



**Università degli Studi di Bologna
Facoltà di Ingegneria**

**Corso di
Reti di Calcolatori L-A**

Sistemi RPC e sistemi di Nomi

Antonio Corradi

Anno accademico 2009/2010

REMOTE PROCEDURE CALL - RPC

REMOTE PROCEDURE CALL (RPC) come estensione del normale meccanismo di chiamata a procedura locale, come **cliente/servitore nel distribuito**

Approccio applicativo di alto livello

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

Differenze rispetto alla chiamata a procedura locale

- sono coinvolti **processi distinti** su nodi diversi
- i processi **cliente e servitore** hanno vita separata
- i processi **non** condividono lo **spazio di indirizzamento**
- sono possibili **malfunzionamenti** sui **nodi** o nella **infrastruttura di comunicazione**

RPC: PROPRIETÀ

Remote Procedure Call

- consente il controllo dinamico del tipo dei parametri e del risultato
- include il trattamento dei parametri di ingresso / uscita dal cliente al servitore (e viceversa) detto marshalling o almeno serializzazione
(livello di presentazione: marshalling)

PROPRIETÀ - uniformità totale impossibile (visibilità guasti)

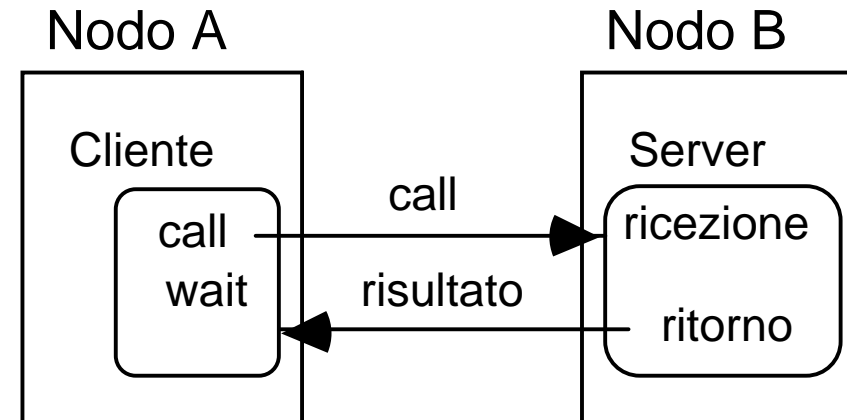
- **trasparenza** approccio locale e remoto
- **type checking** e **parametrizzazione**
lo stesso controllo di tipo e dei parametri
- **controllo concorrenza** e **eccezioni**
- **binding distribuito**
- possibile **trattamento degli orfani**
orfano il processo server che non riesce a fornire risultato
e anche il cliente senza risposta

RPC – STORIA

Prima sistemazione dovuta a **Birrel e Nelson** (1984)

partendo da quanto usato in Xerox, Spice, Sun, HP, etc.

Molte implementazioni



Progetto Athena MIT, ITC CMU, ...

RPC usando scambi di messaggi coordinati

CLIENTE

send

wait

SERVITORE

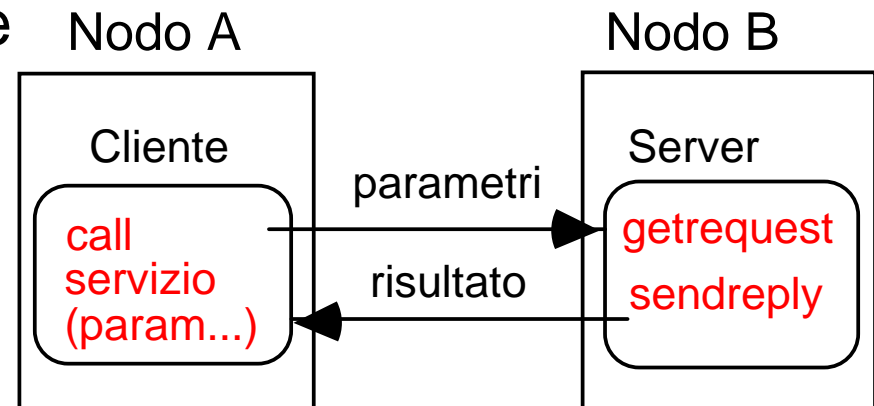
get-request <operazione>

send-reply

si devono anche prevedere trasformazioni di dati, nomi, controllo, ...

RPC – PRIMITIVE

Ogni sistema usa primitive diverse
senza troppe regole standard



dalla parte del cliente

call servizio (**servitore**, **argomenti**, **risultato**)

dalla parte del servitore

Due primitive di **accettazione** e di **risposta**
con **due** possibilità di **concorrenza sul server**:

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

RPC – TOLLERANZA AI GUASTI

Obiettivo applicativo è mascherare i malfunzionamenti

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

Ad esempio, in caso di crash del servitore prima di avere fornito la risposta o in caso di sua non presenza, o in caso di perdita di messaggio in andata o ritorno, ...

il cliente può tentare politiche diverse e diversi comportamenti:

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

Spesso si assume che le operazioni siano idempotenti ossia che si possano eseguire un numero qualunque di volte con lo stesso esito per il sistema cliente/servitore (?)

FAULT TOLERANCE e SEMANTICA

Le RPC hanno alcune semantica tipiche e strategie relative

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

Invece, in caso di crash del cliente, si devono trattare orfani sul nodo servitore, ossia i processi in attesa di consegnare risultato

Politiche tipiche

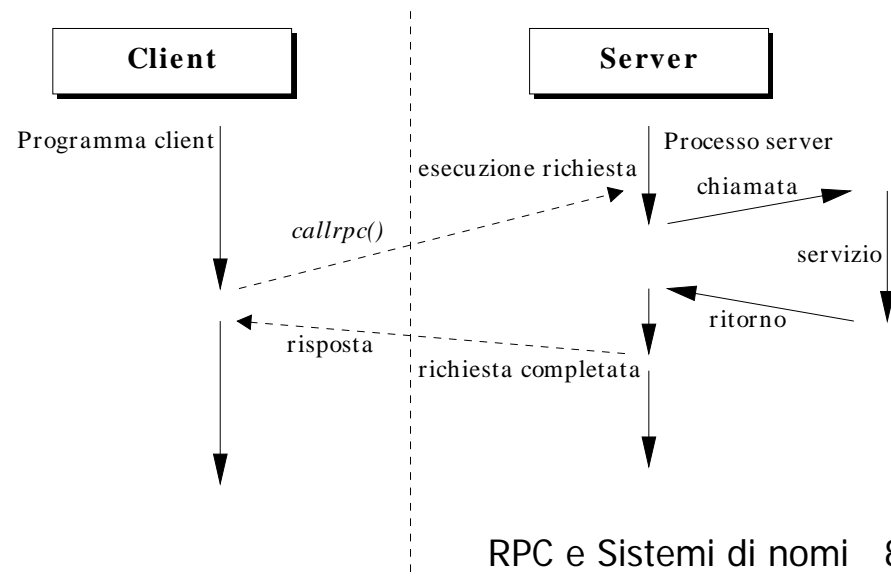
- **sterminio**: ogni orfano risultato di un crash viene distrutto
- **terminazione a tempo**: ogni calcolo ha una scadenza, oltre la quale è automaticamente abortito
- **reincarnazione (ad epoche)**: tempo diviso in epoche; tutto ciò che è relativo alla epoca precedente è obsoleto e distrutto

Open Network Computing ONC

Sun propone una chiamata RPC (diversa dalla locale)
primitiva `callrpc(...)` con parametri aggiuntivi oltre a quelli logici
nome del nodo remoto
identificatore della procedura da eseguire
specifiche di trasformazione degli argomenti (si introduce un formato standard eXternal Data Representation XDR)

Schema del modello ONC NON TRASPARENZA

adottato da
HP-UX, Sun
Sistemi UNIX compatibili,
Novell Netware



Network Computing Architecture NCA

Si introducono ai due endpoint di comunicazione, il client e il server delle **routine stub per ottenere la trasparenza**

Le chiamate diventano del tutto locali all' endpoint e stub

Schema del modello NCA

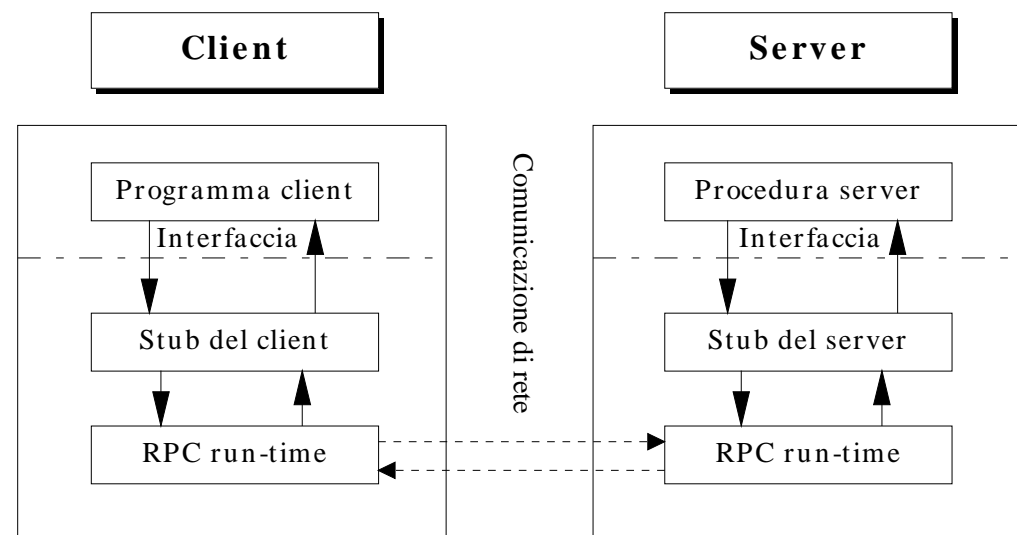
Gli **stub** sono forniti
dall'implementazione e
generati automaticamente

Le parti logiche di programma
sono "del tutto" inalterate

ci si dirige verso lo stub

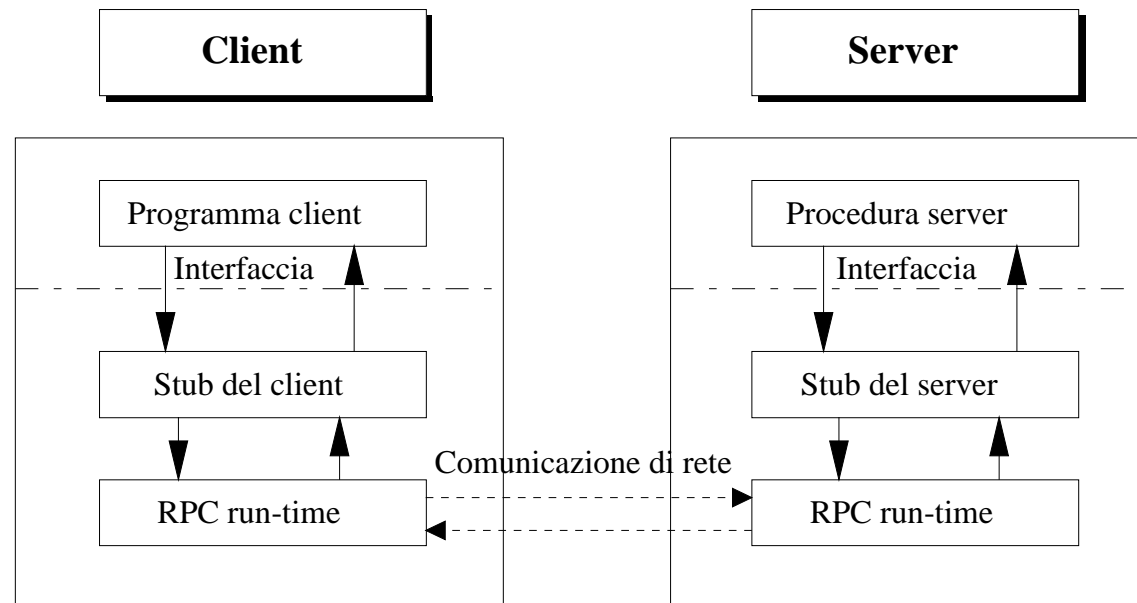
che nasconde le operazioni ⇒ **COSTO: due chiamate per RPC**

Mercury ottimizza i messaggi con stream di chiamate



MODELLO con TRASPARENZA

Nelson descrive il **modello implementativo NCA trasparente** con **uso di stub** ossia **interfacce locali per la trasparenza** che trasformano la richiesta da locale a remota



Modello **asimmetrico a molti clienti/ un solo servitore**

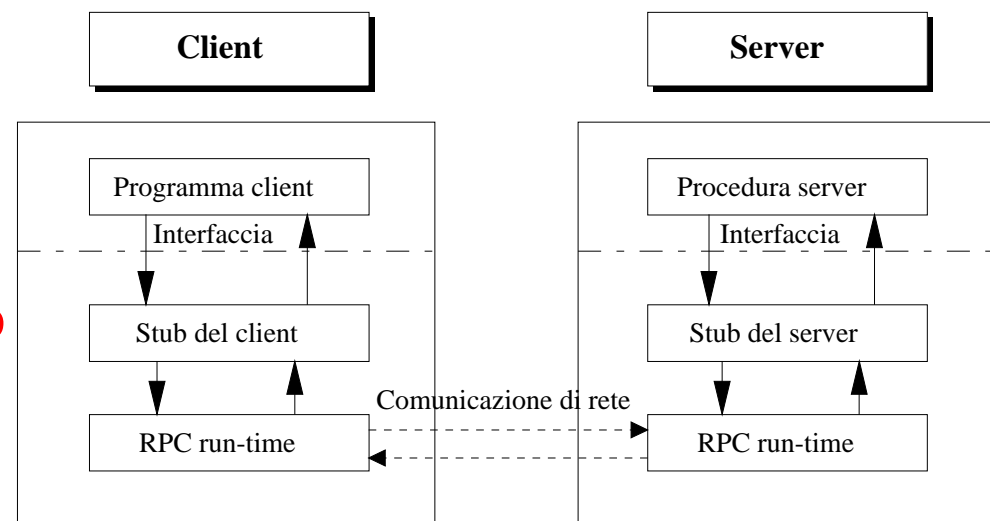
MODELLO con STUB

Modello **cliente servitore asimmetrico applicativo**

- Il cliente invoca uno stub, che si incarica di tutto: dal recupero del server, al trattamento dei parametri e dalla richiesta al supporto run-time, al 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

Lo sviluppo prevede la massima trasparenza

- **STUB prodotti in modo automatico**
- **L'utente finale progetta e si occupa solo delle reali parti applicative e logiche**



ESEMPIO di funzioni nello STUB

Negli stub si concentra tutto il **supporto nascosto all'utente finale**

operazione(parametri)

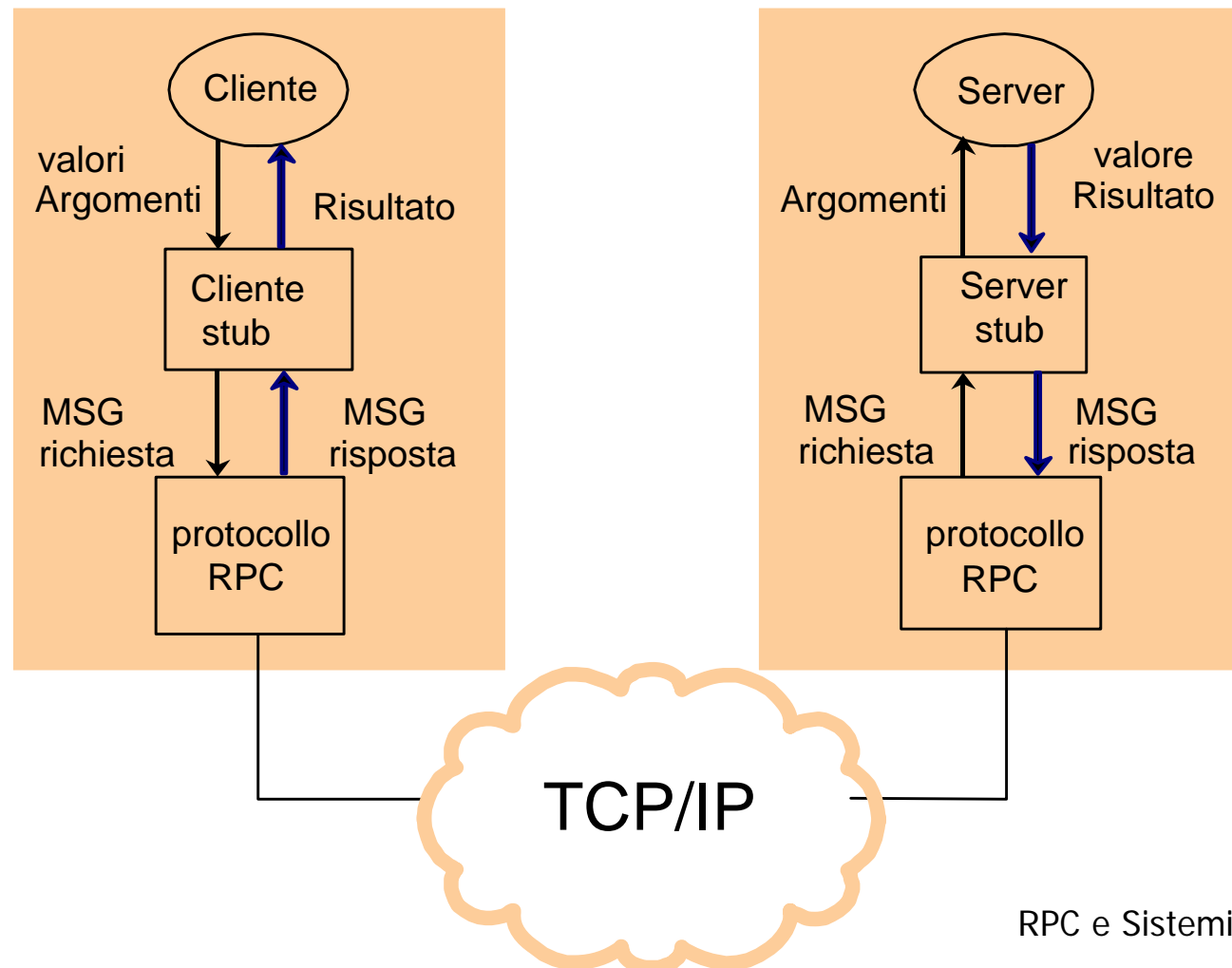
stub cliente: < ricerca del servitore > < marshalling argomenti> <send richiesta> <receive risposta> < unmarshalling risultato> restituisce risultato fine stub cliente;	stub servitore: < attesa della richiesta> < unmarshalling argomenti> invoca operazione locale ottiene risultato < marshalling del risultato> <send risultato> fine stub servitore;
--	---

operazione(parametri)

In questo schema mancano le **ritrasmissioni possibili** da parte del cliente e il **controllo della operazione** da parte del servitore (concorrenza o meno)

Ancora schema di STUB

Gli stub si occupano di tutta la **parte distribuita**



PASSAGGIO di PARAMETRI

I parametri devono passare tra ambienti diversi

passaggio per valore o per riferimento

Trattamento default dei parametri per valore

impaccamento dei parametri (marshalling) e disimpaccamento (unmarshalling) con dipendenza dal linguaggio utilizzato

Per tipi primitivi o una entità con valori privati

marshalling / unmarshalling per la presentazione

Per tipi utente costruiti e dinamici, ad esempio, una lista o un albero e la memoria dinamica (?) la logica guida la trasformazione
si deve (?) copiarla o (?) muoverla e ricostituirla sul server per poi riportarla sul nodo iniziale

In genere si favorisce il passaggio per valore

PASSAGGIO di PARAMETRI

Passaggio parametri dal cliente al servitore
nel caso di **passaggio per valore**

passaggio con trasferimento e visita (valore perso sul server)

nel caso di **passaggio per riferimento**

passaggio senza trasferimento ma rendendo l'**oggetto remoto**
uso di oggetti che rimangono nel nodo di partenza e devono essere
identificati in modo unico nell'intero sistema

Se si vuole riferire un entità del cliente, si passa il riferimento alla
stessa entità che i nodi remoti possono riferire attraverso RPC

Per esempio, un oggetto che sia già in uso sul nodo del cliente deve
potere essere riferito dal nodo servitore e cambiato in stato senza
interferire con l'uso locale dell'oggetto

TRATTAMENTO delle ECCEZIONI

Le RPC hanno previsto integrazione con exception handling

trattando gli eventi anomali dipendenti dalla distribuzione o ai guasti integrandola con la gestione locale e inserendola nella gestione locale delle eccezioni

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

Si può anche inserire l'eccezione nello scope di linguaggio

CLU (Liskov) a livello di invocazione della RPC

MESA (Cedar) a livello di messaggio

Java definizione delle exception

una RPC può produrre il servizio con successo o produrre insuccesso e determinare una eccezione locale con semantica tipica

may-be con time-out per il cliente

at-least-once SUN RPC

at-most-once con l'aiuto del trasporto

INTERFACE DEFINITION LANGUAGE

Interface Definition Language IDL

linguaggi per la descrizione delle operazioni remote, la specifica del servizio (detta firma) e la generazione degli stub

Un IDL deve consentire la identificazione non ambigua

- **identificazione (unica) del servizio tra quelli possibili**

uso di nome astratto del servizio spesso prevedendo 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 dichiarativo con ereditarietà, ambienti derivati con binder ed altre entità

Dalla definizione del linguaggio IDL fornita dall'utente si possono generare gli stub

TIPICI LINGUAGGI IDL

Sono stati definiti molti linguaggi IDL

che hanno permesso di studiare le implicazioni e le politiche a livello di sistema di supporto

Sono nati molti IDL e relativi strumenti correlati per lo sviluppo automatico di parte dei programmi direttamente dalla specifica astratta, esempi sono

SUN XDR primo esempio di standard interno

OSF DCE IDL

ANSA ANSAware

HP NCS IDL

CORBA IDL

Le differenze hanno generato dibattiti e confronti anche accesi, spentisi viste gli scope limitati dei linguaggi stessi ⇒

Mancanza di standard ☹

ESEMPIO di XDR

XDR eXternal Data Representation

Il formato XDR definisce le **operazioni remote** e tutto quello che è necessario conoscere per la generazione di stub (**parametri**)

L'utente deve sviluppare un file di descrizione logica dei servizi offerti da cui si possono generare gli stub

Si prevedono più servizi in **versioni diverse** e **tipi primitivi e anche definiti dall'utente**

file msg.x

```
program MESSAGEPROG {  
    version MESSAGEVERS {  
        int PRINTMESSAGE(string) = 1;  
    } = 1;  
} = 0x20000013;
```

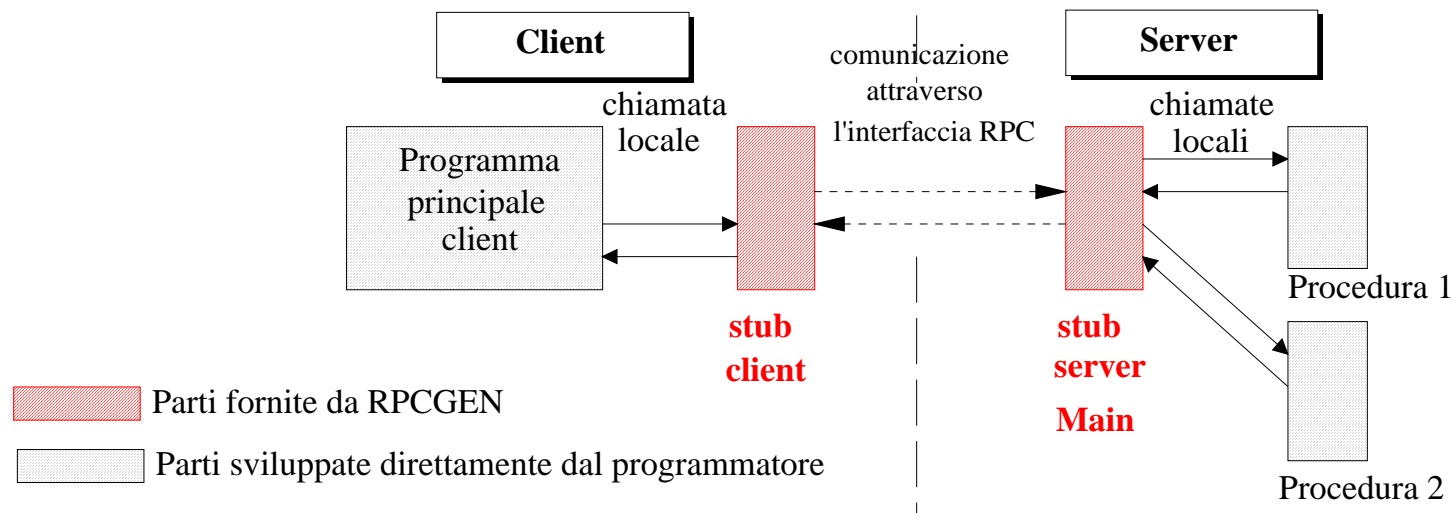
DA IDL A STUB

Gli IDL hanno lo scopo di supporto allo sviluppo dell'applicazione permettendo di generare automaticamente gli stub dalla interfaccia specificata dall'utente

Strumento RPCGEN (Remote Procedure Call Generator)

compilatore di protocollo RPC per generare procedure stub

RPCGEN produce gli stub per il server e il client da un insieme di costrutti descrittivi per tipi di dati e per le procedure remote in linguaggio RPC



SVILUPPO e SUPPORTO RPC

Nel caso di un utente che deve utilizzare le RPC, di varia tecnologia, abbiamo molte fasi, alcune facilitate e poco visibili, svolte tutte dal supporto, altre sotto il controllo utente

Dopo la specifica del contratto in IDL, ...

FASI TIPICHE della IMPLEMENTAZIONE

- compilazione di sorgenti e stub
- binding delle entità
- trasporto dei dati
- controllo della concorrenza
- supporto alla rappresentazione dei dati

Alcuni ambienti facilitano altri meno, ma tutti hanno una qualche caratterizzazione specifica

FASI di SUPPORTO RPC

compilazione di sorgenti e stub

La compilazione produce gli stub che servono a semplificare il progetto applicativo e risponde alla necessità che cliente e servitore raggiungano un accordo sul servizio da richiedere / fornire

trasporto dei dati

Il trasporto connesso e senza connessione è intrinseco allo strumento e tanto più veloce ed efficiente, tanto meglio (TCP vs. UDP)

controllo della concorrenza

Il controllo consente di usare gli stessi strumenti per funzioni diverse, con maggiore asincronicità e maggiore complessità (ripetizione, condivisione di connessione, processo)

supporto alla rappresentazione dei dati

per superare eterogeneità si trasformano i dati, tanto più veloce, tanto meglio, bilanciata con la ridondanza se ritenuta necessaria

RPC BINDING

binding delle entità

Il binding prevede come ottenere l'aggancio corretto tra i clienti e il server capace di fornire la operazione

Il binding del cliente al servitore secondo due possibili linee

scelta pessimistica e statica

La compilazione risolve ogni problema prima della esecuzione e forza un binding statico (nel distribuito) a costo limitato (ma poco flessibile)

scelta ottimistica e dinamica

Il binding dinamico ritarda la decisione alla necessità, ha costi maggiori, ma consente di dirigere le richieste sul gestore più scarico o presente in caso di sistema dinamico

BINDING STATICO vs. DINAMICO

fondamentale nella relazione tra cliente e servitore

BINDING DINAMICO RPC

Il binding dinamico tipico viene ottenuto distinguendo due fasi nella relazione cliente/servitore

- **servizio (fase statica)** prima della esecuzione

il cliente specifica a chi vuole essere connesso, con un nome unico identificativo del servizio (NAMING)

in questa fase si associano dei nomi unici di sistema alle operazioni o alle interfacce astratte e si attua il binding con la interfaccia specifica di servizio

- **indirizzamento (fase dinamica)** all'uso, durante la esecuzione

il cliente deve essere realmente collegato al servitore che fornisce il servizio al momento della invocazione (ADDRESSING)

in questa fase si cercano gli eventuali servitori pronti per il servizio (usando sistemi di nomi che sono stati introdotti proprio a tale scopo)

BINDING RPC

Parte di NAMING statica, SERVIZIO

risolto con un numero associato staticamente alla interfaccia del servizio (nomi unici)

Parte di ADDRESSING dinamica, durante l'esecuzione

La parte dinamica deve avere costi limitati ed accettabili durante il servizio

1) ESPLICITA attuata dai processi

Il cliente deve raggiungere un servitore
si può risolvere con un multicast o broadcast attendendo solo la prima risposta e non le altre

2) IMPLICITA tramite un agente esterno, un servitore di nomi

uso di un name server che registra tutti i servitori e agisce su opportune tabelle di binding ossia di nomi, prevedendo funzioni di ricerca di nomi, registrazione, aggiornamento, eliminazione

SISTEMI di NOMI

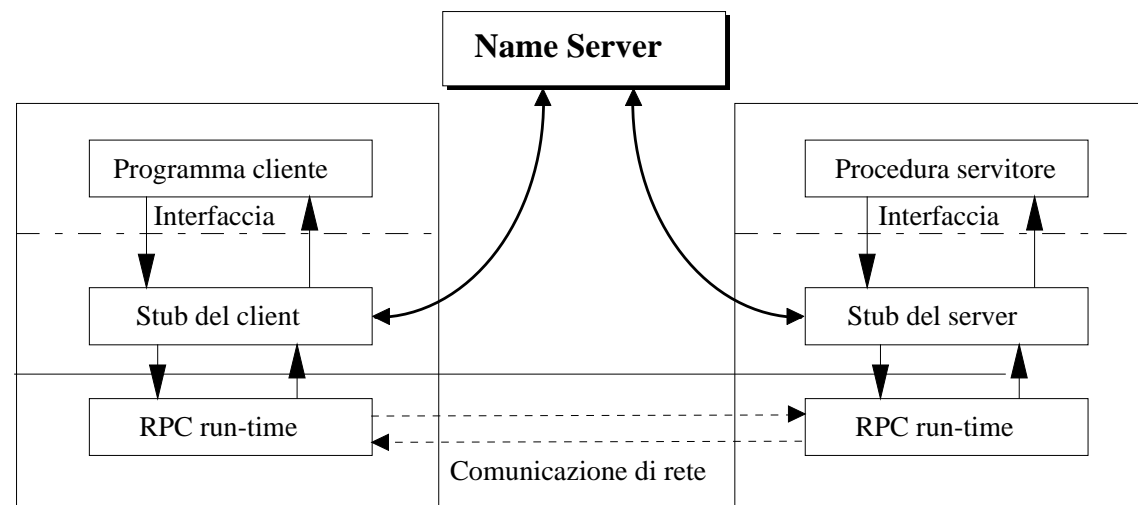
FREQUENZA del BINDING

In caso di BINDING DINAMICO

Ogni chiamata richiede un collegamento dinamico

spesso dopo un primo legame si usa lo stesso binding ottenuto come se fosse statico per questioni di costo

il binding può avvenire meno frequentemente delle chiamate stesse
in genere, si usa lo stesso binding per molte richieste e chiamate allo stesso server



SISTEMI di NOMI

Le RPC hanno portato a molti sistemi di nomi, detti **Binder**, **Broker**, **Name Server**, ecc, tutte entità di sistema per il binding dinamico

Un binder deve fornire operazioni per consentire agganci flessibili

lookup (servizio, versione, &servitore) funzione più usata

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 ogni variazione deve essere comunicata al binder

Il **BINDING** attuato come **servizio coordinato di più servitori**

Uso di binder multipli per limitare overhead e di cache ai singoli clienti o ai singoli nodi

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

CONTROLLO e ASINCRONICITÀ

Necessità di RPC asincrone, o meglio non bloccanti
il cliente non si blocca ad aspettare il servitore

Possibilità di modalità asincrone nei diversi ambienti di uso

- per maggiore parallelismo ottenibile
- con processi leggeri per il server

problema fondamentale come ottenere il risultato

due punti di vista di progetto

RPC bassa latenza vs. RPC alto throughput

RPC bassa latenza

tendono a mandare un messaggio di richiesta ed a trascurare il risultato
realmente asincrone

RPC throughput elevato

tendono a differire l'invio delle richieste per raggrupparle in un unico
messaggio di comunicazione

RPC ASINCRONE

Implementativamente, si usano sia supporti UDP o TCP, ottenendo semantiche diverse

Realmente asincrone (senza risultato)

Athena (XWindows)

usa UDP e bassa latenza - semantica may-be

SUN

usa TCP ad elevato throughput - semantica at-most-once

invio di una serie di RPC asincrone e di una finale in batching

asincrone (con restituzione di risultato)

Chorus - bassa latenza

uso di una variabile che contiene il valore del risultato nel cliente

Mercury - alto throughput

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

PROPRIETÀ delle RPC

Analisi delle proprietà di ogni sistema RPC

proprietà visibili all'utilizzatore

entità che si possono richiedere	operazioni o metodi di oggetti
semantica di comunicazione	maybe, at most once, at least once
modi di comunicazione	a/sincroni, sincroni non bloccanti
durata massima e eccezioni	ritrasmissioni e casi di errore

proprietà trasparenti all'utilizzatore

ricerca del servitore

uso di sistemi di nomi con broker unico centralizzato / broker multipli

presentazione dei dati linguaggio IDL ad hoc e generazione stub

passaggio dei parametri passaggio per valore, per riferimento

eventuali legami con le risorse del server

persistenza della memoria per le RPC

crescita delle operazioni via movimento di codice

aggancio con il garbage collector

RPC di SUN

RPC di SUN con visione a processi e non trasparenza allocazione
operazioni richieste al nodo del servitore

entità che si possono richiedere
semantica di comunicazione
modi di comunicazione
durata massima

solo operazioni o funzioni
at-most-once e at-least-once
sincroni e asincroni
timeout

ricerca del servitore

port mapper sul server

presentazione dei dati

linguaggio IDL ad hoc XDR e generazione stub RPCGEN

passaggio dei parametri

passaggio per valore

le strutture complesse e definite dall'utente sono linearizzate e ricostruite al server per essere distrutte al termine della operazione

estensioni varie

broadcast, credenziali per sicurezza, ...

RMI di JAVA

RMI in Java

visione per sistemi ad oggetti passivi con trasparenza alla allocazione (?), senza eterogeneità, e con scelte che non privilegiano la efficienza

entità da richiedere

metodi di oggetti via interfacce

semantica di comunicazione

at-most-once (TCP)

modi di comunicazione

solo sincroni

durata massima e eccezioni

trattamento di casi di errore

ricerca del servitore

uso di registry nel sistema broker unico centrale
non sono forniti broker multipli (distribuiti?) ma si possono anche avere organizzazioni più complesse

presentazione dei dati

generazione stub e skeleton

passaggio dei parametri

passaggio a default per valore,

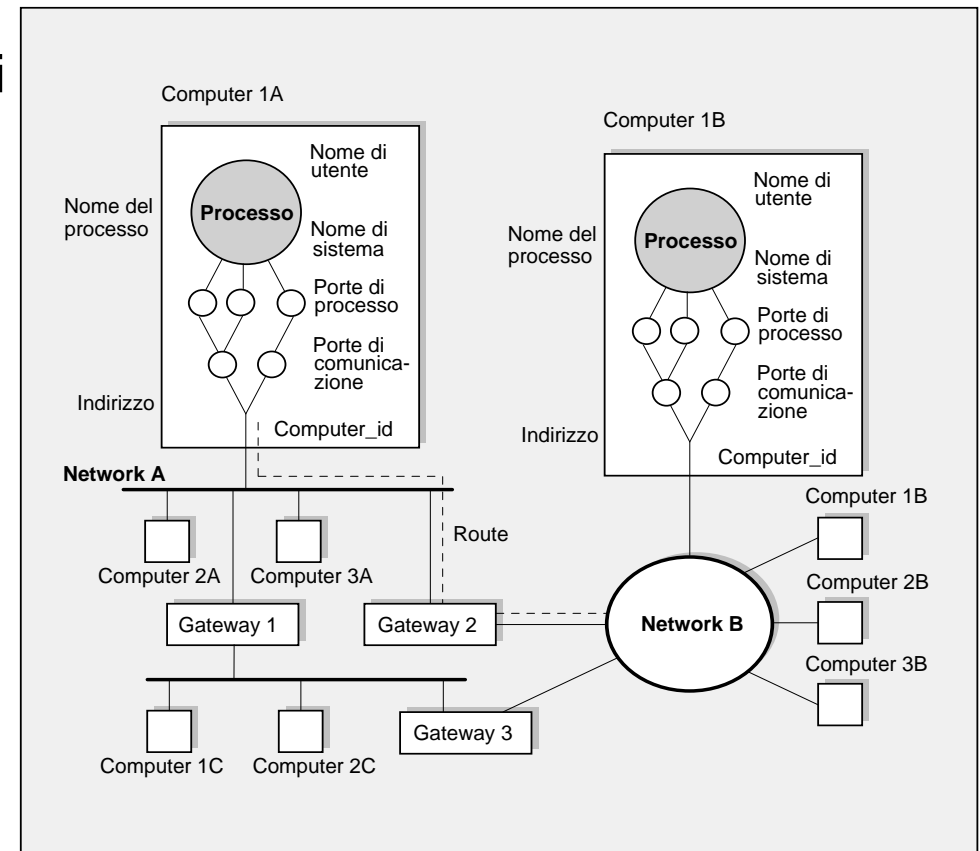
passaggio per riferimento di oggetti con interfacce remotizzabili
eventuali legami con le risorse del server e aggancio con il sistema di sicurezza, e aggancio con il garbage collector

SISTEMI di NOMI

Spesso nei sistemi distribuiti siamo in presenza di molto sistemi di nomi che hanno molte proprietà e sono anche molto diversi

Proprietà di base dei Sistemi di NOMI

- **generalità**
varietà dei nomi disponibili e trattati
- **definizioni multiple della stessa entità**
varietà di nomi per lo stesso oggetto con mapping capace di traslare tra questi
- **distribuibilità**
uso di direttori partizionati e/o replicati
- **user-friendliness**
nomi facili per l'utente



NOMI

Problema fondamentale nel distribuito la necessità di ritrovare (cioè identificare) le altre entità nel sistema

Complessità del problema e difficoltà di soluzioni generali

entità diverse eterogenee ⇒ **livelli diversi di nomi**

più sistemi di naming e più livelli di nomi nel sistema

con contesti di visibilità

più funzioni di trasformazione da nome all'entità

NOMI identificatori come

- **stringa di caratteri** - **nomi esterni** (nomi di utente)
- **numero binario** - **nomi interni** (nomi di sistema)

Le entità di un programma sono oggetti e spesso sono associati a diversi sistemi di nomi

sia nomi di utente (significativi per l'utilizzatore)

sia nomi di sistema (meno leggibili ma più efficienti)

LIVELLI di NOMI

Spesso si possono considerare alcuni livelli di nomi per il distribuito NOME in tre possibili livelli (Shock)

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

nome specifica quale oggetto (entità) si riferisce e denota la entità

indirizzo specifica dove l'oggetto risiede e lo riferisce dopo un binding

route specifica come raggiungere l'oggetto

Funzioni di corrispondenza o MAPPING per passare da una forma ad un'altra e per aiutare l'utente finale

- mapping nomi → indirizzi
- mapping indirizzi → route

I nomi scelti dall'utente, gli indirizzi assegnati dal sistema

l'utente deve specificare nomi e ritrovare route

SISTEMI di NOMI

SPAZI dei NOMI più usati

piatto (flat)

con nessuna struttura, ma adatto per pochi utenti e poche entità

partizionato

gerarchia e contesti (DNS), ad esempio, deis33.deis.unibo.it

descrittivo

con riferimento ad una struttura di oggetto caratterizzato da attributi per identificare la entità corrispondente (OSI X.500)

username e password

attributi con liste di valori (rigidi o meno)

In questi scenari, spesso ci possono essere problemi nell'identificare nomi di gruppo

un nome di gruppo identifica una lista di nomi di entità

Si noti che il problema è simile a quello di un multicast vs, comunicazione punto a punto

SCELTE sui NOMI

Piatto (flat) antonio, acorradi

Chi distribuisce i nomi? Problemi nel caso di molti servitori di nome

Partizionato deis33.cineca.it deis33.deis.unibo.it

Ogni responsabile di partizione mantiene e distribuisce i nomi

Descrittivo nome=Antonio & organizzazione=UniBologna

Ogni nome può identificare anche una molteplicità di entità che si possono trovare solo con una ricerca esaustiva sul sistema di nomi

Nomi di gruppo IP classe D

Ogni gruppo individua un insieme di entità denotate dal nome stesso anche molto lontane, poco correlate, e spesso non gestite dallo stesso servitore di nomi

Necessità di una infrastruttura di supporto al gruppo

COMPONENTI di un SISTEMA di NOMI

In un servizio di nomi, consultato implicitamente o esplicitamente, i clienti del name server sono

- sia i clienti che devono risolvere un nome per potere riferire una risorsa
- sia le entità risorse (server rispetto ai clienti di prima, ossia che devono essere riferiti) che devono rendersi note al servizio e diventano clienti del name server

... a parte questi clienti e le loro richieste, il supporto deve considerare:

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

Le comunicazioni dei clienti con il name server possono essere ottimizzate per le operazioni più frequenti

I clienti propongono la maggiore parte del traffico

Le risorse da registrare fanno operazioni più rare e producono traffico più limitato

COMPONENTI: NAME SERVER

Name server devono **fornire operazioni** per consentire la migliore operatività sugli oggetti interni, ossia le **tabelle di corrispondenza** modellate come **tuple di attributi**

Le operazioni

Query	ricerca un oggetto
AddTuple	aggiungi una tupla dal server
ModifyTuple/DeleteTuple	modifica/togli una tupla
Enumerate	lista tutte le tuple, una alla volta

Ogni sistema di nomi decide:

- il **formato delle tuple**
- il **formato specifico delle operazioni**

Le realizzazioni prevedono sia **Unico servitore** sia **Agenti Multipli**

Il servizio può essere **centralizzato**,

o molto più spesso **distribuito** e anche **replicato** (vedi DNS)

COMPONENTI: COMUNICAZIONE

Nelle realizzazioni con **molteplici Name Server** il servizio prevede una comunicazione tra loro, usando

- messaggi singoli, o datagrammi
- connessioni
- invocazioni di alto livello come RPC

Il traffico tra i diversi Name Server deve essere supportato mentre si continua a fornire il servizio

Il coordinamento dei servitori deve essere minimizzato in tempo ed uso delle risorse tenendo conto anche delle proprietà che si vogliono garantire

In uno spazio piatto, necessità di fare una partizione dello spazio dei nomi per limitare la coordinazione

In uno spazio partizionato, i nomi sono usati dalla autorità preposta senza coordinamento

Le gestione delle tabelle e il coordinamento creano problemi di consistenza e affidabilità, complicato dalla replicazione

COMPONENTI: GESTIONE NOMI

I nomi in molte forme e in base a queste la gestione
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)

Nella gestione dei nomi sono fondamentali due decisioni

- Distribuzione dei nomi
- Risoluzione dei nomi

Distribuzione dei nomi

I nomi sono mantenuti in oggetti che ne hanno la responsabilità o autorità
con un partizionamento tra i server responsabili

Come dividere la gestione e il mantenimento?

Con politiche di Clustering di vario genere

- | | |
|----------------------------|---|
| Algoritmico | (hash table / tabelle hash) es. funzione di mapping |
| Sintattico | (pattern matching) es. iniziale del nome |
| Basato su Attributi | (tuple) es. sulla base del valore di attributi |

RISOLUZIONE NOMI

Per la Risoluzione dei nomi, le richieste dal cliente devono fluire fino al server che può dare risposta

Il processo di risoluzione dei nomi per dare una risposta prevede alcune fasi (non sempre presenti)

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

Ogni nodo specifica i name server noti e tende a limitare se possibile le comunicazioni tra i server

Strategie usuali per limitare i costi sono:

- Politiche di caching
- Politiche di route tra server
- Creazione di contesti o vicinati tra i server
- Propagazione di conoscenza tra vicini

ESEMPIO di SISTEMA GLOBALE

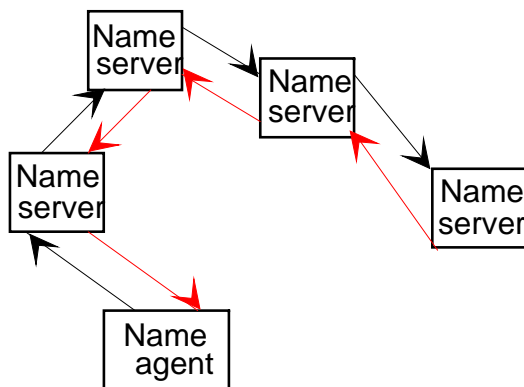
USO DI CONTESTI E LOCALITÀ nei sistemi globali tipo DNS

Si distribuisce e risolve nel contesto locale

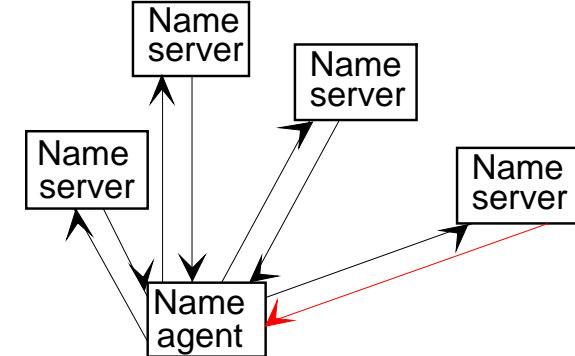
Si ricorre ad altri contesti solo in caso sia necessario

Le strategie di coordinamento tra i server devono essere a basso costo, se possibile

RISOLUZIONE RICORSIVA



RISOLUZIONE ITERATIVA



(non DNS) RISOLUZIONE TRANSITIVA

ALTRI SISTEMI di NOMI (oltre DNS)

oltre al DNS, i sistemi di nomi hanno organizzato i nomi attraverso attributi e ricerca su questi

OSI X.500 o Directory - Servizio standard di Direttorio e di Nomi con *realizzazione partizionata, decentralizzata, disponibile 24/7*

CCITT definisce X.500 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"

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

DIRECTORY X.500

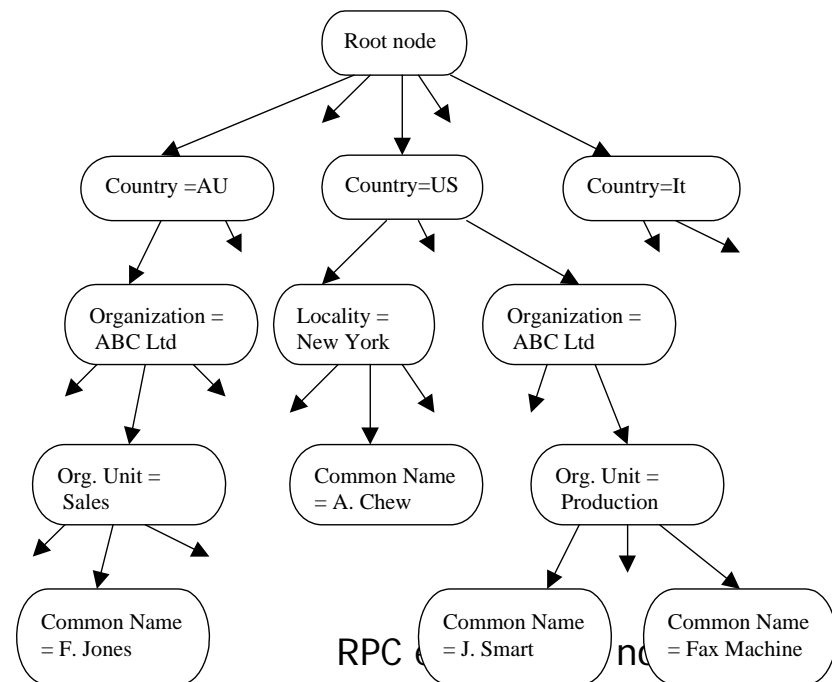
X.500 è un insieme di standard, molto articolato e completo

La base è l'insieme delle informazioni che caratterizzano il directory, che sono organizzate, in prima battuta, in un **albero logico** detto Directory Information Tree (DIT)

a formare il Directory Information Base (DIB),

L'albero logico è costruito in base al valore di attributi del tutto liberi e a scelta dell'utente

La novità sta nella **organizzazione basata sui contenuti** e **le ricerche** che si possono fare **in modo molto flessibile** per **singole entità** e **anche per gruppi di elementi**



NOMI in DIRECTORY X.500

Ogni entry (o nodo) 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

DN può fungere da chiave per identificare unicamente un nodo connotato da un insieme (tupla) di coppie **attributo = valore**

ad esempio: country, organization, organization unit, common name

US ABCLtd Production J.Smart

Le ricerche possono essere fatte in modo globale o contestuale per un DN ma **anche per contenuto dei nodi**, in base agli **attributi**:
ad **un attributo** o ad un **filtro generalizzato sugli attributi** portando ad un nodo risultato o più nodi

RICERCHE con FILTRI

I filtri sono **molto potenti come capacità espressiva**
ad esempio, sono permesse **condizioni logiche sugli attributi**

CN=Corradi AND C=Italy

anche con espressioni regolari

email=*hotmail*

anche con condizioni aritmetiche

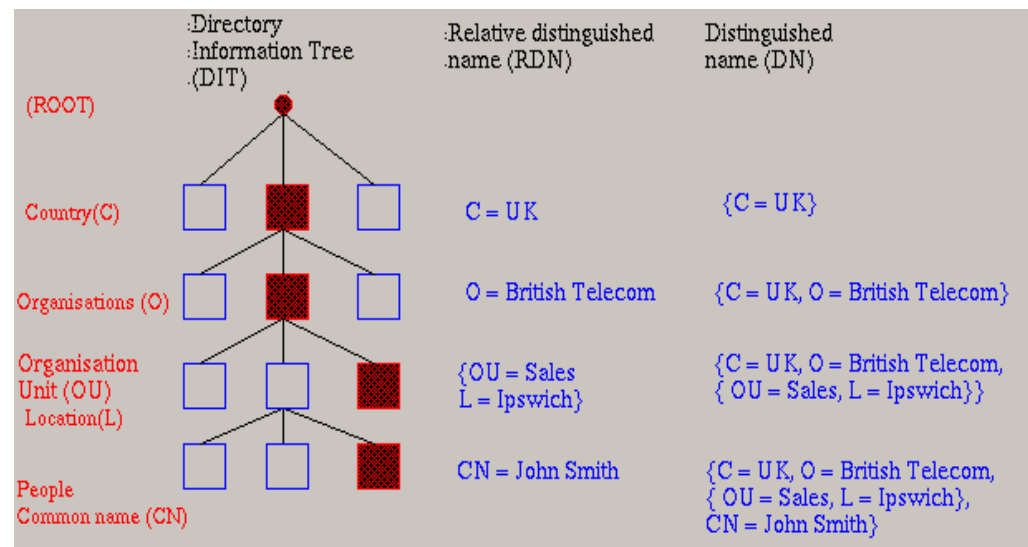
(age >18) AND (cookies <10)

Le ricerche si possono applicare anche a scope limitati (contesti o sottoalberi)

Le operazioni previste sono molte

La prima è il **bind** con il directory poi **ricerche** frequenti e **cambiamenti** meno frequenti

La interfaccia con il direttorio anche molto complessa tenendo conto anche della durata delle operazioni



ORGANIZZAZIONE INTERNA X.500

Ricerca sul direttorio X.500 avviene attraverso agenti:

DUA, Directory User Agent tramite per fare richieste

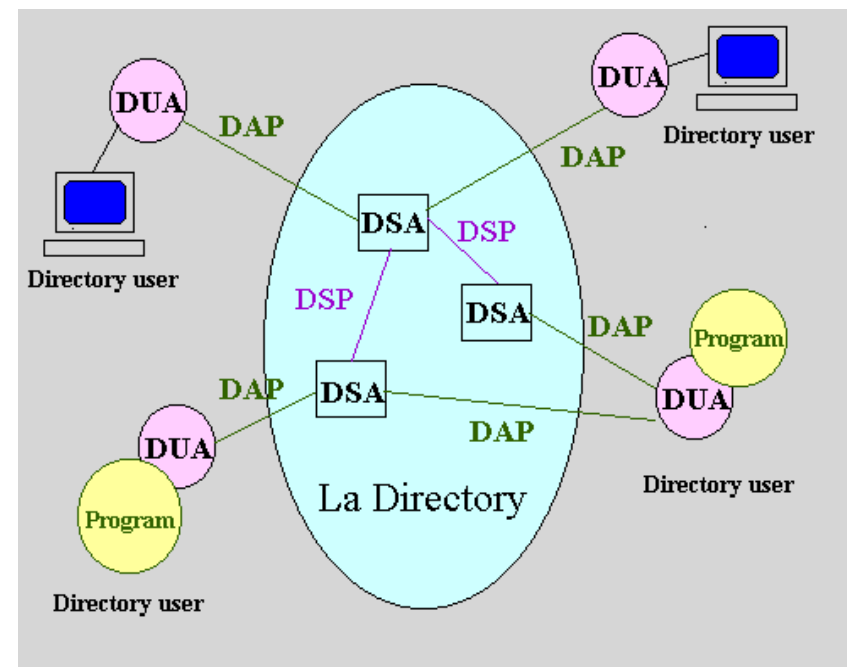
DSA, Directory System Agent che mantiene informazioni di contesto

DSP, Directory System Protocol per scambiare informazioni tra DSA

DAP, Directory Access Protocol, protocollo di accesso al direttorio

Dopo la connessione, si fanno operazioni di lettura, confronto, ricerca, lista delle entità con tecniche di ricerca ricorsive ed iterative

LDAP,
Lightweight Directory Access Protocol
Protocollo limitato compatibile Internet
usato per infrastrutture di
verifica certificati



DIRECTORY e NON DB

Obiettivo di un directory è mantenere informazioni su un insieme di **risorse eterogenee** per un ambiente evitando **duplicazioni di informazioni e problemi di sincronizzazione** e consentendo

- una capacità espressiva ampia ed estendibile
- una gestione anche molteplice con più autorità per parti
- una sicurezza anche partizionata e differenziata

Al contrario di un database (o più database)

- si associano **attributi anche diversi con i singoli oggetti**
- **gli oggetti sono indipendenti tra di loro** (e possono essere diversi)
- si considera la **relazione di contenimento** alla base della organizzazione
- si possono avere proprietà di accesso differenziate per i singoli oggetti
- si ottimizza considerando un numero elevato di letture e 'poche' scritture

Necessità di un protocollo standard unificato per accedere alle informazioni, esprimere le specifiche di accesso, ed estrarre informazioni in modo efficace

USO di DIRECTORY

i Directory sono tipicamente usati (**vedi costo**) per rappresentare entità eterogenee, come persone, risorse, servizi, server, ...

In generale, per informazioni concettuali che rappresentano la gestione di oggetti comuni in un ambiente condiviso

i certificati per autenticare e autorizzare accesso

MANAGEMENT DELLE RISORSE

In un sistema di gestione in cui consideriamo una molteplicità di gestori con autorità e responsabilità anche non completamente note e forse variabili lentamente nel tempo

si usano DIRECTORY per

- . localizzare i diversi gestori e le loro politiche
- . trattare i problemi di domini incrociati (cross-domain)
- . ritrovare le proprietà delle risorse
- . ritrovare le proprietà dei gestori delle risorse

I costi sono dipendenti dal numero dei nodi (e attributi)

motivati dalle proprietà di QoS offerto (affidabilità e disponibilità)

SVILUPPO dei SISTEMI di NOMI

Due forme di evoluzione

Protocolli di Directory vs. Protocolli di Discovery

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

Directory soluzioni di nomi globali

servizi completi e complessi con 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, essenzialmente stabili, come

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

informazioni locali, anche molto variabili, come

descrizione delle risorse locali, dei gestori presenti, ecc.

PROTOCOLLI DI DIRECTORY

Un servizio di directory garantisce le proprietà di replicazione, sicurezza, gestione multiserver, ..., tutto il supporto per memorizzare le informazioni organizzate prevedendo molti accessi in lettura e poche variazioni

UPnP (Universal Plug-and-Play) Standard per 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) e uso di nomi URL

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

implementazioni: Cisco, Apple, Novell

PROTOCOLLI DI DISCOVERY

Per 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 (località delle risorse)

JINI protocollo Java per il discovery di appliances

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

Protocolli di lookup

Start up con multicast

in ambiente locale

Il discovery server
verifica la presenza

delle risorse ad intervalli opportuni

