Indice

Parte I
Introduzione

Un sistema distribuito è una collezione di computer indipendenti che appare ai propri utenti come un singolo sistema coerente. Per garantire la interoperabilità tra sistemi eterogenei (Diversi OS) viene introdotto un livello comune chiamato *middleware* che funge da intermediario tra le diverse applicazioni e i differenti computer. Esempi di sistemi distribuiti sono i DNS e in un certo senso il web.

1 Obbiettivi

Accessibilità

Un sistema distribuito deve rendere accessibili le sue risorse.

Trasparenza

Trasparenza nei sistemi distribuiti rispetto a:

- ACCESSO: nasconde le differenze nella rappresentazione dei dati nella modalità di accesso alle risorse.
- UBICAZIONE(LOCATION): nasconde dove è localizzata una risorsa.
- MIGRAZIONE: nasconde l'eventuale spostamento della risorsa.
- RIPOSIZIONAMENTO: nasconde la possibilità di spostare la risorsa mentre è in uso.
- REPLICA: nasconde la replica di una risorsa.
- CONCORRENZA: nasconde la condivisibilità di una risorsa da parte di molti utenti contemporaneamente.
- GUASTO: nasconde il malfunzionamento e la riparazione di una risorsa.

Apertura (openess)

Un sistema distribuito aperto è un sistema che offre servizi rispettando le regole standard che descrivono la semantica e la sintassi dei servizi stessi. Il rispetto di queste regole avviene mediante l'utilizzo di interfacce scritte in IDL (Interface Definition Language) che descrivono il comportamento che deve rispettare il sistema distribuito.

Un sistema distribuito aperto offre le seguenti proprietà:

- Interoperabilità: due implementazioni di sistemi o componenti possono coesistere e collaborare unicamente sui reciproci servizi definiti da un standard comune.
- Portabilità: un'applicazione sviluppata per il sistema distribuito A può essere eseguita senza modifiche su un sistema distribuito B differente che implementa la stessa interfaccia di A.
- Ampliabilità: l'aggiunta di nuovi componenti in un sistema distribuito deve risultare facile.

L'apertura di un sistema distribuito può essere raggiunta con:

- L'utilizzo di protocolli standard.
- La pubblicazione di interfacce chiave.
- Test e verifica della conformità dei componenti con gli standard pubblici.

Scalabilità

Un sistema può essere scalabile rispetto:

- Alla propria dimensione: possiamo aggiungere utenti e risorse al sistema.
- Al punto di vista geografico: utenti e risorse possono essere anche molto lontani.
- Alla amministrazione: facilmente gestibile anche se copre molte strutture indipendenti.

Non sono scalabili sistemi distribuiti che hanno:

- Servizi centralizzati (es. un unico server per tutti gli utenti, non è scalabile sul numero di utenti che vogliono accedere);
- Dati centralizzati (es. un unico data base, come un'unica guida telefonica per tutti i numeri);
- Algoritmi centralizzati (es. routing basato su conoscenza completa della rete, oppure elettore di un unico coordinatore).

Caratteristiche degli algoritmi decentralizzati

- Nessuna macchina ha informazioni complete sullo stato del sistema.
- Le macchine prendono decisioni basandosi solo sulle informazioni locali.
- Il malfunzionamento della macchina non danneggia l'algoritmo.
- Non si suppone che esista un clock globale (macchine non temporizzate).

Tranelli

False ipotesi che si assumono quando si progetta un sistema distribuito:

- la rete è affidabile
- la rete è sicura
- la rete è omogenea
- la topologia non cambia
- la latenza è 0
- l'ampiezza di banda è infinita
- il costo del trasporto è 0
- c'è un solo amministratore
- debuggare un'applicazione distribuita è analogo a debuggare un'applicazione standard

2 Tipi di sistemi distribuiti

2.1 Sistemi di calcolo distribuito

Cluster

Nei sistemi di calcolo a cluster l'hardware sottostante è costituito da un insieme di workstation o pc connessi ad una LAN ad alta velocità, inoltre ogni nodo ha lo *stesso sistema operativo*.

Obbiettivi:

- Alte performance.
- Alta disponibilità.

Due differenti approcci:

- Asimmetrico: esiste un nodo principale che distribuisce i compiti ad un set di nodi costituiti da computer. (Beowulf Linux Cluster).
- Simmetrico: non esiste un nodo principale, tutti i nodi sono uguali e l'utente percepisce il sistema come un unico singolo computer. (MOSIX permette ai processi di migrare tra i nodi del cluster in modo dinamico).

GRID

Sono sistemi distribuiti spesso costituiti come un gruppo di computer eterogenei dove ogni sistema può risiedere in un dominio di amministrazione diverso e può distinguersi in termini di hardware, software e tecnologie di rete. (A differenza dei Cluster si possono avere *diversi* OS).

Obbiettivi:

- Alte performance.

L'idea base è stata quella di ispirarsi alla rete elettrica; dove i generatori sono distribuiti ma gli utenti hanno la possibilità di fruire dell'energia elettrica senza curarsi della sua provenienza.

I nodi di un sistema distribuito di tipo GRID non sono limitati a server ma possono essere:

- Super computer.
- Network Storage System.
- Database.
- Dispositivi speciali (sensori, telescopi...).

Il GRID è percepito come un singolo sistema di una organizzazione virtuale.

OGSA Open Grid Services Architecture É un'evoluzione dei sistemi GRID che viene applicata ai Web Service.

Cloud Computing

É un'evoluzione dei OGSA + paradigma software-as-a-service (es Google che offre Office attraverso GoogleDocs).
È un modello emergente e scalabile. Un servizio Cloud offre accesso alle informazioni personali (o aziendali) sempre e ovunque. Un azienda che si propone di offrire servizi cloud mette a disposizione la sua infrastruttura per immagazzinare i dati in modo sicuro. É possibile anche trattare le informazioni immagazzinate nel cloud.

Esempi: Infrastrutture e applicazioni di Google, AE2C (Amazon Elastic Compute Cloud) e DropBox.

Infrastruttura di Google non è stata interamente svelata per questioni di privacy; è costituita da diversi Data Center sparsi in tutti il mondo (per questioni di efficienza, loadbelancing).

2.2 Sistemi Informativi Distribuiti

Sistemi Transazionali

La caratteristica principale di una transazione è che vengono eseguite tutte le operazioni o nessuna di loro. Le operazioni in base alle transizioni possono essere chiamate di sistema, procedure di librerie o blocchi di istruzioni in un linguaggio. Transazioni nested possono essere distribuite su diversi nodi.

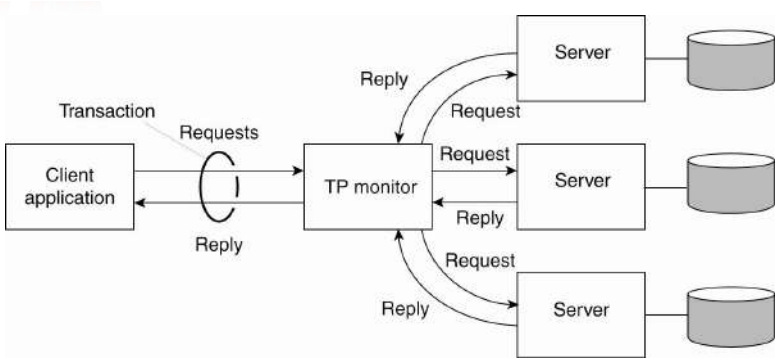


Figura 1: Transaction Process System

Le transazioni hanno queste proprietà: (ACID)

- Atomiche, la transazione per il mondo esterno è indivisibile.
- Consistenti, quando inizia una transazione il database si trova in uno stato coerente e quando la transazione termina il database deve essere in uno stato coerente, quindi non devono verificarsi contraddizioni (inconsistenza) tra i dati archiviati nel DB.
- Isolamento: ogni transazione deve essere eseguita in modo isolato e indipendente dalle altre transazioni.
- Durabilità: una volta che la transazione ha reso effettiva la modifica (commit) esse sono permanenti.

EAI (Enterprise Application Integration)

La comunicazione tra le diverse applicazioni del sistema avviene senza ricorrere a transazioni sui db. Si utilizza un livello *middleware* tra client e server. Questo livello *middleware* gestisce le comunicazioni tra i diversi componenti del sistema utilizzando RPC, RMI e messaggi.

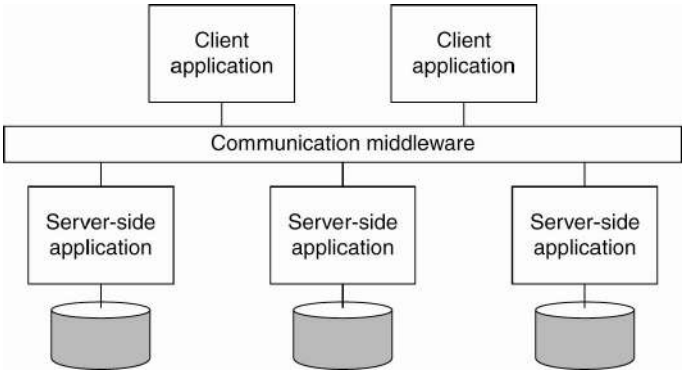


Figura 2: EAI

2.3 Sistemi Distribuiti Pervasivi

Sono caratterizzati da:

- Eterogeneità dei computer.
- L'instabilità di nodi e della rete (es dispositivi mobili, embedded system, sensori...): i nodi possono muoversi e cambiare le loro caratteristiche, passare da una rete a un'altra (perdere la connessione o cambiare il tipo di connessione)...
- Sistema libero dal contesto, adattabile.

Parte II
Architetture

3 Stili architetturali

Si esprimono in termini di *componenti*, dal modo in cui sono connessi, dai dati scambiati e infine come questi elementi sono configurati congiuntamente tra loro. Un *componente* è un'unità modulare con interfacce richieste e fornite ben definite. Un *connettore* è descritto come un meccanismo che media la coordinazione o la cooperazione tra i componenti. Non sono rigidamente applicati ossia si può adottare una soluzione ibrida.

3.1 Architetture a livelli o multilivello

É un approccio già utilizzato per i due modelli di rete (TCP-IP, ISO-OSI). L'architettura a livello permette di astrarre dai particolari che caratterizzano i livelli inferiori e permette di separare funzionalmente le componenti del sistema delegando a ciascun livello il proprio compito.

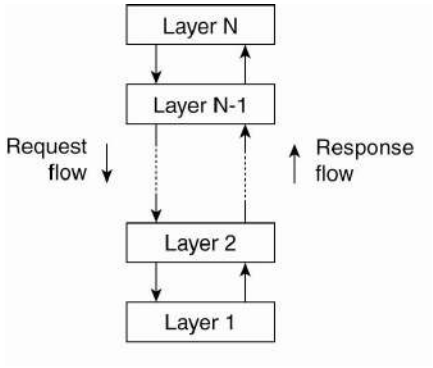


Figura 3: Architettura a livello

3.2 Architetture basate sugli oggetti

Rappresenta ogni componente del sistema come un oggetto. Ogni oggetto mette a disposizione dei metodi (remoti) i quali sono visibili e possono essere richiamati dagli altri oggetti dell'architettura. (CORBA).

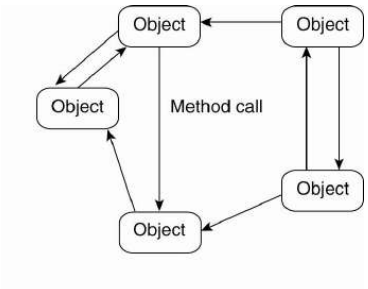


Figura 4: Architettura basata sugli oggetti

3.3 Architetture centrate sui dati

É uno stile in cui le componenti del sistema usano uno spazio di dati condiviso per interagire tra di loro. (File system distribuiti).

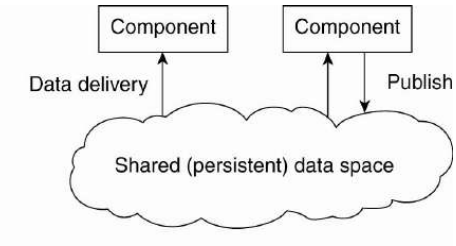


Figura 5: Architettura centrate sui dati

3.4 Architetture basate sugli eventi

In questo tipo di stile il *middleware* che realizza il sistema distribuito deve gestire gli eventi. La comunicazione tra le varie componenti avviene tramite occorrenze di eventi. Ogni componente interessato ad un particolare evento viene notificato ogni volta che quel particolare evento accade.

Esempio: Publish Subscribe (Un componente pubblica una notizia, le altri componenti fanno una sottoscrizione solo a particolari notizie, quando viene pubblicata una notizia, solo alle componenti che hanno fatto la sottoscrizione viene notificata la notizia).

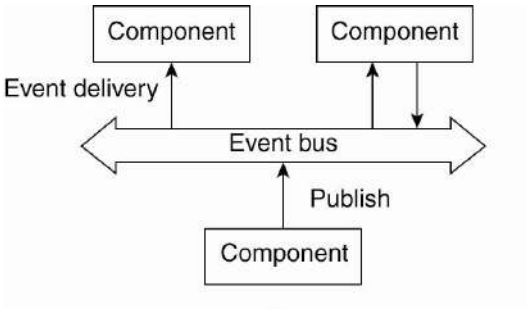


Figura 6: Architettura basata sugli eventi

4 Architetture di sistema

4.1 Architetture Centralizzate

Il modello base è quello in cui vi è un'interazione tra un componente che richiede un servizio (client) e l'altro componente che risponde (server). L'interazione avviene stabilendo un canale di comunicazione. Il processo server si occupa di gestire le richieste che provengono da n client.

Anche in questo tipo di architetture è possibile applicare lo stile a livelli, per esempio:

- Livello interfaccia utente.
- Livello applicativo.

- Livello gestione dei dati.

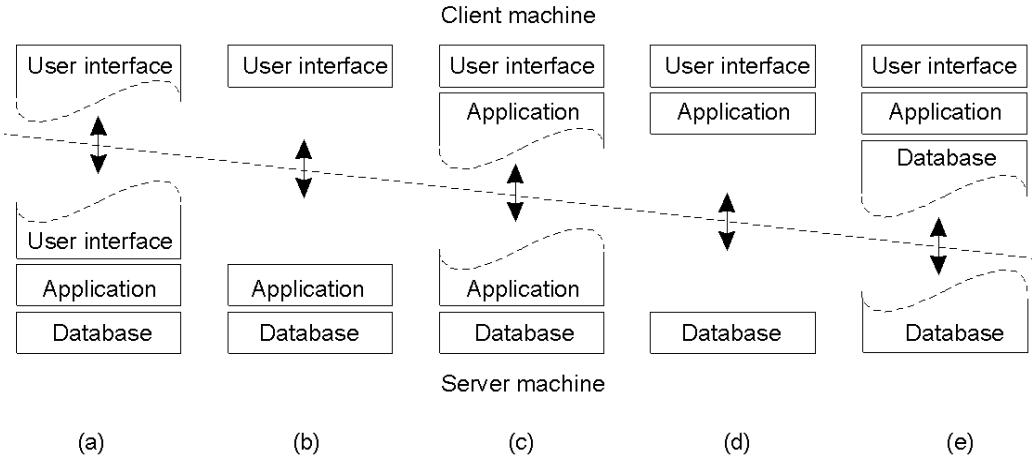
Esempio un motore di ricerca in internet.

Distribuzione verticale: Funzionalità diverse vengono assegnate a componenti diversi del sistema (es architettura a livelli). Ad ogni livello dell'architettura viene assegnato una funzionalità diversa.

Distribuzione orizzontale: La stessa funzionalità è distribuita tra più componenti del sistema (es replica di web service). Risolve diversi problemi di scalabilità, loadbelancing...

Architetture Multitired

É un architettura client server nella quale la suddivisione dei livelli viene gestita in base alle caratteristiche sia del client che del server; per esempio se ho a disposizione un dispositivo mobile il livello applicativo risiederà sul lato server, mentre se ho a disposizione un computer, parte (o tutto) il livello applicativo risiederà sul client.



Alternative client-server organizations (a)–(e)

4.2 Architetture Decentralizzate

P2P

Nel modello Peer-To-Peer:

- Tutti i nodi hanno le stesse capacità funzionali.
- Ogni nodo condivide risorse.
- Le operazioni non dipendono da un nodo che viene amministrato centralmente.

Esempio di sistema P2P: Napster. Non è un sistema P2P puro infatti esistevano dei directory server nei quali venivano immagazzinati gli indirizzi dei nodi che contenevano i file condivisi.

Operazioni svolte durante una richiesta di un file:

1. Richiesta al directory server.

2. Risposta contente la lista di peer.
3. Scelta di un peer dalla lista e richiesta del file a quel nodo.
4. Invio del file.
5. Aggiornamento della lista sul directory server con il proprio indirizzo.

Problema delle reti P2P: Come immagazzinare i dati in diversi host per raggiungere il bilanciamento del carico per l'accesso ai dati e assicurare la disponibilità dei dati senza generare troppo traffico inutile.

Tre generazioni di reti P2P:

1. Napster
2. Dai limiti del protocollo di Napster (scalabilità, resistenza ai guasti, anonimità) sono nati:
   - Gnutella, FreeNet, bitTorrent (più decentralizzato e puro rispetto a Napster)
3. P2P Middleware
   - Pastry, Tapestry, Chord, CAN, Kadmlia
     Obbiettivi:
     Permettere ai client in modo trasparente di:
     – Localizzare le risorse e comunicare con tutti i nodi in rete.
     – Aggiungere nuove risorse.
     – Aggiungere e rimuovere nodi.
     Criteri di ottimizzazione:
     – Scalabilità globale.
     – Efficienza, rapidità.
     – Bilanciamento del carico.
     – Località della interazioni (si privilegiano i nodi vicini).
     – Supporto alla autenticazione e alla cifratura.
     – Anonimità.

Overlay Network

Ovvero una rete in cui i nodi sono costituiti dai processi e i collegamenti rappresentano i possibili canali di comunicazione.
Overlay significa che sono ad un livello che astrae dal livello reale. L'instradamento dei messaggi avviene tramite un algoritmo indipendente dalla costituzione della rete sottostante. Questo significa che l'instradamento, oltre che avvenire a livello rete, avviene anche a livello applicazione (dove viene creata la Overlay Network). Vengono introdotti degli identificatori globali i quali identificano una singola replica della risorsa. Questi algoritmi gestiscono anche l'aggiunta e la rimozione di risorse e nodi.

Vi sono due tipi di Overlay Network:

- Strutturato
  - L'Overlay Network viene costruita in modo deterministico utilizzando una DHT (Distributed Hash Table). L'obbiettivo è quello di fare un instradamento efficiente al nodo che contiene i dati.
- Non strutturato
  - L'Overlay Network viene costruita mediante l'utilizzo di algoritmi randomizzati.
  - Ogni nodo (peer) non conosce tutto il resto della rete, non c'è una conoscenza totale della rete ma solo una conoscenza locale.
  - Spesso vi è una struttura gerarchica. Per problemi di scalabilità a volte vi sono dei nodi che sono visti come superpeer in quanto svolgono funzioni differenti dai normali peer (es. Directory Server di Napster).

Chord

É un esempio di Overlay Network strutturata. L'assegnamento delle risorse al nodo avviene mediante DHT. La topologia tipica dell'algoritmo Chord è ad anello. Lo spazio di indirizzamento è determinato a priori e permette l'indirizzamento dei nodi (nodi e risorse condividono il medesimo spazio di indirizzamento). Nell'immagine lo spazio di indirizzamento è composto da 4 bit quindi da 0 a 15 nodi. Una risorsa risiede sul nodo che ha il più piccolo identificatore tra quelli con identificatori più grandi o uguale della risorsa stessa. L'inserimento di un nodo nell'anello di Chord non è banale: se dovessi inserire un nodo dovrei spostare le risorse (del nodo con identificatore maggiore) che hanno identificatore minore o uguale all'identificatore del nodo che inserisco. Se dovessi eliminare un nodo dovrei spostare le risorse del nodo eliminato al nodo che ha il primo identificatore più grande rispetto a quello eliminato.

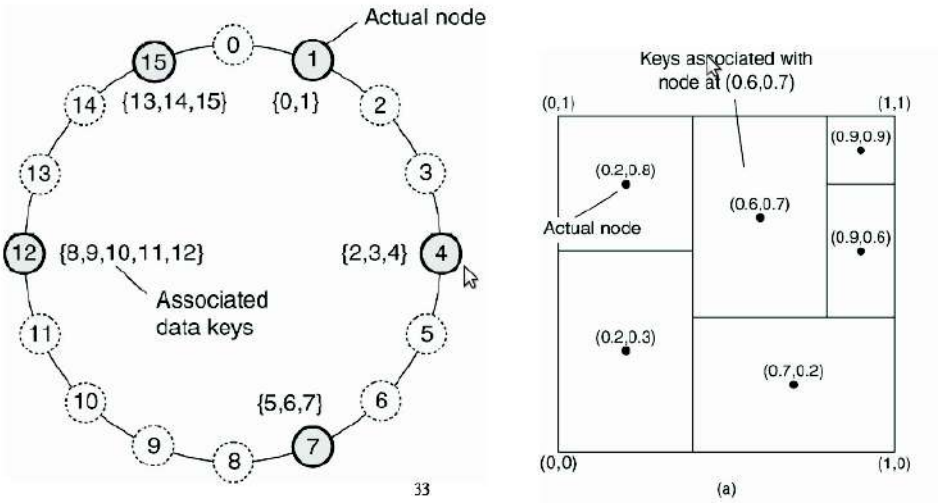


Figura 7: Chord & CAN

CAN (Content Addressable Network)

É un altro esempio di rete strutturata. L'algoritmo partiziona lo spazio di indirizzamento in modo equo e assegna una regione di questo spazio a ciascun nodo. Le risorse vengono indirizzate attraverso coordinate spaziali e, in base alla regione in cui cadono, vengono gestite dal nodo di competenza.

Il metodo è stato ideato per migliorare l'inserimento di nuovi nodi infatti, l'inserimento di un nodo avviene mediante la divisione in due parti del rettangolo. Le risorse sono suddivise equamente tra il nodo nuovo e il nodo precedente. La rimozione di un nodo non è altrettanto semplice: bisogna ripartizionare tutto lo spazio perché non è sempre possibile creare un rettangolo dall'unione di due rettangoli di forma differente.

Overlay non strutturati

L'Overlay è creato mediante lo scambio di viste (conoscenza della rete che ha il peer - può essere parziale). L'unione di tutte le viste forma la vista completa della rete.
Vantaggio:

- Scalabilità, in questo caso sono i nodi che svolgono tutto il lavoro.

Svantaggio:

- Possibilità di zone morte nella rete, difficoltà nella ricerca di risorse in rete poiché la ricerca avviene mediante l'interrogazione dei peer vicini, i quali, a loro volta, interrogano i peer vicini (flooding). Non è detto che in questo modo si riesca a raggiungere tutti i nodi presenti nella rete.

Esempio: Ethernet si basa su un protocollo con componente di casualità elevata. Algoritmo Gossip ("spio" quello che fanno i vicini).

Information Dissemination Models

- Modello a propagazione Anti-Entropica
  - Il nodo P sceglie un altro nodo Q in modo random nella sua vista parziale.
  - Successivamente scambia aggiornamenti con Q (non tutti - si adottano meccanismi di *aging* - informazioni vecchie non vengono trasmesse).
- Modi di scambiarsi gli aggiornamenti
  - P invia i propri aggiornamenti a Q.
  - P riceve gli aggiornamenti da Q.
  - P & Q si scambiano messaggi tra di loro.

Per la comunicazione tra i vari nodi vengono utilizzati due thread: uno attivo (periodicamente ripetuto - invia metà vista al nodo scelto random), uno passivo (che è sempre pronto a riceve i messaggi).

Gestione della topologia

SuperPeers è un nodo che, oltre a fare quello che fanno gli altri, è a conoscenza dello stato dei nodi nella sua sotto rete. Questo approccio migliora il loadbelancing.

4.3 Architetture Ibride

Sono architetture parzialmente centralizzate e parzialmente decentralizzate.

BitTorrent

É P2P ma c'è anche una componente globale (ricerca via web del file .torrent). Ottenuto il .torrent si acquisisce il tracker del file che contiene la lista dei nodi che hanno a disposizione il file. Torrent utilizza un criterio per cui scarica prima le parti più rare e, nel momento che noi abbiamo completato una parte la mette a disposizione agli altri utenti della rete.

Parte III
Processi

5 Virtualizzazione

Virtualizzazione: permette di nascondere i dettagli dell'hardware rispetto ai servizi che vengono implementati su di esso.

Differenza tra virtualizzazione classica e virtualizzazione nei sistemi distribuiti
Virtualizzazione classica:

- I programmi si appoggiano ad una interfaccia che rende trasparente il livello sottostante (hardware e OS).

Virtualizzazione nei sistemi distribuiti:

- I programmi sfruttano ancora un'interfaccia che si appoggia su un sistema virtuale (es JAVA e la Java Virtual Machine)

5.1 Architettura di una macchina virtuale

É costituita da interfacce a differenti livelli:

- Un'interfaccia tra hardware e software che include le varie istruzioni macchina (questa interfaccia può essere invocata da qualsiasi programma - "General instructions").
- Un'interfaccia tra hardware e software che include le varie istruzioni macchina (questa interfaccia può essere invocata solo da programmi privilegiati come ad esempio l'OS - "Privileged instructions").
- Un'interfaccia che contiene le chiamate di sistema offerte da un OS - "System calls".
- Un'interfaccia che contiene le chiamate di libreria (Application Programming Interface - API) - "Library functions".

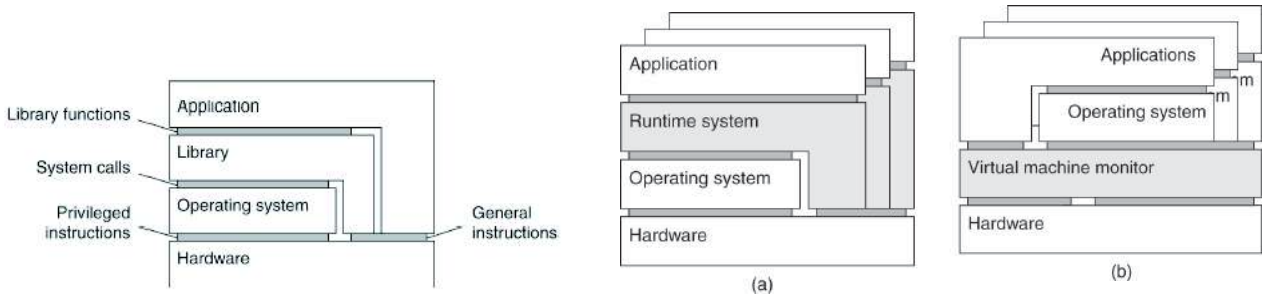


Figura 8: Nel caso di OS multipli si inserisce un livello di interfaccia tra i vari OS e l'hardware della macchina. Questo livello è chiamato "Virtual Machine Monitor".

Due modi per intervenire con la virtualizzazione:

- Process Virtual Machine: (a) Fornisce un insieme di istruzioni astratte che vengono fornite al livello applicativo. Questo livello utilizza sia *system call* che *general instruction*s. Es. Java che si appoggia al Java Runtime Environment (JRE), che è composto dalla JVM, dalle classi core di Java e dalle librerie.
- Virtual Machine Monitor: (b) Nasconde totalmente il livello hardware del sistema fornendo un'insieme di istruzioni sulle quali possono essere installati diversi OS. Es. VMWare.

Part IV
Communication

6 Tipi di comunicazione

OSI: Comunicazione orizzontale ideale tramite protocolli, in realtà si tratta si comunicazione verticale.
Protocollo: insieme di regole che stabiliscono il formato dei dati che vengono scambiati.

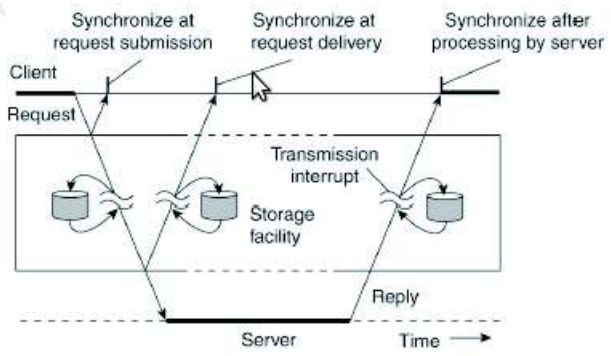


Figura 9: Esempio di comunicazione *Persistente*. Tutto quello presente nel rettangolo è il livello middleware.
Storage Facility: permettono di ovviare al problema di sovraccarico del server. Memorizzano temporaneamente la richiesta nel caso in cui il server non sia momentaneamente raggiungile (per es. a causa di sovraccarico). Esempio SMS: anche a telefono spento i messaggi vengono memorizzati, quando viene riacceso i messaggi vengono ricevuti.

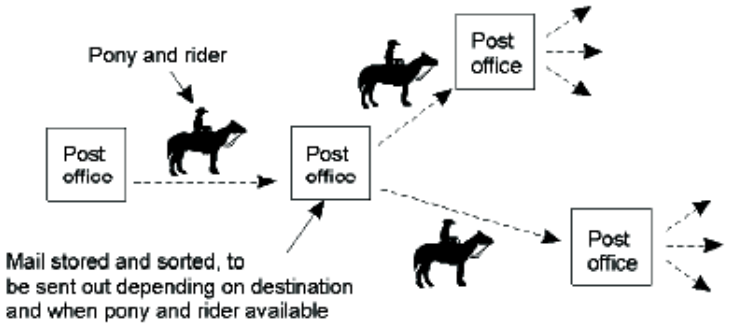


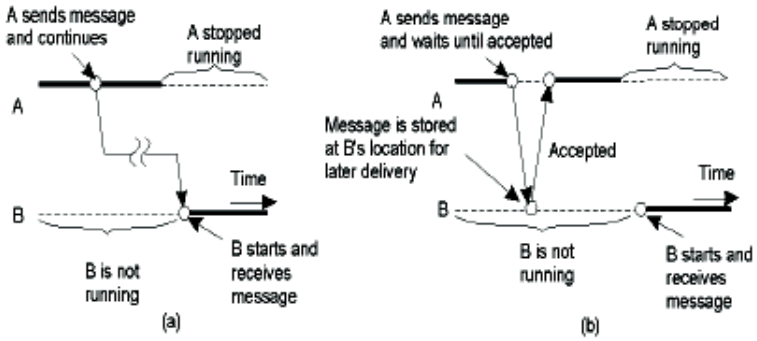
Figura 10: Altro esempio di persistenza poiché le lettere venivano immagazzinate negli uffici postali in attesa che vengano consegnate.

6.1 Comunicazione Persistente

La comunicazione avviene anche quando il destinatario non è presente. Due tipi:

(a) Asincrona: A non si ferma in attesa della risposta di B inoltre è compito del middleware memorizzare il messaggio finché B non si collega.

(b) Sincrona: A si ferma quando invia il messaggio in attesa di una risposta, in B è implementato un sistema di memorizzazione che immagazzina i dati nel caso in cui l'applicazione non è disponibile. Inoltre invia ad A una notifica (*accepted*) per permettergli di risvegliarsi.



a) Persistent asynchronous communication
b) Persistent synchronous communication

Figura 11: Comunicazione Persistente

6.2 Comunicazione Transiente

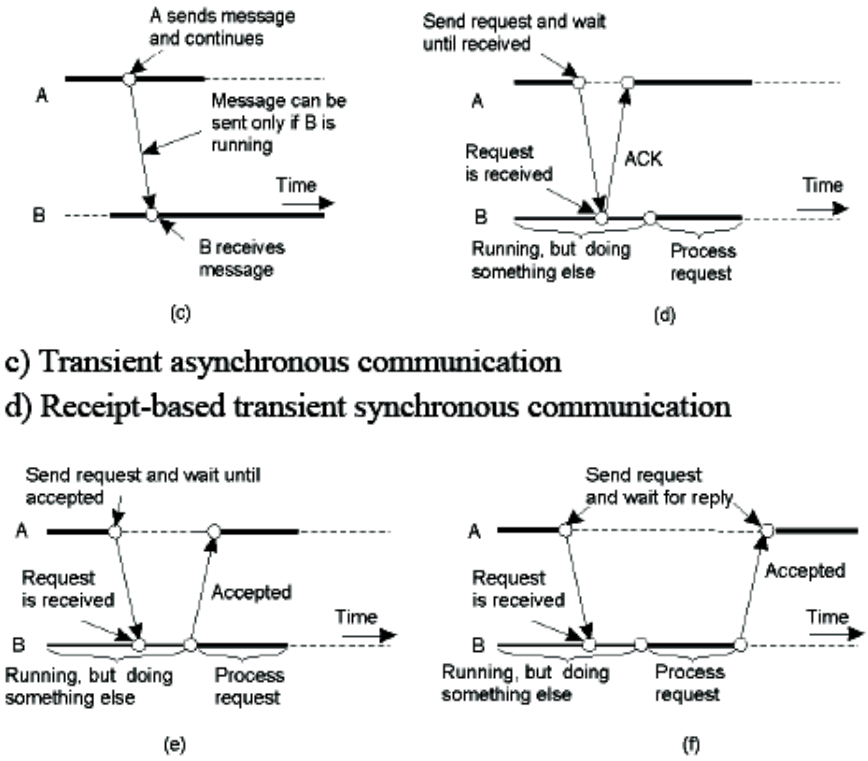
In questo tipo di comunicazione se il destinatario non è attivo la comunicazione fallisce.

(c) Asincrona: A invia un messaggio a B e continua la sua esecuzione, B quando riceve il messaggio non deve inviare nulla ad A;

(d) Sincrona Basata sulla ricezione: A invia il messaggio a B e si ferma, B invia un ACK ad A quando ha ricevuto il messaggio, quando ha tempo, esegue la richiesta;

(e) Sincrona Basata sulla consegna: A aspetta finché il messaggio non è in elaborazione da B;

(f) Sincrona Basata sulla risposta (esecuzione): A aspetta finché B non finisce di elaborare la sua richiesta.



c) Transient asynchronous communication

d) Receipt-based transient synchronous communication

e) Delivery-based transient synchronous communication at message delivery

f) Response-based transient synchronous communication

Figura 12: Comunicazione Transiente

6.3 Comunicazione orientata ai messaggi

Berkley Socket

Primitive	Meaning
Socket	Create a new communication endpoint
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection

Socket: collegamento virtuale che collega un processo su un nodo ad un altro processo su un altro nodo.

Normal Queue System

C'è un livello middleware che si occupa di trasferire e immagazzinare i messaggi da consegnare ai processi che devono comunicare.

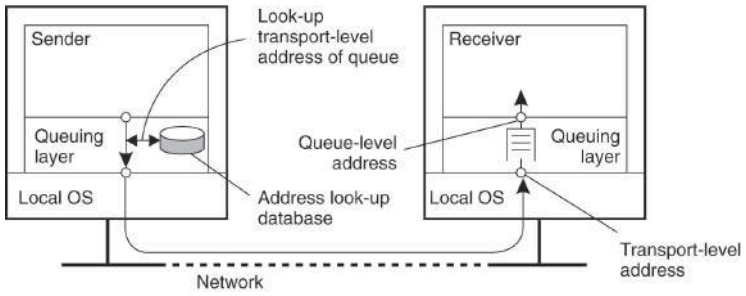
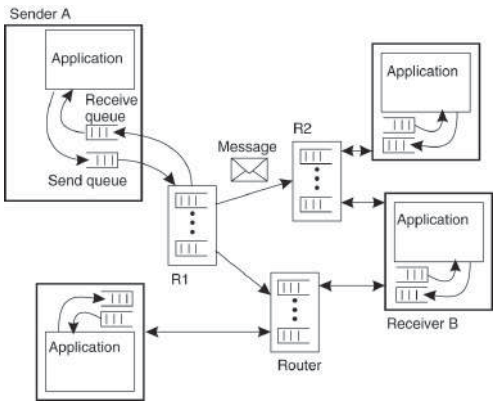


Figura 13: Queue System.

Queue System With Routers

In questo caso tra sender e receiver vi sono diversi nodi che gestiscono i messaggi. Nel caso il ricevente non è disponibile i nodi intermedi che formano il livello *middleware* immagazzinano il messaggio finché il ricevente non torna disponibile. Vi sono problemi perché c'è la necessità di indirizzare le code. La soluzione è utilizzare un identificatore per le code (*naming*).

Primitive	Meaning
Put	Append a message to a specified queue
Get	Block until the specified queue is nonempty, and remove the first message
Poll	Check a specified queue for messages, and remove the first. Never block
Notify	Install a handler to be called when a message is put into the specified queue



Message Brokers

É colui che si occupa di ridirigere i messaggi nelle code di appartenenza permettendo inoltre la conversione in un formato comprensibile ai vari nodi (anche se i due utilizzano standard differenti).

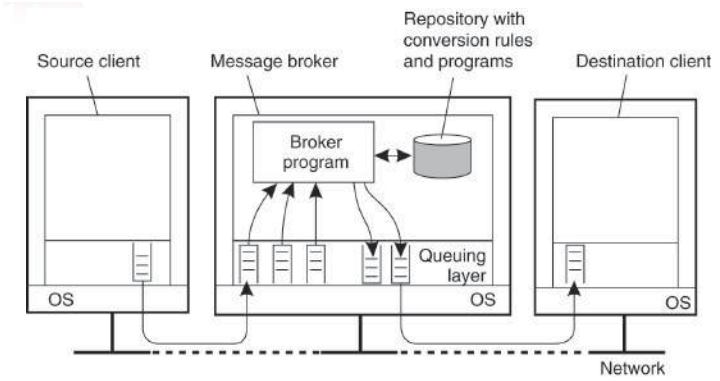


Figura 14: Message Broker.

6.4 Remote Procedure Call & Remote Object Invocation

Conventional Procedure Call

Alloca sullo stack lo spazio per i parametri necessari (in ordine inverso).

Remote Procedure Call

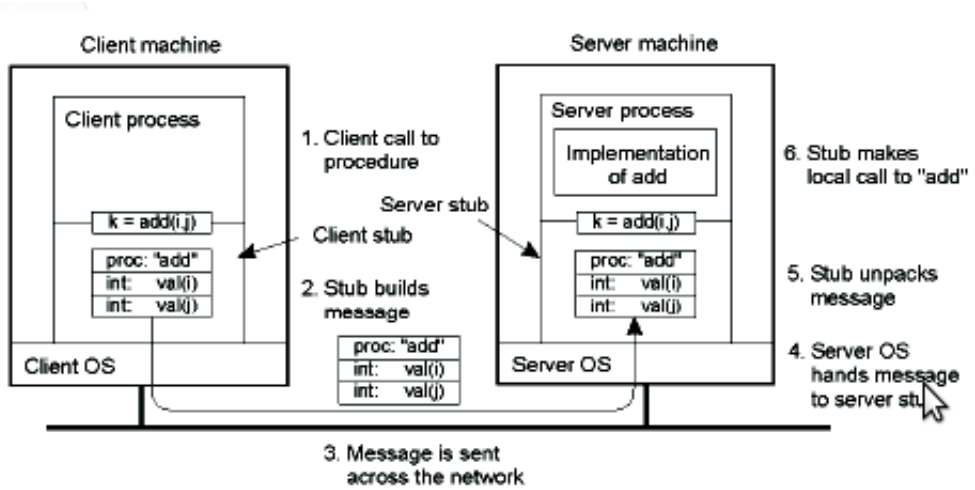
Segue il classico esempio del client server. Invece di inviare un messaggio si richiama una procedura remota sul server. L'esecuzione viene fatta lato server e una volta terminata al client viene restituito il risultato.

Stub: realizza le funzionalità di alto livello del middleware. Impacchetta permanentemente la chiamata a procedura, infine attende la risposta per spacchettarla, deve occuparsi anche della consistenza dei messaggi se si utilizzano standard differenti.

Passi di una RPC:

1. Il client richiama il suo stub (la richiesta viene impacchettata)
2. Lo stub del client richiama l'OS locale
3. L'OS del client manda il messaggio al OS remoto (TCP, IP, filo/wi-fi ecc)
4. L'OS remoto passa il messaggio al server stub (lato server)
5. Lo stub sul server spacchetta il messaggio e richiama la procedura richiesta
6. Il server esegue la procedura richiesta e restituisce il risultato al server stub
7. Il server stub impacchetta il messaggio e lo passa all'OS locale
8. L'OS server invia il messaggio all'OS client
9. L'OS del client passa il messaggio allo stub del client
10. Stub spacchetta la risposta restituendola al client

Il passaggio di parametri è semplice se fatto per valore ma diventa più complesso quando si ha la necessità di avere un riferimento ad un oggetto condiviso.



Steps involved in doing remote computation through RPC

Problema: le macchine possono interpretare i messaggi che si scambiano in modo differente soprattutto quando il passaggio di parametri avviene per riferimento.

RPC Asincrono e Sincrono

- Sincrono aspetta la risposta.
- Asincrono può aspettare l'ACK di messaggio ricevuto.

Scrivere un client e un server

Il server per pubblicare metodi remoti deve scrivere una interfaccia in IDL.
Caratteristiche di una IDL (deve essere scritta in un linguaggio neutro - esistono più compilatori in base al linguaggio che si vuole ottenere, questi compilatori, dato l'IDL creano automaticamente Server e Client Stub):

- Come si chiama il metodo
- Parametri e tipo del metodo

Remote Object Invocation

Può essere vista come una Remote Procedure Call orienteta agli oggetti. A differenza delle RPC con Remote Object Invocation si ottiene un riferimento all'oggetto che risiede sul server. Ottenuto il riferimneto è possibile invocare i metodi pubblici dell'oggetto e farli eseguire sulla macchina del client senza che il server svolga alcun lavoro. Java mette a disposizione la Remote Method Invocation (RMI) che permette l'inovocazione di metodi da remoto senza doversi occupare di Marshaling e Unmarshaling dei parametri della procedura.

Proxy e *skeleton* sono gli analoghi degli stub del client e del server. Proxy è lo stub del client e si occupa di fare richieste al server, lo skeleton è colui che impacchetta i dati e li invia al proxy. L'operazione di impacchettamento si chiama *Marshaling* e *Unmarshaling* è l'operazione inversa.

6.5 Data Streams

Stream di dati: sequenza di dati, anche i metodi visti in precedenza supportano gli stream di dati. In questo caso la comunicazione dei dati è continua ed è soggetta ad alcuni vincoli temporali.

Tre modalità:

- Asincrona i dati non hanno un vincolo temporale ma la sequenza dell'ordine dei dati deve essere mantenuta.
- *Isocrona* c'è un limite massimo e minimo di tempo in cui i dati devono essere consegnati. Es (stream di audio/video). Implementa il meccanismo di QoS nel livello Middleware, questo meccanismo controlla che:
  - I vincoli temporali vengano rispettati.
  - Il servizio sia affidabile.
  - Il volume dei dati scambiati.
- Sincrona c'è un limite massimo al tempo che viene impiegato per l'arrivo dell'unità dei dati.

QoS with Stream

Proprietà per i QoS:

- Stesso bit rate tra sorgente e destinazione.
- Vincolo massimo:
  - sul ritardo nell'instaurare una sessione della comunicazione.
  - sul ritardo di propagazione.
  - sulla varianza (jitter).
  - sul round-trip delay.

Problema degli Stream:

- Quanti dati bufferizzare per ridurre il jitter?
  - Tecnica classica utilizzata da youtube: ritardo la riproduzione a lato client per permettermi di bufferizzare una parte dello stream.
- Quanto deve essere grande il buffer?
  - Se il buffer è limitato potrei avere dei salti nella riproduzione mentre se è troppo grande dovrei aspettare troppo prima di far partire la riproduzione.

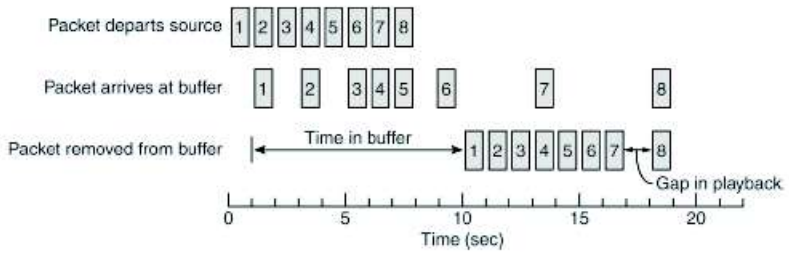


Figura 15: Bufferizzazione dello Stream

Per limitare la perdita dei pacchetti si utilizzano:

- Tecniche di controllo dell'errore, ossia delle codifiche che aiutano a ricostruire i pacchetti persi.
- Tecniche di interliving: la pacchettizzazione dello stream non avviene in modo sequenziale ma vengono mischiati.

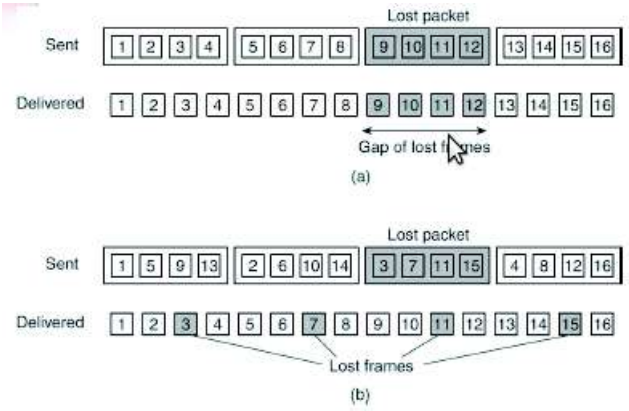


Figura 16: Interliving nei pacchetti di dati

Vantaggi: se perdessi un pacchetto invece di perdere n frame sequenziali perderei n frame in posizioni diverse (dal punto di vista utente invece di notare la perdita di una porzione del video, noterebbe solamente la perdita di alcune piccole porzioni di video, che in alcuni casi potrebbero non essere notate).

Svantaggi: devo attendere l'arrivo di tutti pacchetti per poter ricomporre la sequenza originale e iniziare la riproduzione dello stream.

Meccanismi di sincronizzazione

Se lo stream è complesso significa che è composto da più stream e questi devono essere sincronizzati.

Esempi: audio video nello stesso stream, sottotitoli di un film che devono essere temporizzati.

La sincronizzazione può essere fatta dal lato ricevente in tre modi:

1. Il programmatore si occupa della sincronizzazione,
2. Un livello middleware si occuperà di fornire le interfacce al livello applicativo e gestire la sincronizzazione degli stream,
3. Si usa una soluzione distribuita: si compone un unico stream contente uno stream per la sincronizzazione, es mpeg.

6.6 Application Level Multicast

I dati trasmessi non hanno un unico ricevente (un gruppo o tutti - in quest'ultimo caso si parla di broadcast). Per effettuare una comunicazione uno a molti è possibile utilizzare anche una Overlay Network per costruire le strutture di propagazione e inviare i messaggi. Un altro sistema per effettuare multicast è utilizzare gli algoritmi di gossip.

Part V
Naming

Nomi: Identificatori e Indirizzi

Nomi: stringhe di bit o caratteri, si dividono in:

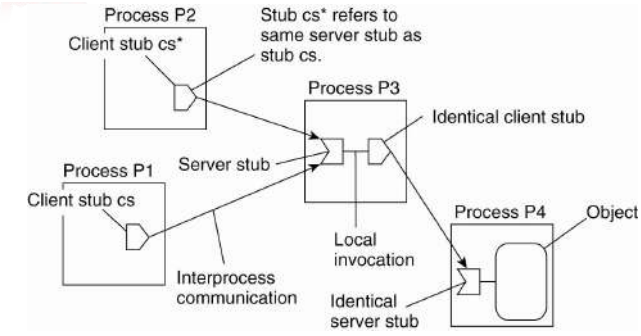
- Identificatori: un nome che ha corrispondenza biunivoca con la risorsa inoltre deve riferirsi sempre alla stessa entità (es. C. Bettini).
- Indirizzi: punto di accesso specifico ad una risorsa (es. mail o telefono).

7 Classi del sistema di Naming

- Flat - trattano nomi semplici indipendenti dalla locazione delle risorse.
- Strutturati - utilizzano nomi simbolici es DNS.
- Basati sugli attributi.

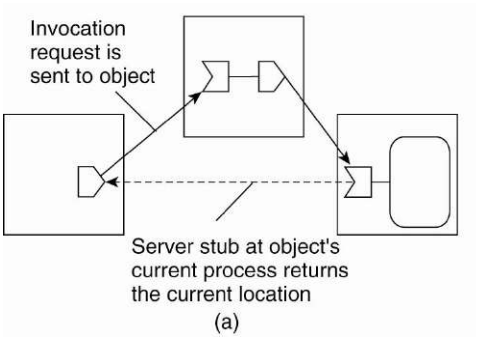
7.1 Flat Naming

Nomi senza alcun riferimento alla posizione della risorsa. Un metodo spesso utilizzato è il *Forwarding Pointers:* se la risorsa si sposta rilascia un puntatore che ne indica la nuova locazione. L'oggetto espone un interfaccia contenente metodi che lo rende accessibile agli altri processi. Il metodo Forwarding Pointers crea degli identical client e server stub fittizi perché di fatto fanno forwarding al luogo dove si trova realmente l'oggetto.



The principle of forwarding pointers using (client stub, server stub) pairs.

Figura 17: Forwarding Pointers Method



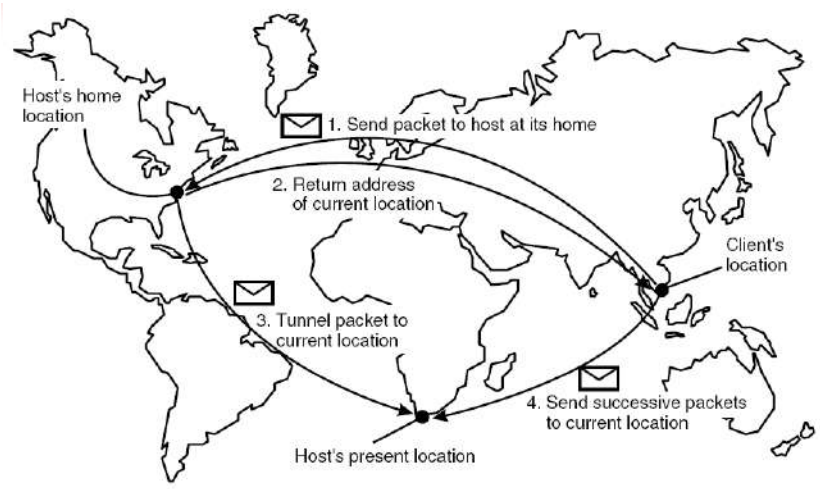
Redirecting a forwarding pointer by storing a shortcut in a client stub.

Figura 18: Forwarding Pointers Method

Il server stub finale rileva l'indirizzo di chi realmente ha effettuato la chiamata, e gli invia l'indirizzo reale dell'oggetto per evitare che esegua i vari passaggi intermedi.

Problemi: le catene possono diventare lunghe se gli oggetti sono molto mobili, se cade un nodo intermedio crolla tutto il sistema.

Approccio Home-Based



The principle of Mobile IP.

Figura 19: Approccio Home-Based

Più adatto a reti di grandi dimensioni (più scalabile), non si creano le catene viste con il *Flat Naming*. Il sender conosce solamente la locazione della Home, successivamente la Home inoltra il pacchetto direttamente al destinatario (tunneling) e restituisce l'indirizzo del destinatario al sender per eventuali altri messaggi che deve spedire. Il sender quando riceve l'indirizzo invia i messaggi direttamente al receiver.

Problemi: l'indirizzo destinatario non è stabile, se cambia perdo il messaggio; se la Home non risponde non riesco a comunicare (single point of failure) inoltre se la home è troppo distante ho problemi di efficienza.

Distributed Hash Table

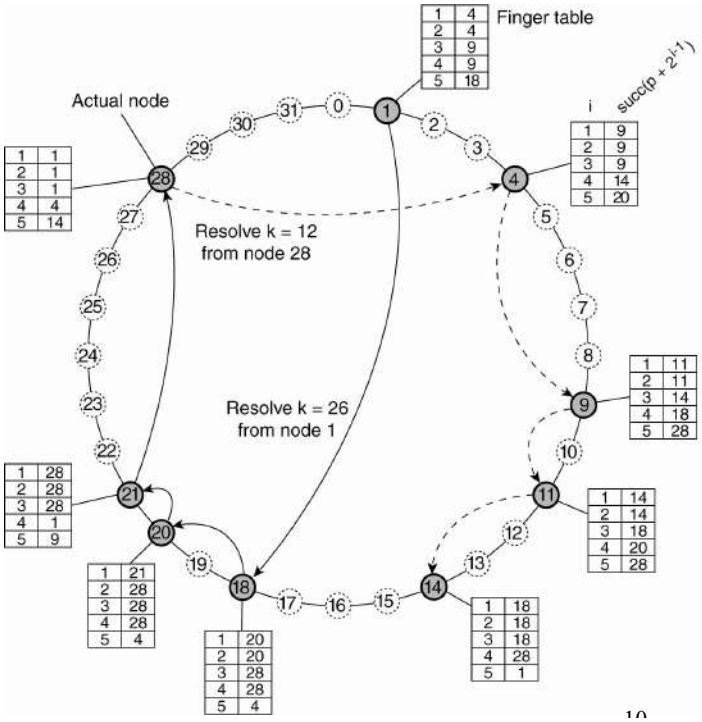


Figura 20: Funzionamento del DHT

Lookup: Metodo per trovare una risorsa. Mediante l'utilizzo delle tabelle di hash si riesce a raggiungere una particolare risorsa. Il contenuto della tabella è suddiviso in due colonne: nella prima si indicano i bit che rappresentano il massimo numero di nodi (es. noi abbiamo 5 bit quindi $2^5 = 32$ sono i possibili nodi), la seconda contiene:

$$succ(p + 2^{i-1})$$

p è l'id del nodo, i è l'elemento della finger table.

ESEMPIO: costruzione della *finger table*: nodo 4 riga 2:

$$succ(4 + 2^2 - 1) = succ(6) = 9 \qquad (il\ più\ grande\ nodo\ ATTIVO!)$$

ESEMPIO: ricerca dell'id 26; partendo da 1, controllo la seconda colonna e cerco 26 o il numero minore o uguale più grande, in questo caso trovo 18; vado al nodo 18 cerco 26 ma trovo 20 come numero minore o uguale più grande, vado al 20 e trovo 21; vado a 21 e trovo 28; vado a 28 e lui sarà l'indirizzo della risorsa che stavo cercando.

ESEMPIO: ricerca dell'id 12; partendo da 28, vado a 4, vado a 9, vado 11, trovo 14.

7.2 Structured Naming

Linking and Mounting

Quando cancello un hardlink in unix cancello completamente la risorsa mentre se cancello un simboliclink non cancello la risorsa.

Informazioni richieste per montare un *name space* esterno in un sistema distribuito:

- Il nome del protocollo di accesso.
- Il nome del server.
- Il nome del punto di mounting.

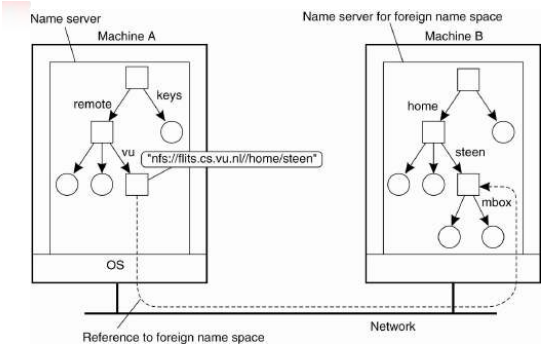


Figura 21: Abbiamo due macchine A e B. La macchina A avrà un nodo che punterà ad una porzione del grafo di B.

DNS

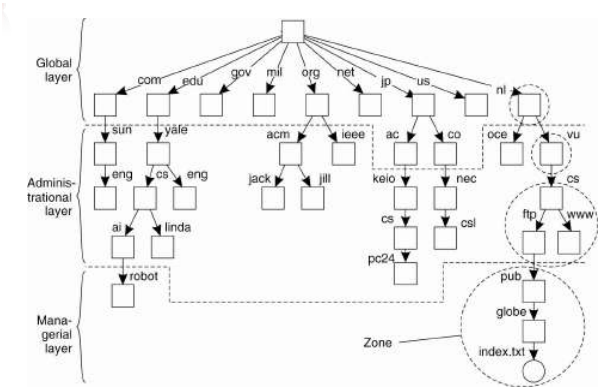


Figura 22: Schema dei DNS.

- Obbiettivo: risolvere i nomi simbolici degli host in indirizzi IP.
- Organizzazione gerarchica in un albero con radice.
- Ogni nodo ha come nome un'etichetta (55 chars)
- Pathnames: concatenazione di etichette separate dal "."
- Canonical Names (CName): sono distinti dagli aliases e devono essere risolti. Es. www.myserver.unimi.it e www.unimi.it

Tre livelli:

- Global layer: sono quelli che si diramano in tutto il mondo (com, edu, gov, it, ecc)
- Administration layer: sono specifici di alcune organizzazioni (sun, yale, unimi, ecc)
- Managerial layer: sono specifici di un specifico dipartimento (dico, dsi, ecc)

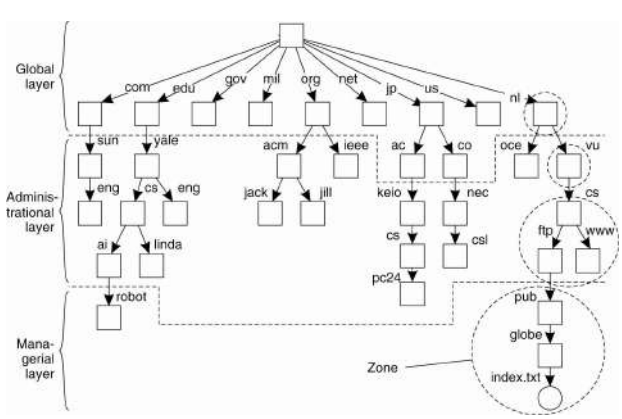
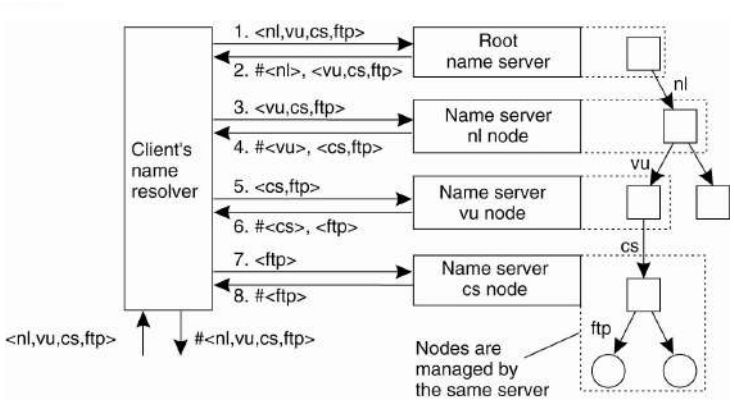


Figura 23: Differenze tra i tre livelli dei DNS.

Due metodi di risoluzione del *name*:

- Iterativo: Il server restituisce al client l'indirizzo parziale. Il client si occupa di contattare il sottoserver per risolvere il resto dell'indirizzo. Costituito da una sequenza di passi fatti tutti dal client name resolver.



The principle of iterative name resolution.

Figura 24: Implementation of Name resolution. Iterativo.

- Ricorsivo: Il server non restituisce l'indirizzo parziale del client ma è il server stesso che contatta gli altri server per risolvere l'indirizzo. La comunicazione avviene a cascata tra i server. Una volta risolto l'indirizzo lo restituisce al client.

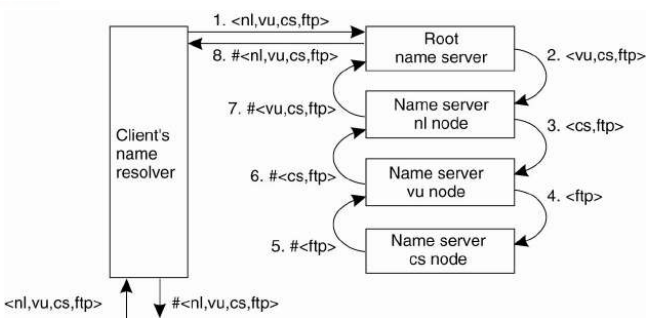


Figura 25: Implementation of Name resolution. Ricorsivo.

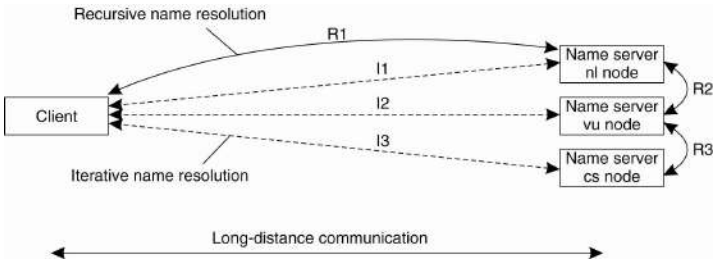


Figura 26: Differenze tra Iterativa e Ricorsiva.

Vantaggi:

Il caching con la ricorsiva funziona meglio rispetto a quella iterativa perché il caching è fatto sui server nei vari livelli. I costi di comunicazione sono inferiori nella ricorsiva (tempi di latenza maggiori con la soluzione iterativa).

Svantaggio: Il carico dei server. Nel caso della ricorsiva sono i server che eseguono un lavoro in più che potrebbe essere svolto dai client con il metodo iterativo.

Type of record	Associated entity	Description
SOA	Zone	Holds information on the represented zone
A	Host	Contains an IP address of the host this node represents
MX	Domain	Refers to a mail server to handle mail addressed to this node
SRV	Domain	Refers to a server handling a specific service
NS	Zone	Refers to a name server that implements the represented zone
CNAME	Node	Symbolic link with the primary name of the represented node
PTR	Host	Contains the canonical name of a host
HINFO	Host	Holds information on the host this node represents
TXT	Any kind	Contains any entity-specific information considered useful

Figura 27: Record DNS

7.3 Attribution-Based Naming

Problema: Come possiamo cercare un'entità in un sistema distribuito basandoci su una descrizione?
Soluzione: I directory services sono sistemi di naming che supportano la descrizione dell'entità in termini di coppie di $< attributo, valore >$

LDAP (Lightweight Directory Access Protocol)

Associa il naming strutturato con quello a coppie $< attributo, valore >$ descritto prima.
Directory Entry corrispondono ai record DNS che descrivono i singoli elementi, in questo caso le descrizioni saranno più ricche poiché fatte anche dagli utenti.
Le Directory Information Base sono collezioni delle descrizioni delle risorse.
Le informazioni possono essere raggiunte con delle combinazioni di attributi/valore

RDN

Relative Distinguished Names.

Attribute	Abbr.	Value
Country	C	NL
Locality	L	Amsterdam
Organization	O	Vrije Universiteit
OrganizationalUnit	OU	Comp. Sc.
CommonName	CN	Main server
Mail_Servers	—	137.37.20.3, 130.37.24.6, 137.37.20.10
FTP_Server	—	130.37.20.20
WWW_Server	—	130.37.20.20

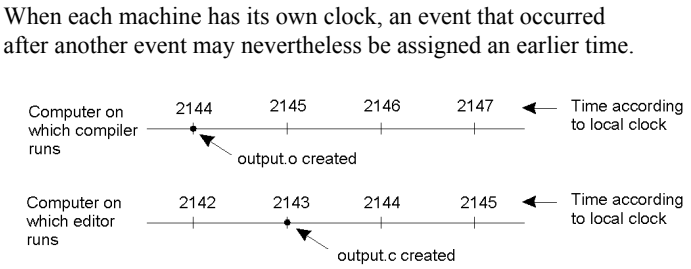
The first 5 attributes are naming attributes
(RDN = Relative Distinguished Names)

Figura 28: RDN

Parte VI
Sincronizzazione nei Sistemi Distribuiti

8 Sincronizzazione con orologi fisici

Esempio: esecuzione di una make nel linguaggio C (make: compila tutti i moduli che sono stati modificati dopo l'ultima compilazione). L'editor e il compilatore sono in due computer differenti, come si vede nell'esempio gli orologi dei due nodi sono leggermente sfasati. Al tempo 2144 viene effettuata la compilazione nel computer A e al tempo 2143 viene modificato il sorgente nel computer B. Se viene ricompilato al tempo 2146 dal computer A la modifica del file non viene riconosciuta in quanto, secondo il compilatore, la modifica del sorgente è stata effettuata prima della sua ultima compilazione, ERRORE!!!!!!!



Orologi atomici: sono basati sulla transizione dell'atomo di cesio 133.

International atomic time: misura i secondi in base alle transizioni dell'atomo di cesio, ogni 9 mld di transizioni circa conta un secondo.

Secondi intercalare o leap second: ogni tanto si aggiungono o si tolgono dei secondi senza che ce ne accorgiamo.

UCT: Universal Coordinator Time, è una media del tempo misurato dagli orologi atomici sparsi per il mondo. Se vi sono errori, le correzioni si effettuano aggiungendo o togliendo i leap second.

8.1 GPS

Misura il tempo di percorrenza del segnale dal satellite al ricevente poi, utilizzando la trilaterazione, calcola la posizione del ricevitore GPS. Su ogni satellite è presente alcuni orologi atomici perché devono mantenere un orario esattamente uguale agli altri satelliti.

Nella trilaterazione del GPS si utilizzano quattro satelliti: tre sono per latitudine longitudine e altitudine, il quarto è per mantenere sincronizzati gli altri tre satelliti.

Problemi del GPS:

1. Ci vuole un certo tempo per trasferire i dati dal satellite al ricevitore GPS.
2. L'orologio del ricevitore generalmente non è sincronizzato con l'orologio del satellite.

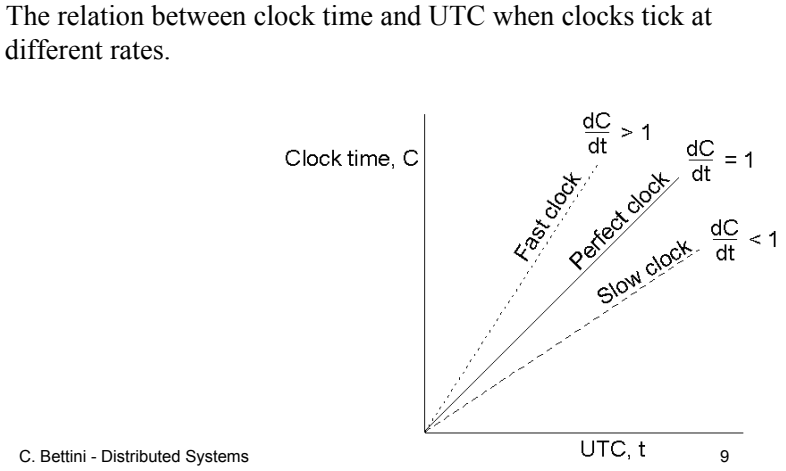


Figura 29: Algoritmi di sincronizzazione degli orologi.

8.2 Network Time Protocol

É un protocollo che si occupa di gestire le richieste di tempo da parte degli altri computer in rete. Vi sono problemi per quanto riguarda il tempo di propagazione nella rete. Questo protocollo permette di scrivere il timestamps in ogni messaggio che viaggia attraverso la rete. Il protocollo inoltre cerca di stimare il ritardo basandosi sui timestamps presenti nei messaggi scambiati, in questo modo cerca di aggiustare il tempo rendendolo più vicino a quello reale del time server.

Ogni computer che si collega alla rete può prelevare l'ora da un server che contiene il tempo esatto. Nel NTP il time server è passivo perché risponde solamente a richieste da parte degli altri nodi ma di sua iniziativa non manda messaggi a nessuno.

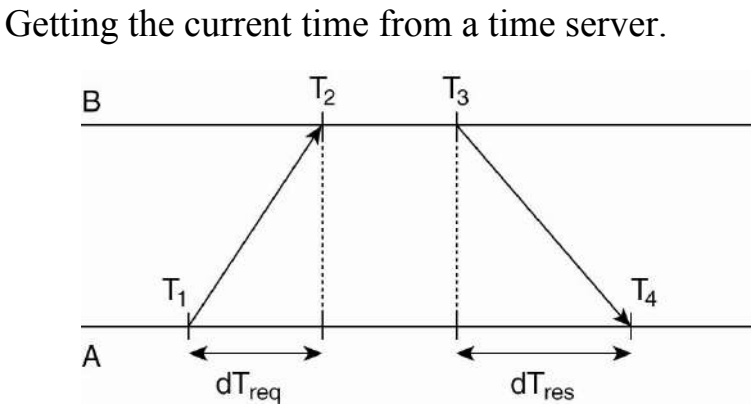


Figura 30: NTP

8.3 Algoritmo di Berkeley

Serve per sincronizzare gli orologi del sistema indipendentemente da un orologio esterno.

Funzionamento: uno dei nodi funge da coordinatore (time daemon), in questo caso il time daemon è un server attivo.

Il *time daemon* è attivo perché periodicamente manda un messaggio a tutti gli altri nodi (broadcast perché arriva anche a se stesso) ma non obbliga nessuno a sincronizzarsi. Il nodo risponde con

l'*offset* (differenza tra proprio orologio e quello ricevuto), anche il time daemon risponde a se stesso con (ovviamente) 0. Non manda l'UCT a tutti ma manda il tempo che ha il suo computer! Fa una media di tutti gli orologi presenti nella rete (utilizza questo tempo per sincronizzare tutti) e invia ad ogni nodo, compreso se stesso, lo spostamento che devono effettuare per sincronizzarsi al tempo medio. L'operazione di riportare indietro l'orario non è semplice poiché potrebbe causare problemi con il file system. La soluzione potrebbe essere quella di rallentare leggermente il clock con l'obiettivo di raggiungere il tempo medio della rete.

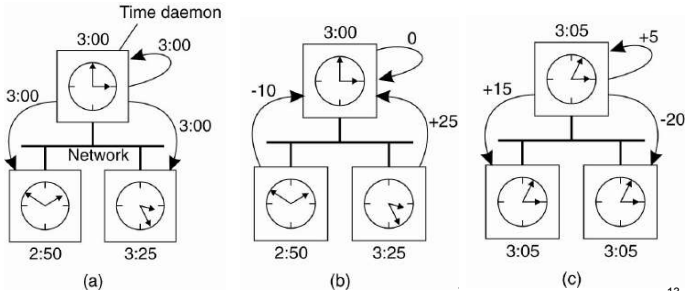


Figura 31: I vari passaggi dell'algoritmo Berkeley

9 Orologi Logici

In molti casi non si ha la necessità di sincronizzarsi con un orologio fisico esterno, può essere sufficente un ordine parziale (relazioni di precedenza).

9.1 Algoritmo di Lamport

Se a, b sono eventi, l'espressione $a \to b$ significa "a avviene prima di/precede b"
Obiettivo: assegnare un valore di questo contatore (condiviso da tutti) $C(a)$ ad un evento di invio/ricezione di un MSG in un sistema in modo che se vale $a \to b$ allora $C(a) < C(b)$
Algoritmo:

- Ogni MSG porta il timestamps (valore del contatore) assegnato dal mittente.
- Se all'arrivo del messaggio il clock del destinatario è minore di quello ricevuto, questo viene reimpostato al valore ricevuto incrementato di 1. **NB:** Vengono aggiornati anche tutti i timestamps successivi degli eventi di quel processo.

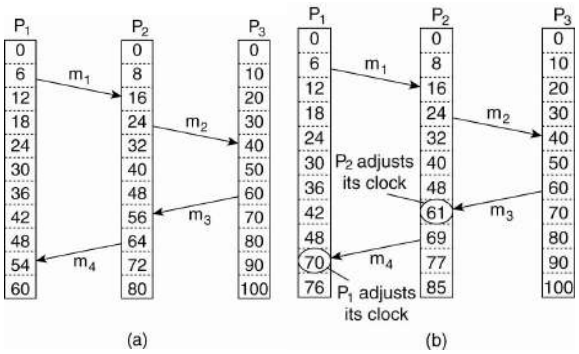


Figura 32: Esempio di esecuzione dell'algoritmo di Lamport. (a) è errato, (b) è corretto.

10 Algoritmi di mutua esclusione

Mutua Esclusione

Un insieme di processi deve accedere in modo concorrente a delle risorse condivise.
Soluzione in un singolo sistema:

- Identificazione delle regioni critiche (insiemi di istruzioni che accedendo in modo concorrente alle risorse condivise, potrebbero portare problemi di consistenza).
- Uso di semafori o monitor per garantire che quando un processo è in regione critica nessun altro possa entrarvi.

1. Algoritmo centralizzato:
   (a) Il processo 1 chiede al coordinatore il permesso di entrare in sezione critica. Il permesso viene accordato.
   (b) Successivamente, il processo 2 chiede il permesso di entrare in regione critica. Il coordinatore non risponde ma inserisce 2 nella sua coda di attesa.
   (c) Quando il processo 1 esce dalla regione critica avvisa il coordinatore, il quale controlla nella sua coda e risponde affermativamente al primo processo presente, in questo caso il processo 2.

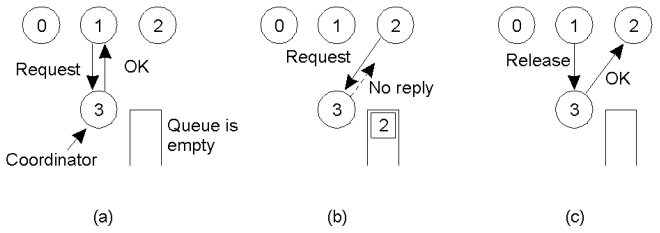


Figura 33: Algoritmo Centralizzato. I messaggi per entrata/uscita sono tre: uno per richiedere al coordinatore l'accesso, uno per la risposta del coordinatore e l'ultimo per segnalare l'uscita.

Problemi:

- Punto debole per eventuale crash del coordinatore. (Single Point of Failure)
- Collo di bottiglia per le prestazioni. (Traffico intenso su un singolo nodo)
- Difficile distinguere l'inattività del coordinatore dalla indisponibilità della regione critica. (Il coordinatore non rispondendo non fa capire se è caduto o se mi ha messo in attesa)

Soluzione nei sistemi distribuiti:

1. ALGORITMO DI RICART E AGRAWALA (1981)
   Assunzione: ordine totale tra gli eventi del sistema e consegna dei messaggi affidabile (sistema di ACK)
   (a) Se un processo P vuole entrare in una regione critica (RC) costruisce un MSG che contiene:
      i. il nome della regione critica
      ii. l'ID di P

iii. il tempo attuale

E manda il MSG a tutti (incluso se stesso).

(b) Quando un processo Q riceve un MSG si distinguono 3 casi:

i. Se Q non è in RC e non vuole entrarci risponde con un OK a P.

ii. Se Q è in RC non risponde e accoda la richiesta. (P deve ricevere da tutti i nodi l'ok per entrare nella RC)

iii. Se Q vuole entrare in RC ma non l'ha ancora fatto confronta il timestamps di MSG con quello del MSG che lui ha mandato a tutti gli altri. Il minore vince. Se quello inviato da P è il minore risponde OK a P, altrimenti accoda il messaggio ed entra in RC.

(c) Il processo P dopo aver inviato il MSG aspetta l'OK da tutti per poi entrare in RC.

(d) Quando P esce da RC manda MSG di OK a tutti i processi nella propria coda di attesa svuotandola.

Esempio:

(a) I processi 0 e 2 vogliono entrare nella regione critica allo stesso momento. 0 ha timestamps 8 e 2 ha timestamps 12. Sono diversi perché è totalmente ordinato.

(b) Il processo 1 da ad entrambi l'ok.

(c) Il processo 2 confronta i timestamps con quello di 0; 0 ha il timestamps più basso quindi gli invia l'ok.

(d) Il processo 0 confronta i timestamps con quello di 2; 0 ha il timestamps più basso, quindi accoda 2 e aspetta tutti gli OK.

(e) Quando il processo 0 ha completato, manda anch'esso un OK in modo che il processo 2 possa entrare nella regione critica.

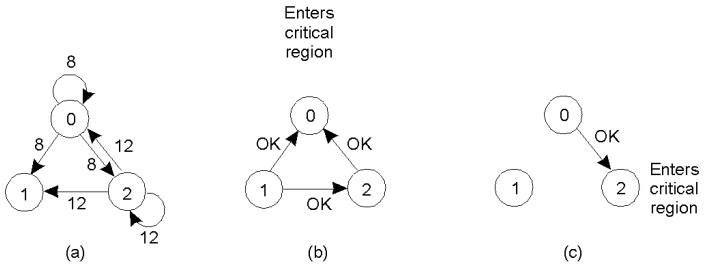


Figura 34: Algoritmo distribuito

Problemi:

- La mancata risposta di un processo può anche essere dovuta a terminazione imprevista dello stesso.
- Coinvolgere tutti i processi del sistema può essere uno spreco di risorse.

1. Algoritmo ad anello

   (a) Un gruppo non ordinato di processi su una rete.

   (b) Un anello logico costruito via software.

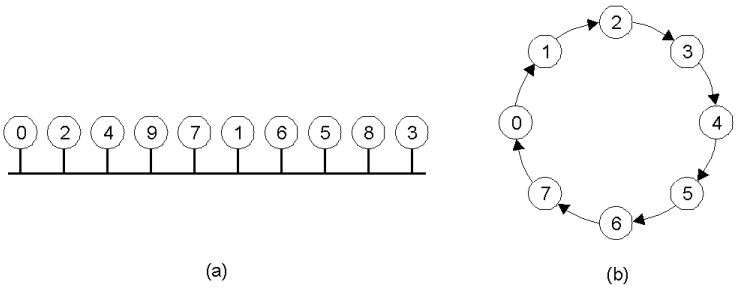


Figura 35: Algoritmo ad anello

Funzionamento:

(a) All'inizio il token viene dato al processo 0.

(b) Viene poi passato con un messaggio dal processo i al processo $i+1$ (modulo n).

(c) Se un processo è interessato, all'arrivo del token entra nella regione critica e solo quando vi esce passa il token. Non può utilizzare due volte lo stesso token.

Algoritmo	**Messaggi per entrata/uscita**	**Ritardo prima dell'entrata (in n. di messaggi)**	**Problemi**
Centralizzato	3	2	Crash del coordinatore
Distribuito	2 (n – 1)	2 (n – 1)	Crash di qualsiasi processo
Distribuito ad anello	1 to ∞	0 to n – 1	Perdita del token, crash di un processo

Figura 36: Confronto dei tre algoritmi

11 Algoritmi di elezione

Elezione di un coordinatore

- Coordinatore utile in molti algoritmi con gestione centralizzata (es. accesso ad una risorsa).
- In un sistema distribuito i processi si distinguono per un identificatore univoco (es. ID = indirizzo di rete e numero del processo).
- Gli algoritmi di elezione tipicamente cercano di eleggere il processo con l'ID più alto.

- In alcuni algoritmi si suppone che tutti i processi conoscano l'ID degli altri. Non sanno comunque se i processi siano attivi o meno.

1. ALGORITMO DI BULLY

   (a) Il processo 4 contatta il vecchio coordinatore, si accorge che non è più attivo e indice un'elezione inviando un messaggio a tutti i processi con ID maggiore del proprio (5, 6 e 7, non a 1, 2 e 3).

   (b) I processi 5 e 6 (se sono attivi) rispondono dicendo a 4 di fermarsi (MSG OK).

   (c) Ciascuno dei processi 5 e 6 indice un'elezione (stesso algoritmo dal punto (a)).

      i. 5 invia a 6 e 7; 6 invia solo a 7;

      ii. 7 non risponde perché caduto;

   (d) 5 riceve OK da 6; 6 non riceve messaggi;

      i. Scaduto il timeout 6 capisce di essere lui il coordinatore;

   (e) 6 dice in broadcast di essere lui il nuovo coordinatore.

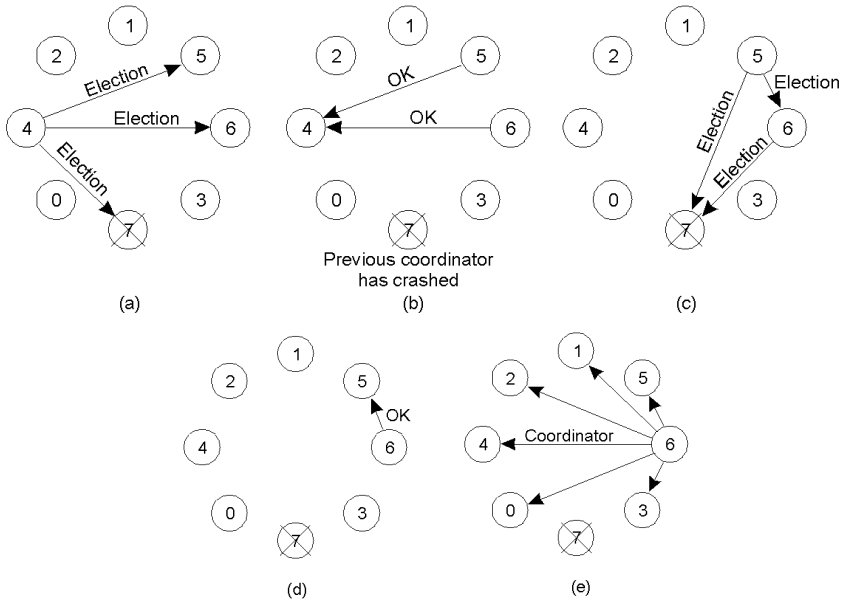


Figura 37: Algoritmo Bully

2. Un algoritmo ad anello (usato se sono tantissimi nodi e non ho la conoscenza globale della rete):

   Assunzione: Processi ordinati logicamente in un anello in modo che ciascun processo conosca come comunicare con il proprio successore (e, se questo non è attivo, con il seguente)

   (a) Quando un processo P nota che il coordinatore non risponde indice un'elezione inviando al successore un MSG di tipo *ELECTION* che contiene il proprio ID. (Se il successore non è attivo usa il primo attivo nell'anello).

   (b) Quando un processo riceve un MSG di elezione aggiunge il proprio ID e lo fa proseguire.

(c) Quando il MSG ritorna a P, questi cambia il tipo di messaggio in *COORDINATOR* e lo fa ricircolare.

(d) Alla ricezione di un MSG *COORDINATOR* ciascun processo capisce che il coordinatore è quello con ID maggiore tra quelli nella lista allegata al messaggio e che gli altri sono gli attuali partecipanti all'anello.

Part VII
Fault Tolerance

12 Concetti base

- Dependability (fedeltà):
  - Availability. Misura la *disponibilità* del sistema ossia la probabilità che il sistema funzioni correttamente in un determinato momento. Un sistema ad alta disponibilità può non essere molto affidabile per eventuali reboot giornalieri.
  - Reliability. Misura la *affidabilità* ossia l'abilità di funzionare correttamente per un lungo periodo di tempo. Uptime: tempo trascorso dall'ultimo boot del sistema. Un sistema con alta reliability può non avere alta disponibilità, es uptime di 11 mesi ma per un mese è down.
  - Safety. Non è sicurezza. Un fallimento parziale non deve portare al fallimento totale del sistema.
  - Maintainability. La capacità e la facilità di riparare il sistema.

13 Modelli di fallimento

- Crash Failure: Un server si ferma, ma funzionava correttamente finché non si è fermato.
- Omission Failure:
  - Un server fallisce nel rispondere alle richieste.
  - Un server non riceve i messaggi che gli dovrebbero arrivare.
  - Un server non invia i messaggi.
- Timing Failure: Una risposta del server viene inviata in ritardo.
- Response Failure:
  - Una risposta del server non è corretta.
  - Il valore di una risposta è errato.
  - Il server devia dal corretto flusso di lavoro.
- Arbitrary failure: Un server può produrre risposte arbitrarie in momenti casuali.

14 Ridondanza come mascheramento dei fallimenti

- Information redundancy: extra bit sono aggiunti per correggere e ricostruire gli errori dovuti al rumore sul canale.
- Time redundancy: i malfunzionamenti potrebbero essere temporanei. Riprovo a fare un'altra volta la stessa operazione. Una transazione viene ripetuta se un operazione nella transazione non viene eseguita correttamente.
- Physical redundancy: extra hardware o software (processi) vengono aggiunti per ripristinare un fallimento di un componente. Es. RAID di hard disk.

Un sistema si dice *k-fault tolerance* se continua a funzionare correttamente anche se k componenti smettono di funzionare.

14.1 Process Resilience (elasticità, capacità di recupero)

La ridondanza si applica utilizzando gruppi di processi identici.

- Ciascun processo riceve i messaggi in ingresso.
- Un protocollo per gestire l'entrata e l'uscita di un processo nel gruppo.
- Quanti ridondanza mettere?
  - Crash Failure: in questo caso sono sufficienti $k+1$ processi per fornire *k-fault tolerance*.
  - Valori errati dai processi: in questo caso sono sufficienti $2k+1$ processi per fornire *k-fault tolerance*. Si può utilizzare un sistema Voter in cui la maggioranza vince.

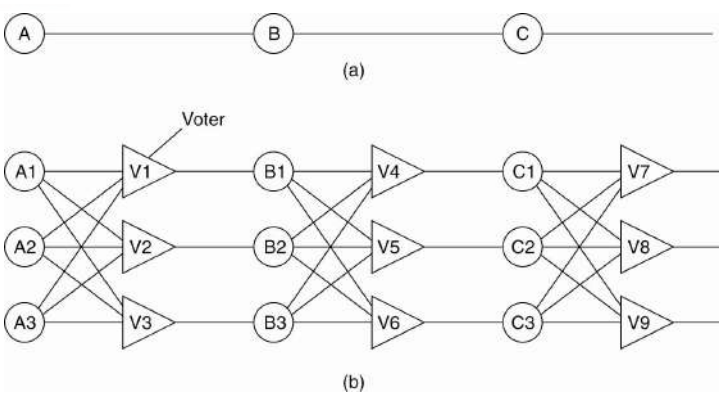


Figura 38: Sistema di Voter

- Consenso: i processi del gruppo devono mettersi d'accordo su chi di loro ha ragione. É un problema non sempre risolvibile. Vi sono vari algoritmi ma funzionano solo sotto determinate condizioni. Esempio dei generali bizantini.

Parte VIII
Sicurezza

15 Minacce e meccanismi di protezione

Minacce

- Intercettazione: accesso non autorizzato alle informazioni.
- Interruzione: servizi o dati non più accessibili. Esempio: DOS Attacks.
- Alterazione: cambiamenti non autorizzati a dati o servizi.
- Generazione: evoluzione della alterazione, invece di modificare i dati vengono generati. Generazione di dati o attività che normalmente non esisterebbero.

Servizi

- Riservatezza: protegge i dati trasmessi da un eventuale attacco.
- Autenticazione: servizio che permette di identificare utenti e processi.
- Integrità: ci protegge contro l'alterazione. Il messaggio deve essere lo stesso tra mittente e destinatario.
- Non-ripudio: impedisce al mittente di negare che lui ha trasmesso quel messaggio.
- Controllo dell'accesso: in base a come l'utente/processo si è autenticato gli metto a disposizione differenti servizi/risorse secondo le politiche fissate.
- Disponibilità: sono contromisure alle minacce di Interruzione come i DOS.

Meccanismi

La crittografia è alla base di vari meccanismi per la sicurezza.

- Cifratura: permette l'implementazione di canali sicuri garantendo la riservatezza. Esistono due metodi di crittografia:
  - A chiave *simmetrica* (esiste una chiave che viene condivisa tra i partecipanti alla comunicazione).
  - A chiave *asimmetrica* (esistono due chiavi per ogni partecipante, il mittente del messaggio prende la *chiave pubblica* del destinatario e cripta il messaggio; in questo modo solo il destinatario che possiede la sua chiave privata è in grado di decodificare il messaggio).

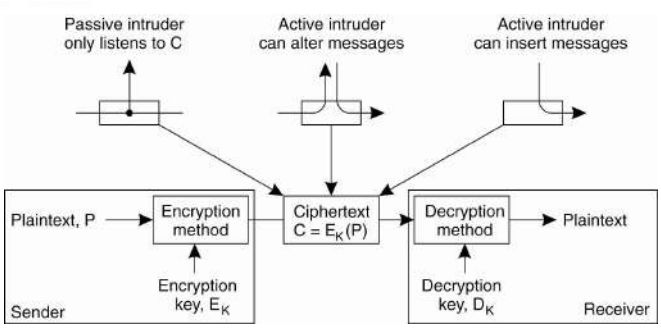


Figura 39: Crittografia.

- Firma digitale: permette l'autenticazione delle parti.
- Hashing: l'utilizzo di checksum o funzioni di hash permette di mantenere l'integrità dei dati.
- Controllo dell'accesso: permette di concedere o negare un'operazione su una risorsa da parte di un processo. Può richiedere un meccanismo per valutare regole complesse.
- Auditing: Usato per tener traccia degli accessi effettuati alle risorse da specifici utenti/processi; utili per analisi (es. intrusion detection) e pianificazione di contromisure. Può anche essere utilizzato a fini statistici per prevedere il comportamento normale del sistema. In questo modo comportamenti anomali sono un segnale di comportamento sospetto. É possibile modellare anche i comportamenti degli attacchi ma non è sempre efficace; infatti non sono riconosciuti attacchi nuovi.

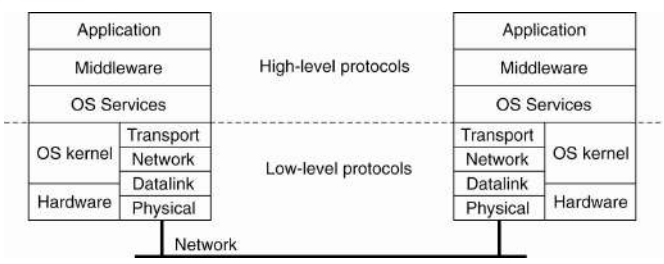
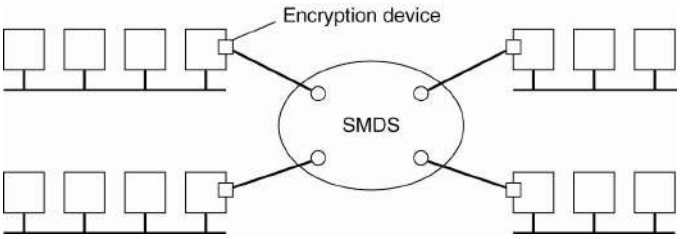


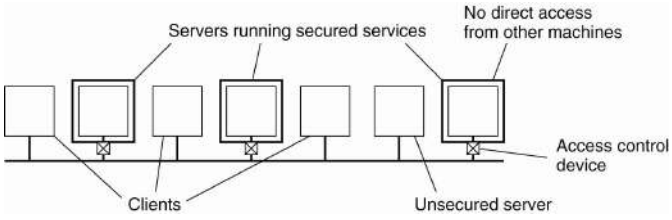
Figura 40: Nel livello Midlleware vengono inseriti questi meccanismi.

Esempio



Il sistema garantisce la sicurezza della comunicazione
tra siti ma considera "trusted" ciascun sito

Figura 41: Esempio di una possibile implementazione di un sistema di sicurezza. I nodi interni sono considerati fidati e quindi non vengono controllati; viene controllato solamente la comunicazione tra le varie reti periferiche.



I servizi che devono essere messi in sicurezza sono installati su
nodi del sistema con specifiche caratteristiche.

Figura 42: Esempio di protezione di una rete locale in cui non esiste certezza che gli altri sistemi siano sicuri.

16 Comunicazione di gruppo sicura: Kerberos

Canali sicuri e autenticazione:

1. Un canale sicuro protegge mittenti e destinatari dalle intercettazioni, alterazioni e generazioni di messaggi.
2. Autenticazione e garanzia di integrità sono tipicamente entrambi necessari. Si utilizzano vari protocolli basati sulla crittografia per garantire queste due proprietà.

- Kerberos implementa entrambi i punti precedenti ed è:
  - Sistema largamente usato per costruire canali sicuri tra client e server in un sistema distribuito.
  - Basato su protocolli a chiave segreta.
  - Utilizza un servizio di autenticazione e distribuzione di "ticket" per la comunicazione con i server. Il ticket ha una scadenza.

– Utilizza una chiave simmetrica per la comunicazione tra lui e il client.

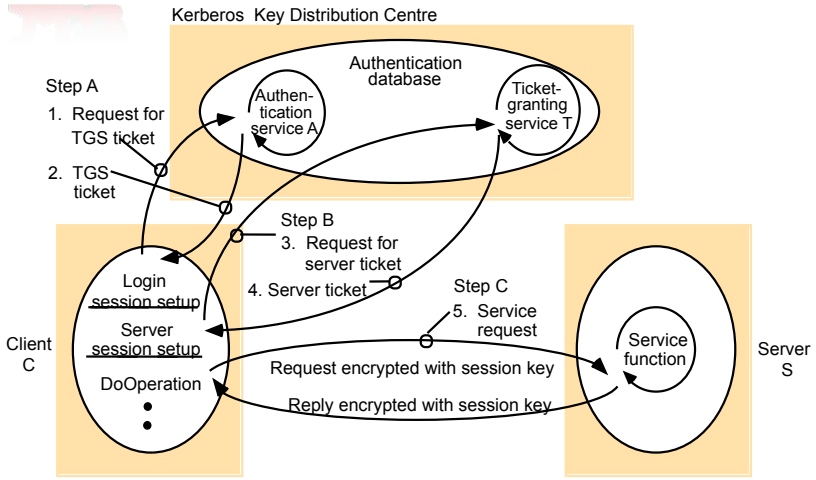
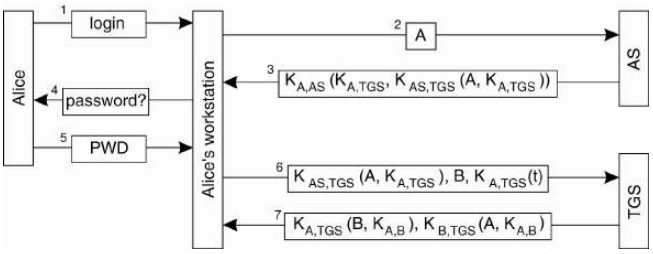


Figura 43: Kerberos.

- Abbiamo tre componenti:
  - Kerberos Key Distribution Center, ha due parti principali che svolgono due funzioni:
    - AS - Authentication Server (Autentica gli utenti)
    - TGS - Ticket Grant Server (il bigliettaio - distribuisce i biglietti)
  - Client C
  - Server S
- Passi:
  1. Richiesta per il TGS ticket
  2. Ottiene il TGS ticket
  3. Richiede il server ticket
  4. Ottiene il server ticket
  5. Richiesta del servizio
  6. Ottiene il servizio
- Fasi:
  - Client C vuole parlare con il server S
  - C richiede un biglietto a Kerberos mandandogli il nome utente
  - Kerberos (AS) controlla, tramite il suo database interno, che il client che si vuole loggare sia veramente C; nel database ci sono tutti gli utenti che si sono precedentemente registrati al servizio Kerberos.
  - Kerberos (AS) restituisce un ticket a C che gli permetterà di comunicare con Kerberos (TGS). Questo messaggio è cifrato con una chiave che solamente C possiede.

– C contatta il TGS, dicendogli che vuole parlare con S. Il TGS consegna un biglietto a C con scadenza. In questo modo ho la garanzia che C sia veramente C e S sia veramente S. Ciò che invierò a S è cifrato.
– Tra C e S si utilizzano chiavi di sessione, si utilizzano solo una volta e non sono più valide per una eventuale futura comunicazione tra C e S.



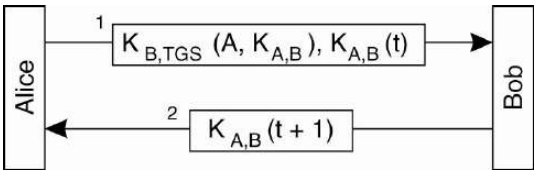


Figura 44: Autenticazione in Kerberos. Il punto 6 è la seconda immagine.

Legenda:
K = chiavi;
AS = Authentication Server;
TGS = Ticket Grant Server;
Fasi:

1. Login dell utente (solo username).
2. Il processo client contatta AS comunicandogli l'username, A vuole parlare con TGS.
3. AS risponde con un messaggio cifrato da una chiave simmetrica che conoscono solamente lui e il client: nel messaggio vengono inseriti una chiave e un'informazione crittografata in modo tale che sia decifrabile solo da AS e TGS

   - $K_{A,AS}$ = è la chiave simmetrica condivisa da A e AS, A con la sua PWD riesce a decifrare il messaggio codificato con questa chiave.
   - $K_{A,TGS}$ = è la chiave di cifratura che permette ad A di comunicare con TGS.
   - $K_{AS,TGS}(A, K_{A,TGS})$ = è un messaggio cifrato con la chiave $K_{AS,TGS}$, A non può capirlo!
   - PWD = è la password dell'utente, viene utilizzata per decifrare i messaggi che gli invia Kerberos. Solo quando Alice si è autenticata nella sua workstation può decifrare il messaggio.

4. A questo punto A capisce il messaggio che gli è stato inviato e costruisce il messaggio con TGS.

- $K_{AS,TGS}(A, K_{A,TGS})$ = non essendo riuscito a capirlo lo passa a *TGS*, questo messaggio assicura a TGS che dall'altra parte del canale ci sia veramente A che sta parlando e non un altro utente.
- B = il destinatario/server/Bob/B.
- $K_{A,TGS}(t)$ = chiave di sessione per parlare col *TGS,* .

5. *TGS* risponde:

   - $K_{A,TGS}(B, K_{A,B})$ = contiene la chiave che serve per comunicare con *B,* è cifrata con la chiave $K_{A,TGS}$ che si sono scambiati prima. É quella con (t) di prima!
   - $K_{B,TGS}(A, K_{A,B})$ = contiene la chiave che B potrà usare per comunicare con A. A non riesce a capirlo!

6. A è compiaciuto: ha tutte le chiavi che vuole, ma B ancora non ha sentito nulla!

   - $K_{B,TGS}(A, K_{A,B})$ = non essendo riuscito a leggerlo lo inoltra a *B.* Questo è il Ticket che ha consegnato TGS ad A per poter parlare con B!! Ora siamo sicuri che A è A e B è B.
   - $K_{A,B}(t)$ = chiave di sessione temporanea che condividono *A e B.*

7. B è contento che A lo abbia contattato e gli da un bel ACK ($K_{A,B}(t+1)$). Ora possono comunicare tranquillamente con la chiave $K_{A,B}$. Naturalmente la versione nuova di Kerberos è differente in qualche piccolo aspetto ma "il meccanismo" sarà più o meno simile.

17 Controllo degli accessi

Matrice per il controllo degli accessi

- M[s,o] elenca le operazioni alle quali il soggetto s è abilitato sull'oggetto o
  - Esempio: metodi su un oggetto richiamabili da un certo processo
- La matrice è tipicamente sparsa e di dimensioni considerevoli $\Longrightarrow$ implementata in modo compatto:
  - ACL (Access Control List) associata ad ogni oggetto: contiene per ogni utente i permessi su quello specifico oggetto

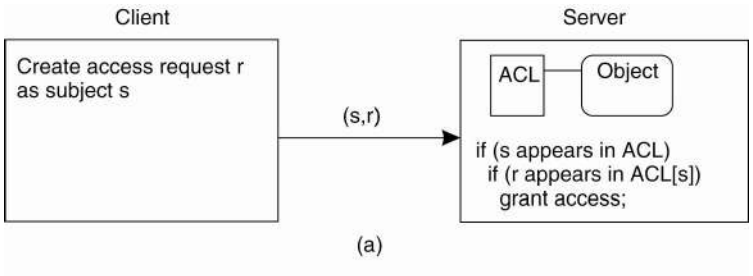


Figura 45: ACL

– Capability list per ogni soggetto

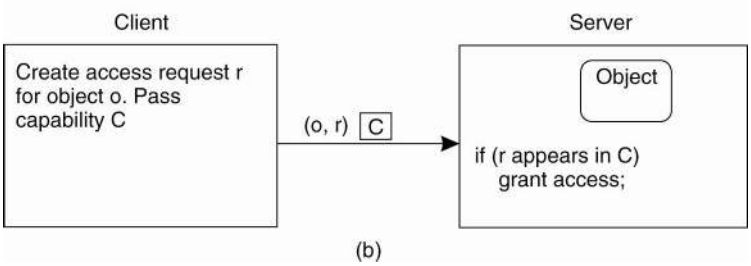


Figura 46: Capability

Domini di protezione

Un modo per ridurre le ACL: organizzazione gerarchica dei domini di protezione come gruppi di utenti.

Controllo dell'accesso basato sui ruoli

- Controllo dell'accesso basato sui ruoli
- Altro modo di sfruttare i domini di protezione: considerare i ruoli dei soggetti
- Un soggetto può avere diversi ruoli (esempio: studente di UniMi, consigliere comunale, presidente di una associazione)
- A seconda del ruolo con il quale ci si connette si determina il dominio di protezione (e i privilegi)
- E' possibile cambiare ruolo

Estensioni dei modelli per il controllo dell'accesso

- Accesso condizionato
  - per esempio, solo in determinati intervalli di tempo, solo con specifici parametri, ecc...
- Granter e grant-option
  - con delega di concedere privilegi (e specifica di chi ha concesso il privilegio).

Part IX
DISTRIBUTED FILE SYSTEM

18 Obiettivo

Rendere disponibile all'utente un filesystem virtuale che offre interfacce simili al filesystem locale ma opera su file distribuiti su diverse macchine. I Distributed File System sono presenti da molto tempo (NFS). Indipendentemente da come sono implementati devono offrire un FILE SERVICE per l'accesso e la gestione di file remoti.

19 Requisiti dei file service

- TRASPARENZA:
  - Accesso: Stesse interfacce sia in locale che in remoto. Modalità di accesso uniforme
  - Locazione: Stesso *name space* dopo la rilocazione dei file o processi.
  - Mobilità: Rilocazione automatica dei file.
  - Performance: Devono essere le medesime dei file system locali.
  - Scalabilità: Il sistema scala in base ai nodi che ci sono nel sistema.
- CONCORRENZA:
  - Isolation: Eseguire due operazioni in modo concorrente danno il medesimo risultato rispetto all'esecuzione delle stesse operazioni in modo sequenziale.
  - Meccanismi di mutua esclusione:
    - *File-Level locking*: il lock è sull'intero file del file system.
    - *Record Level locking*: il lock è solo su una parte (record) di quel file.
  - Minimizzare il conflitto tra le richieste dei file: Con *Record-Level* vi sono minori contese in quanto non considero tutto il file.
- REPLICAZIONE:
  - I File Service mantengono copie identiche dei file per:
    - Bilanciare il carico su più server rendendo il servizio più scalabile.
    - Avendo i server più vicini si percorrono meno tratte della rete migliorando la velocità e diminuendo i problemi.
    - Fault Tolerance
  - La piena replicazione è difficile da implementare causa problemi di consistenza.
  - La soluzione è il caching di una o più parti del file (non da benefici alla *fault tolerance*)
- ETEROGENEITÀ:
  - I Servizi devono essere accessibili da diversi OS.

  - Devono essere compatibili con i diversi OS.
  - Le interfacce devono essere open ossia le precise specifiche delle API devono essere pubblicate.

- FAULT TOLERANCE:
  - Gli errori e i crash lato client devono essere tollerati dai servizi. Se un client crasha, il servizio deve continuare ad operare.
    * at-most-once semantics: richieste multiple generano una sola richiesta.
    * at-least-once semantics: garantisce che l'operazione venga svolta almeno una volta.
      · richiede operazioni idempotenti.

      É difficile ottenere contemporaneamente le due semantiche (si parlerebbe di exactly-one).
  - Dopo un crash della macchina Server il servizio deve ripristinarsi.
  - Se il servizio è replicato può continuare ad operare anche in caso di crash di un server.

- CONSISTENZA:
  - Il problema della consistenza dei file si ha quando si effettua un operazione di caching infatti potrei avere inconsistenze tra quello che ho scritto in cache e quello che c'è sul file system.

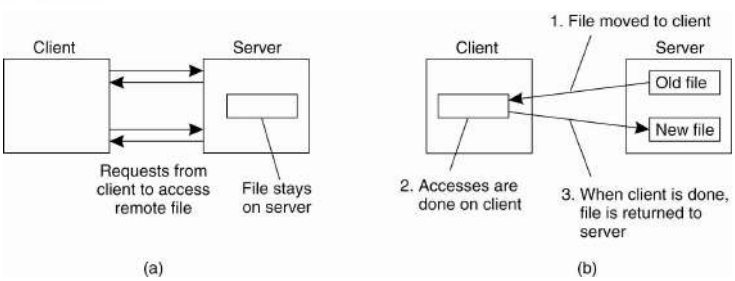
- SICUREZZA:
  - Il DFS deve implementare un sistema di sicurezza per garantire la lettura e la scrittura.
  - Bisogna mantenere il controllo sull'accesso come se fossimo in locale:
    * basato sull'identità dell'utente che fa richiesta;
    * le identità degli utenti remoti devono essere autenticate;
    * occorre una comunicazione sicura.

- EFFICIENZA:
  - Il sistema deve funzionare come se fossimo in un sistema locale!

19.1 Tre architetture

- Client Server (NFS, AFS, SMB, CIFS, Coda)
- Cluster-based (GFS - Google File System, Yahoo)
- Symmetric (P2P, Ivy)
  - Soluzione totalmente decentralizzata.

19.2 Client Server



(a) The remote access model.

(b) The upload/download model.

Figura 47: Client Server Architetture.

NFS

Supporta:

- trasparenza
- eterogeneità
- efficienza
- fault tolerance

Limitazioni:

- Concorrenza
- Replicazione
- Consistenza
- Sicurezza

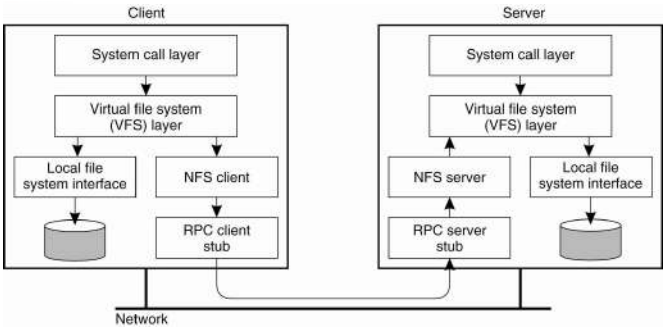
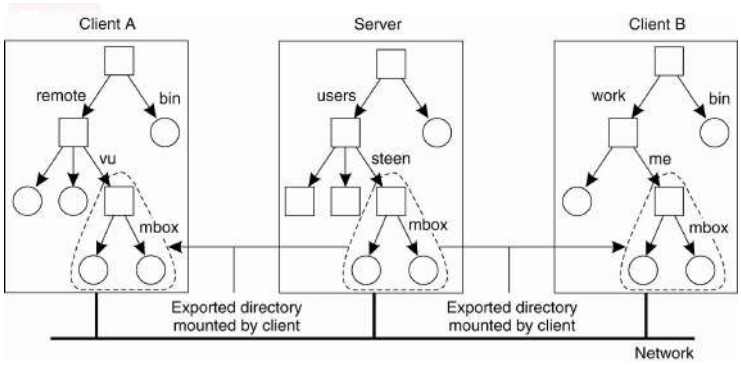


Figura 48: VFS: Virtual File System, è un livello aggiuntivo che astrae il file system. Il NFS sfrutta il livello VFS per rendere le operazioni sui file trasparenti. La comunicazione tra le due architetture può avvenire tramite RPC o Socket.

DFS Naming Schemes

1. *Mount:* monta cartelle remote nel file system locale, dando l'apparenza di un albero di directory locale coerente.
   - L'accesso alle cartelle remote montate può essere trasparente
2. I file sono nominati dalla combinazione di *host name* e *local name*
   - Garantisce un nome univoco al livello di sistema
   - Windows Network Places
3. Un unico nome identificatore del file
   - Un singolo *global name structure* per tutto il file system
   - Se un server non è disponibile alcune directory potrebbero essere inaccessibili



Mounting (part of) a remote file system in NFS.

Figura 49: Mounting di una directory remota. Montare una parte di file system su un nodo già occupato comporta la sostituzione temporanea del sotto albero precedente.

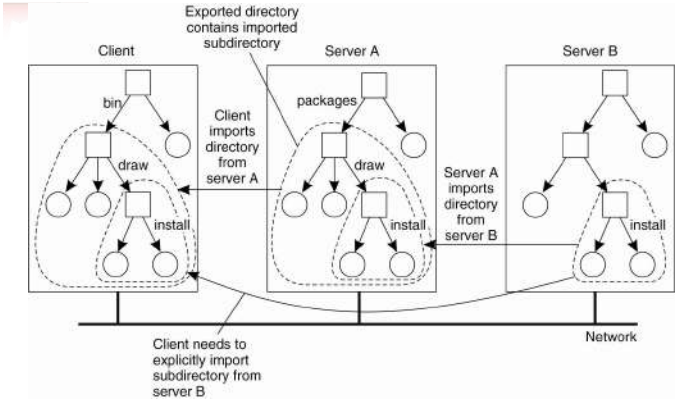


Figura 50: Montare directory innestate da più server.

File Locking

- LOCK Crea un lock per un range di bytes
- LOCKT Controlla se ci sono lock già concessi sulla risorsa
- LOCKU Rimuove un lock da un range di bytes
- RENEW Richiede un nuovo tempo limite per tenere il file

Client-Side Caching

In NFSv4 si usano le call-back: il server concede al client un file, se il server richiede la restituzione del file, il client deve restituire il file.

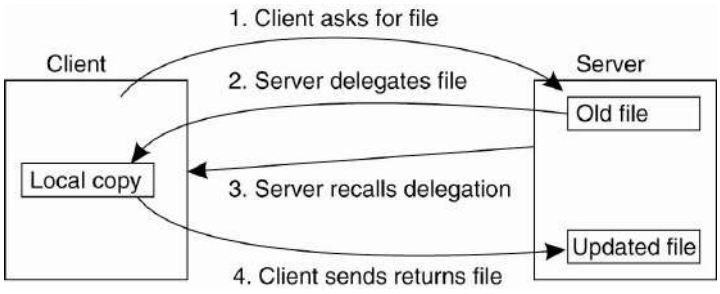


Figura 51: Client-Side caching.

Cluster-Based DFS

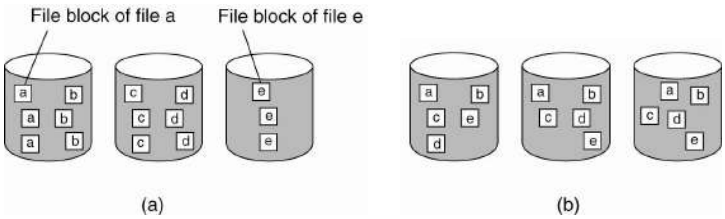


Figura 52: Cluster-Based DFS.

1. Due possibili implementazioni:
   (a) Un file su unico nodo
   (b) Lo stesso file distribuito su più parti (Striping); Vantaggi:
      - Throughput
      - Accesso al file più veloce perché l'accesso è parallelizzato. (es Torrent).

19.3 GFS - Google File System

Motivi:

- Gestione di dati grandi.
- Tante macchine (e tanti failure).
- I file sono usati prevalentemente in lettura o al massimo aggiungo qualche parte.
- Tanti utenti (richiede un alto throughput).

Idee:

- Dividere i file in "chunks" (pezzetti da 64 MB) con etichetta univoca (da 64 bit).
- I "chunks" sono distribuiti su più "chunk servers".
- Esiste un *master node* che memorizza i metadati necessari a mappare l'etichetta dei chunk nei chunk server. Inoltre il *master node:*
  - Fa la *name resolution.*
  - Fa *replacing* (ogni chunk in tre diversi chunk server).
  - La richiesta viene erogata dai chunk server e non dal *master node*.
  - Se cade il *master node* esistono vari *shadow master* (master secondari).
  - Gestisce i lock.
  - I chunk server comunicano con il master tramite un *heart bit* (battito di cuore) per segnalare la loro presenza. In caso di rottura il master non sentendo più il "battito" reagisce al guasto.

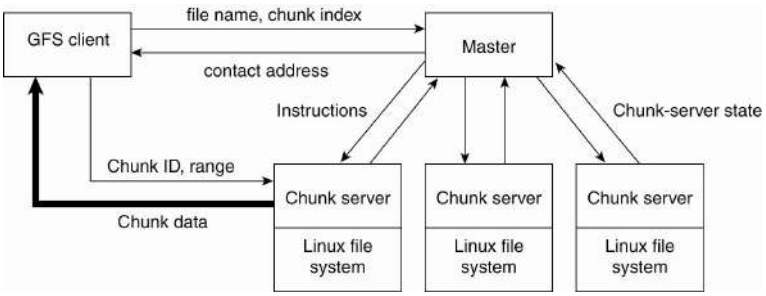


Figura 53: GFS.

Symmetric Architecture

- Non ha una componente centralizzata.
- C'è un identificatore globale. Tramite la DHT (Distributed Hash Table) trovo la risorsa.
- Due approcci principali:

– Costruire il file system sul livello di *storage* distribuito (IVY)
– Immagazzinare interi file sui nodi partecipanti (Kosha)

Parte X
Da CORBA ai Web Service

20 Message-Based

20.1 JMS - Java Messaging Service

- Le API per l'utilizzo di messaggi sono indipendenti dal venditore.
- Supporta diversi modelli: punto a punto, publish subscribe.
- Supporta la comunicazione asincrona.
- Supporta il meccanismo di ACK.

JMS for Enterprise

- Usa API uniformi per lo scambio di dati tra le applicazioni. Lo scambio di dati avviene tramite messaggi.
- Gli *adapter* convertono i dati in un formato comprensibile al componente a cui sono collegati. C'è un *adapter* per ogni componente del sistema.

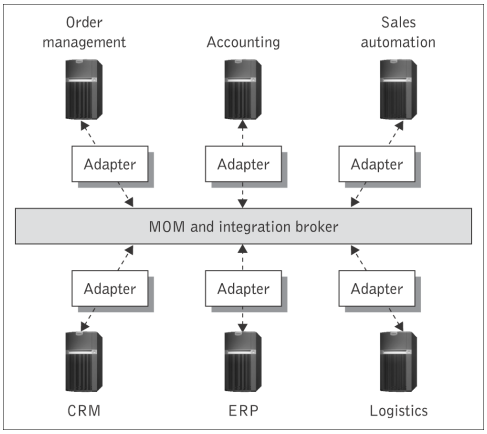


Figura 54: Un MOM e gli *adapter* che consentono le interazioni tra i diversi componenti del sistema.

21 Object-Based

21.1 CORBA - Common Object Request Broker Architecture

Non è un linguaggio di programmazione ma è una architettura che facilita la comunicazione tra processi sviluppati in linguaggi differenti e che risiedono in luoghi diefferenti.
Insieme di specifiche implementate in diversi prodotti, essi offrono diverse *facilities*. Permette di svincolarsi da un linguaggio specifico di programmazione senza impedire la comunicazione tra nodi che utilizzano linguaggi di programmazione differenti.

Caratteristiche:

- Trasparenza rispetto all'implementazione (eterogeneità dei linguaggi, il client non sa in che linguaggio è scritto il metodo che vuole invocare).
- Trasparenza rispetto alla locazione (al client non interessa in che luogo si trova il metodo, è presente un meccanismo di *naming* per identificare il determinato oggetto).
- Trasparenza sul meccanismo di comunicazione (non so come il MOM comunica con gli altri nodi del sistema).
- Trasparenza sullo stato in cui si trova il processo.

CORBA IDL

É un linguaggio di descrizione. Gli sviluppatori devono descrivere un interfaccia (che verrà utilizzata in remoto) in un linguaggio indipendente (CORBA IDL).
Bisogna creare compilatori specifici per ogni linguaggio che deve essere supportato.

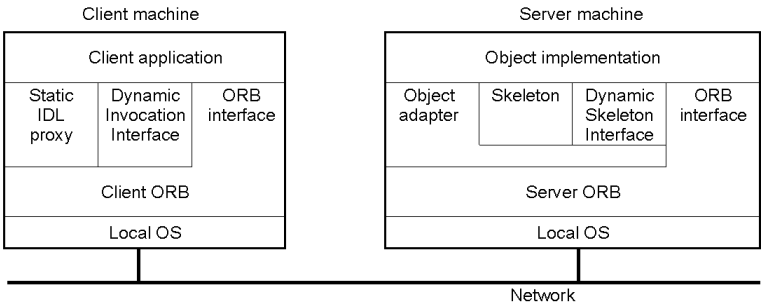


Figura 55: ORB - Object Request Broker

ORB: è un componente distribuito ogni nodo ha il suo ORB che comunicherà con gli altri ORB attraverso metodi specificati da CORBA. É suddiviso in due parti:

1. Client:
   (a) Proxy: si occupa del Marshaling dei parametri.
   (b) Dynamic Invocation Interface: si occupa di prelevare (anche a runtime) interfacce da repository in altri nodi del sistema.
   (c) ORB Interface: si occupa della *name resolution.*
2. Server:
   (a) Object Adapter: si occupa di gestire l'oggetto durante la sua esecuzione (puntatori, variabili ecc)
   (b) Skeleton: si occupa dell'Unmarshaling.
   (c) Dynamic Skeleton Interface: si occupa di prelevare (anche a runtime) interfacce da repository in altri nodi del sistema.
   (d) ORB Interface: si occupa della *name resolution.*

- Interface repository:
  - Gestisce tutte le definizioni di interfacce.
  - Usata dal client per le Dynamic Interface Invocation.
- Implementation repository
  - Contiene tutto quello che è necessario per eseguire l'oggetto
  - Usato da Object Adapter per localizzare e attivare l'oggetto

Corba Services

Service	Description
Collection	Facilities for grouping objects into lists, queue, sets, etc.
Query	Facilities for querying collections of objects in a declarative manner
Concurrency	Facilities to allow concurrent access to shared objects
Transaction	Flat and nested transactions on method calls over multiple objects
Event	Facilities for asynchronous communication through events
Notification	Advanced facilities for event-based asynchronous communication
Externalization	Facilities for marshaling and unmarshaling of objects
Life cycle	Facilities for creation, deletion, copying, and moving of objects
Licensing	Facilities for attaching a license to an object

Naming	Facilities for systemwide name of objects
Property	Facilities for associating (attribute, value) pairs with objects
Trading	Facilities to publish and find the services on object has to offer
Persistence	Facilities for persistently storing objects
Relationship	Facilities for expressing relationships between objects
Security	Mechanisms for secure channels, authorization, and auditing
Time	Provides the current time within specified error margins

Figura 56: Funzionalità offerte da CORBA ai programmatori.

Global Inter-ORB Protocol

Definisce le regole che devono seguire due ORB per parlarsi.
IIOP: Internet Inter-ORB Protocol è GIOP su Internet.

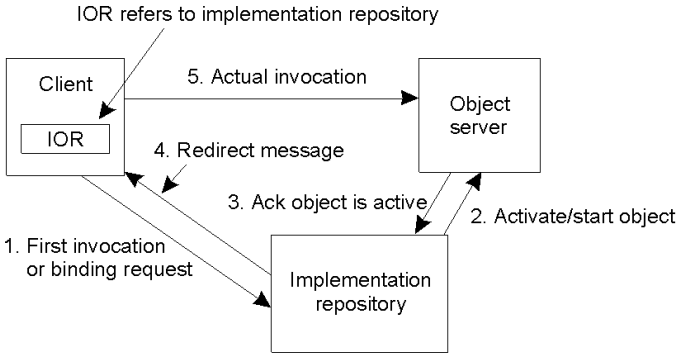


Figura 57: *Indirect Binding* - Implementation Repository (è il registry in CORBA) riceve la richiesta da parte del client, attiva l'oggetto sul server e invia al client la posizione dell'oggetto a cui vuole accedere. Il client invoca l'oggetto nel luogo indicato dalla Implementation Repository.

22 Web-Based

22.1 Web Services

- Il modello Web funziona bene e scala bene.
- CORBA è complicato e poco efficiente.
- Qualche problema di sicurezza con RMI e CORBA, con i firewall che bloccano le porte non standard.
- XML emerge come uno standard per la descrizione e lo scambio di dati.

Definizioni:

- Componenti per la programmazione web.
- Tutto è descritto nell'ambito del servizio stesso, sono modulari e possono essere pubblicati, individuati e utilizzati con protocolli web.

Proprietà:

- Possono essere usati internamente, in ambito intranet limitate ad un certo numero di nodi.
- Sono accessibili attraverso una interfaccia standard, serve WSDL (XML) per la definizione dell'interfaccia.
- Permettono a sistemi eterogenei di comunicare tra di loro.
- Usano un formato dei dati standard.

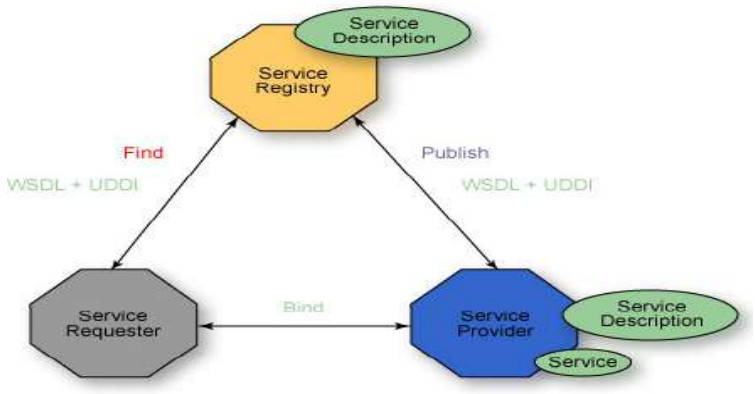


Figura 58: Service-Oriented Architecture.

1. Il *Service Provider* implementa un servizio e lo rende disponibile. La descrizione di questo servizio avviene tramite WSDL. WSDL descrive come un client può richiedere una funzione esposta in questo servizio (parametri, cosa restituisce e in che formato). Inoltre bisogna creare una *Service Description:* è una descrizione non tecnica di cosa fa il servizio.

2. Il *Service Registry* tiene un elenco di tutti i servizi che sono disponibili. I servizi non risiedono sul nodo registry, ma questo restituisce solo un link al server che mette a disposizione un determinato servizio. L'operazione di pubblicazione del servizio avviene mediante la *publish* del servizio sul *Service Registry* da parte del *Service Provider*. La *publish* utilizza il protocollo UDDI e permette di pubblicare la WSDL sul *Service Provider*.

3. Il *Service Requester* esegue la *find* ossia ricerca il servizio nel *Service Registry* tramite UDDI. Ottiene dal registry o un link al server su cui c'è il servizio (URL) oppure direttamente la WSDL (solo se il *Service Provider* ha deciso di pubblicarla). Se il service registry restituisce solo il link sarà il *Service Requester* a dover contattare il provider per farsi dare la WSDL altrimenti non può comunicare con il Web Service.

4. Il *Service Requester* esegue la *bind*: stabilisce il canale di comunicazione per permettere al *Service Requester* di comunicare direttamente con il *Service Provider*. La comunicazione avviene tramite il protocollo SOAP.

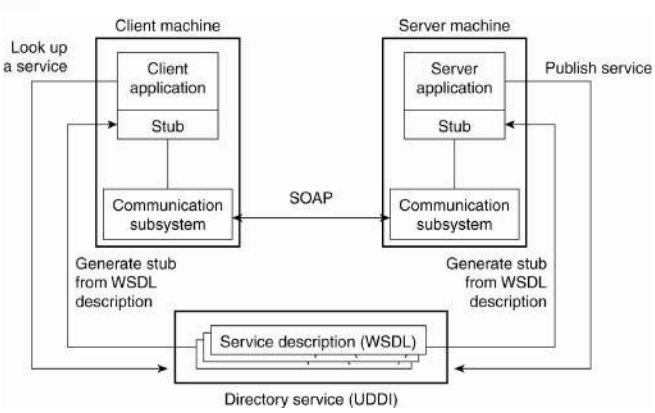


Figura 59: Architettura di un Web Service.

Figura 60: Comunicazione un Web Service.

Protocolli principali:

- WSDL - Utilizzato per descrivere come il client deve comunicare con il Web Service. Sintassi XML definita in modo particolare con un certo schema, il contenuto è la descrizione del servizio in un formato tecnico in modo che può essere possibile generare automaticamente la parte di comunicazione. La produzione di questi file può avvenire in modo automatico ma la qualità di questi file è pessima.
- UDDI (Universal Description Discovery and Integration) - Utilizzato per la comunicazione con il Service Registry. Contiene tutte le informazioni riguardanti un particolare servizio. Ha una sua sintassi in parte IDL e in parte informazioni in linguaggio naturale. Può anche contenere la WSDL del servizio. Contiene tutte le informazioni tecniche necessarie per contattare il Web Service che offre il determinato servizio.
- SOAP - Protocollo di comunicazione tra client e server. Descritto nella sezione 22.2.

Elementi dell'infrastruttura dei Web Service:

- Directories.
- Discovery.
- Description: contengono il WSDL.
- Wire Formats.

22.2 SOAP

Protocollo per scambiare informazione semi strutturata (anche tipizzata). É un formato di serializzazione che usa XML, la comunicazione avviene tramite il payload di HTTP. Il messaggio è spedito tramite una richiesta Post HTTP.

Due viste:

- Come una RPC, è come invocare una funzione sul Web Service.
- Come un protocollo di general-purpose "messaging".
  - La strutturazione viene decisa da chi costruisce il servizio.

Caratteristiche:

- É basato su XML, l'idea è quella di rendere il servizio auto descrittivo e auto contenuto.
- Envelope: che cosa contiene il messaggio e come lo si può interpretare in modo tale che chi lo riceve sa come trattarlo.
- Convenzione per rappresentare le RPC.
- É modulare ed estensibile (niente semantica di Applicazione e Trasporto). L'informazione deve essere la più semplice possibile. SOAP è definito a livello Applicazione e si appoggia (principalmente) su HTTP.

SOAP Message

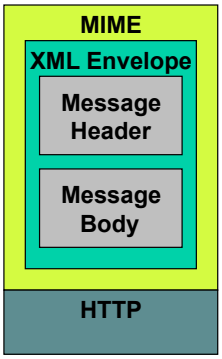


Figura 61: Composizione del messaggio SOAP.

É composto da un envelope che a sua volta è composto da un Header e da un Body:

- Header (opzionale): usato per sicurezza e per il coordinamento della combinazione di diversi Web Service
- Body contiene il nome dei metodi e i parametri della funzione che si vogliono eseguire e l'eventuale risposta.
  - Fault (extra) usato in caso di eccezioni.