (una sola richiesta per volta per la stessa sessione)



In questo caso, il thread è unico e viene mantenuto per la intera sessione
Si limita il parallelismo per ogni cliente

Repository di interfacce e implementazioni

Funziona da gestore dei nomi del sistema
informazioni sugli oggetti disponibili

- informazioni di interfaccia per invocazioni dinamiche, controlli, ispezioni run-time ...
- informazioni di implementazioni per localizzare e attivare gli oggetti

Organizzazione delle tabelle interne tramite il riconoscimento di contenuto (**contained**) e di contenitore (**container**)

Le operazioni elementari sono del tipo:

```
interface Repository: Container {
    Contained lookup_id ( // lista
                        in RepositoryId search_id)
}

interface Contained {
    struct Description { Identifier name; any value; }
    attribute Identifier name;
    attribute RepositoryId id;
    attribute RepositoryId defined_in;
    ContainerSeq within ();
    Description describe ();
}
```

Uno stesso ORB può avere accesso a Repository diversi

disponibilità strumenti compliant

Molto ampia ed in crescita

Object Broker	DEC
ORB	HP
DSOM	IBM
Orbix	IONA
(DOM Facility) DOE	Sun
PowerBroker	ExperSoft

CURVA DI APPRENDIMENTO

complessità del modello ad oggetti

per rispondere alla organizzazione di

soluzioni a sistemi complessi

nuovi concetti

nuovi strumenti

nuovi costrutti

In genere è necessario spendere il tutto
su un insieme di progetti
per ottenere un vantaggio consistente

PERFORMANCE

Al momento le realizzazioni possono introdurre overhead
*Man mano che gli strumenti si affinano, l'obiettivo di avere
soluzioni altrettanto efficienti quanto la codifica a mano
diventa sempre più raggiungibile*

CORBA <==> JAVA

. rappresentano ambiti diversi

Java: programming technology + mobile code system;
CORBA: integration technology...

. ...ma si integrano in maniera ottimale

Java *implementation transparency*
linguaggio ed ambiente unico per la
programmazione su macchine eterogenee
con **applet** aggancio al WEB
con **beans** componenti grafici indipendenti

CORBA *network transparency*
standard per la interconnessione di ambienti
diversi per lo scambio di servizi su
macchine eterogenee
realizzazione dell'ORB

	Java	CORBA
efficienza	unica macchina virtuale	diversi ambienti OO
integrazione	facile	uso dello standard