Università degli Studi di Roma “Tor Vergata”
Facoltà di Ingegneria

Architetture dei Sistemi Distribuiti

Corso di Sistemi Distribuiti

Valeria Cardellini
Anno accademico 2008/09

Architettura sw di sistemi distribuito

- Definisce l’organizzazione e l’interazione dei vari componenti software che costituiscono il sistema
- Diverse scelte possibili nella realizzazione di un sistema distribuito

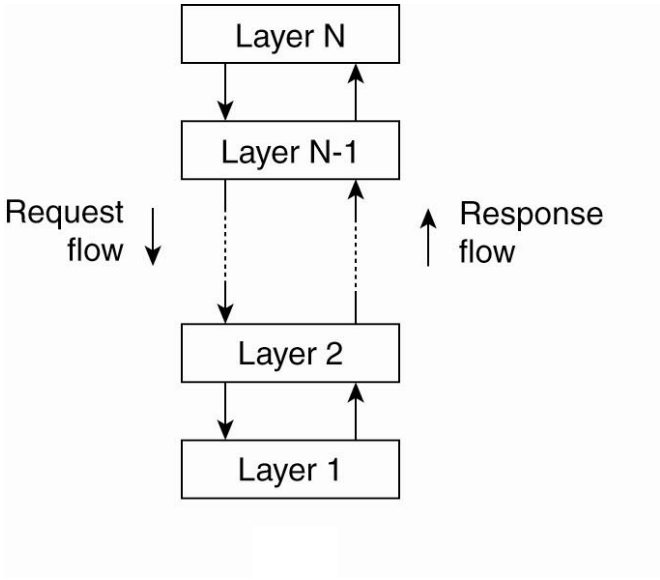
- Architettura di sistema: istanziazione finale di un’architettura software
  - I vari componenti del sistema distribuito sono posizionati sulle diverse macchine che lo costituiscono
- Tipologie di architetture di sistema
  - Architetture centralizzate
  - Architetture decentralizzate
  - Architetture ibride

Stili architetturali

- Espressi in termini di definizione e uso di componenti e connettori
- Componente
  - Unità modulare dotata di interfacce ben definite
  - Completamente sostituibile nel suo ambiente
- Connettore
  - Meccanismo che consente comunicazione, coordinamento o cooperazione tra componenti
- Stili architetturali prevalenti per SD
  - Architetture a livelli (layer)
  - Architetture basate su oggetti
  - Architetture basate su eventi
  - Architetture orientate ai dati

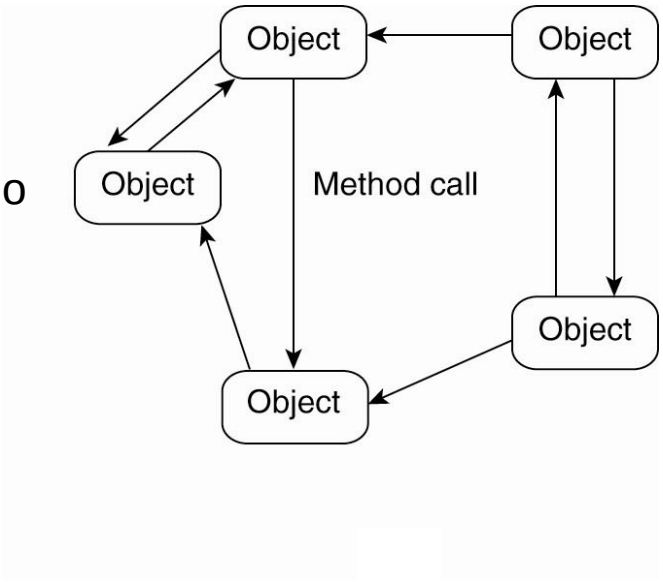
Architettura a livelli

- Componenti organizzati in livelli
- Un componente a livello *i* può invocare un componente del livello sottostante *i-1*
- Le richieste scendono lungo la gerarchia, mentre le risposte risalgono
- Largamente adottato dalla comunità della rete



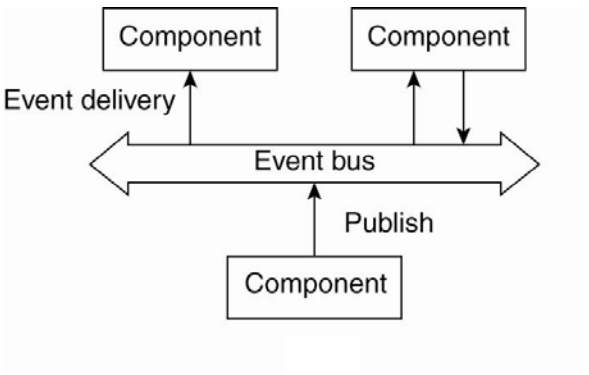
Architettura basata su oggetti

- Ogni componente è un oggetto
- Oggetti connessi attraverso un meccanismo di chiamata a procedura remota (o invocazione di metodo remota)



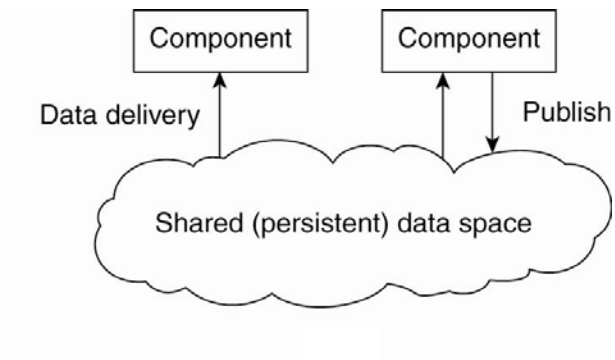
Evoluzione degli stili architetturali

- Il disaccoppiamento dei componenti nello spazio ("anonimi") ed anche nel tempo ("asincroni") ha determinato stili architetturali alternativi



Architettura basata su eventi

I componenti comunicano tramite la propagazione di eventi



Architettura orientata ai dati

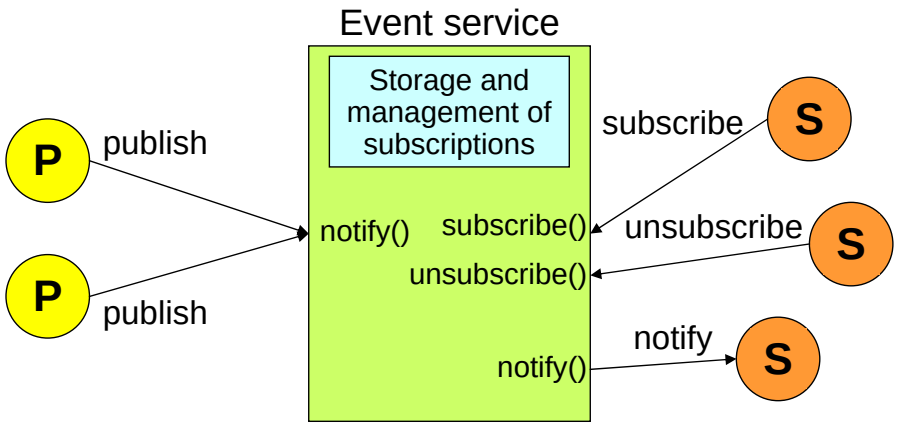
I componenti comunicano tramite un repository comune (passivo o attivo), ad es. un file system distribuito

Disaccoppiamento

- Porre vincoli sui componenti che possono interagire può introdurre necessità di conoscenze che a volte non sono necessarie
- Il disaccoppiamento diventa un fattore per l'abilitazione di maggiore flessibilità e per arrivare a definire stili architetturali che possono sfruttare al meglio la potenziale distribuzione e scalabilità
- Proprietà di disaccoppiamento
  - Spaziale: i componenti non devono conoscersi (reciprocamente o meno)
  - Temporale: i componenti interagenti non devono essere compresenti (presenti insieme nello stesso instante) quando la comunicazione ha luogo
  - Di sincronia: i componenti interagenti non devono aspettarsi a vicenda e non sono soggetti a blocchi reciproci

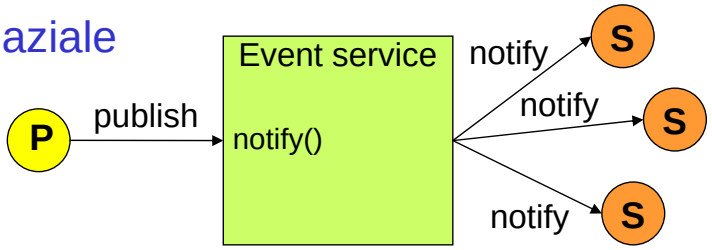
Publish/subscribe

- I produttori generano eventi (publish) e si disinteressano della loro consegna
- I consumatori si sono registrati come interessati ad un evento (subscribe) e sono avvisati (notify) della sua occorrenza
- Disaccoppiamento tra entità interagenti

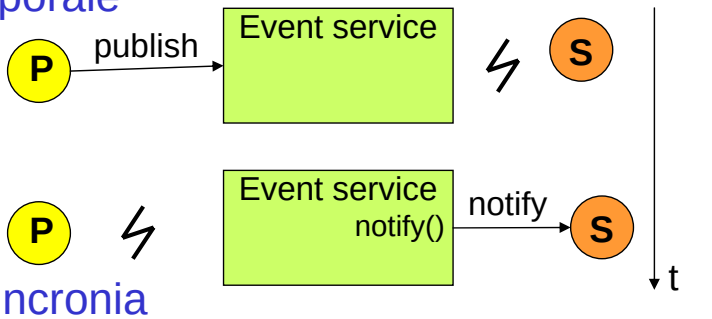


Publish/subscribe e disaccoppiamento

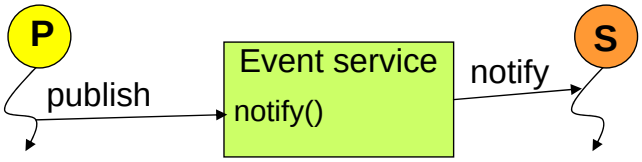
- Disaccoppiamento spaziale



- Disaccoppiamento temporale



- Disaccoppiamento di sincronia



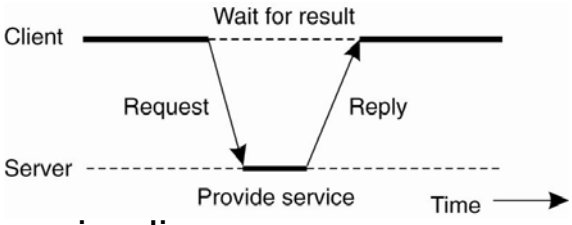
Riferimento: P.T. Eugster *et al*, "The many faces of publish/subscribe" *ACM Comput. Surv.* 35(2):114-131, June 2003.

Architetture di sistema

- Quali componenti software usare?
- Come interagiscono i componenti?
- Dove sono posizionati i componenti?

- Tipologie di architetture di sistema
  - Architetture centralizzate
  - Architetture decentralizzate
  - Architetture ibride

Architetture centralizzate



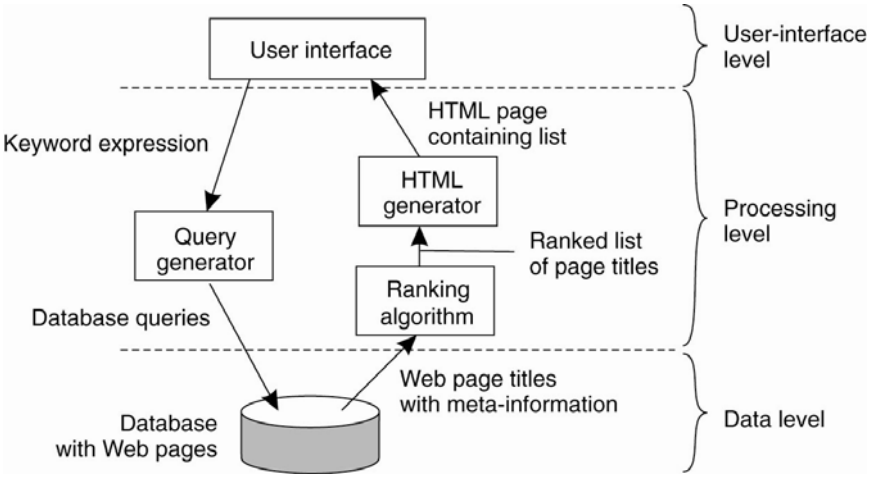
- E' il modello client/server
  - Già analizzato
  - Intrinsecamente distribuito
- L'interazione tra client e server implica un comportamento "request-reply"
  - Una sorgente del problema prestazionale nel Web
  - Alcune richieste (ma non tutte) sono idempotenti
    - Possono essere ripetute più volte senza causare danni o problemi
    - Proprietà importante in caso di comunicazioni inaffidabili
    - Rendere logicamente affidabile una interconnessione fisicamente inaffidabile ha un costo molto elevato
- Asimmetria con il client che conosce il server e interagisce in modo sincrono e bloccante (di default)
- Forte accoppiamento
  - Compresenza di chi interagisce

Stratificazione

- Il tradizionale stile architetturale a livelli (layer) applicato alle architetture centralizzate
  - Livello dell'interfaccia utente
    - Spesso suddiviso in due livelli (livello client e livello di presentazione lato server)
  - Livello applicativo
  - Livello dei dati
- Stratificazione presente in molti sistemi informativi distribuiti, che implementano il livello dei dati mediante basi di dati relazionali, ad oggetti o object-relational

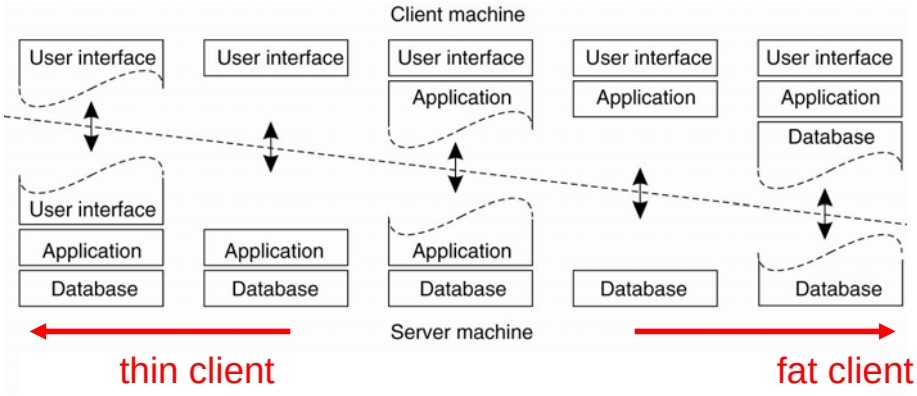
Stratificazione delle applicazioni (2)

- Esempio: motore di ricerca Web



Architetture multilivello

- Mapping tra livelli logici (layer) e livelli fisici (tier)
- Architettura ad un livello (single-tier): configurazione monolitica mainframe e terminale "stupido" (non è C/S)
- Architettura a due livelli (two-tier): due livelli fisici (macchina client/singolo server)
- Architettura a tre livelli (three-tier): ciascun livello su una macchina separata
- Diverse configurazioni two-tier



Architetture multilivello (2)

- Da un livello ad N livelli
- Con l'introduzione di ciascun livello
  - L'architettura guadagna in flessibilità, funzionalità e possibilità di distribuzione
- Ma l'architettura multilivello potrebbe introdurre un problema di prestazioni
  - Aumenta il costo della comunicazione
  - Viene introdotta più complessità, in termini di gestione ed ottimizzazione

Architetture multilivello (3)

- Varie architetture client-server sono possibili
  - In relazione all'organizzazione del servizio e dei suoi dati
- Distribuzione verticale
  - Componenti logicamente diversi dello stesso servizio possono essere assegnati a macchine distinte
    - Sia sul lato server che sul lato client (delega parziale)
  - Servizio reso tramite cooperazione di componenti distribuiti
    - Ripartizione gerarchica (anche di autorità)
- Distribuzione orizzontale
  - Server e client possono essere partizionati ma ogni loro componente può operare da solo
  - Condivisione del carico (con ripartizione del lavoro gestita da un dispatcher)
  - Ogni componente sa fornire "il" servizio richiesto
  - Esempio già esaminato: Web cluster

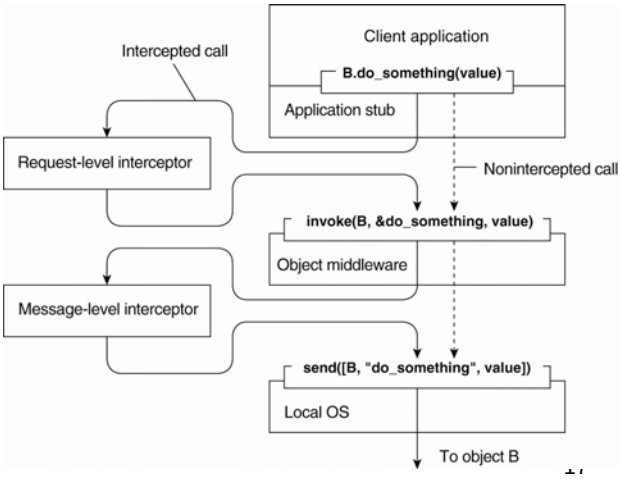
Architetture decentralizzate e ibride

- Abbiamo già esaminato esempi di architetture decentralizzate
  - Sistemi P2P
- …e di architetture ibride
  - In cui la classica soluzione client-server è combinata con un'architettura decentralizzata, ad esempio:
  - Sistemi edge server nelle Content Delivery Network
  - Sistemi distribuiti collaborativi, ad esempio basati sul protocollo Bit Torrent

Architetture e middleware

- In molti casi i sistemi middleware seguono uno specifico stile architetturale
  - Ad es. Message Oriented Middleware (MOM), Distributed Object Computing (DOC) Middleware
- Un approccio architetturale al middleware offre
  - Semplicità progettuale
  - Scarsa adattabilità e flessibilità
- E' preferibile un middleware che si possa adattare all'applicazione specifica ed all'ambiente anche in modo dinamico, reattivo e radicale

- Interceptor: costrutto software che interrompe il normale flusso di esecuzione e consente ad altro codice di essere eseguito
- Modifica la richiesta prima che sia inviata al componente
- E' uno strumento per adattare il middleware



Middleware adattivo

- Tre tecniche di base per ottenere un software adattivo
  - Separazione degli ambiti
  - Riflessione computazionale
  - Progettazione per componenti
- Separazione degli ambiti
  - Cercare di separare le funzionalità dalle extra funzionalità (affidabilità, prestazioni, sicurezza, …) e successivamente integrarle in una singola implementazione
  - Anche aspect-oriented software
  - Fino ad oggi solo esempi “giocattolo”
- Riflessione computazionale
  - Capacità di un programma di controllare se stesso e, se necessario, di adattare il proprio comportamento durante l’esecuzione
    - Relativamente al middleware: le politiche di azione sono espresse e visibili nel middleware stesso e si possono cambiare come componenti del sistema
  - Integrato in alcuni linguaggio di programmazione (ad es. Java), non in sistemi distribuiti a larga scala

Middleware adattivo (2)

- Progettazione per componenti
  - Organizzare un’applicazione distribuita in componenti che possono essere cambiati dinamicamente quando occorre
  - Supporto per il binding tardivo (o late composition): il sistema può essere configurato a tempo di esecuzione
    - Non solo staticamente in fase di progettazione
  - Complessa, anche per le interdipendenze tra componenti
- Occorre un software adattivo o un sistema adattivo?

Riferimento: P.K. McKinley et al., “Composing adaptive software”, *IEEE Computer*, 37(7):56-64, July 2004.

Sistemi distribuiti autonomici

- *Autonomic Computing*: introdotto come paradigma architetturale e tecnologico in grado di rispondere all'esigenza di gestire la crescente complessità ed eterogeneità dei sistemi IT mediante adattamenti automatici
- Metafora del sistema nervoso centrale, in grado di controllare alcune funzioni vitali (battito cardiaco, temperatura, …) mascherando la complessità all'uomo
- Un sistema autonomico dovrebbe essere in grado di gestire le proprie funzionalità autonomamente (ovvero senza o con limitato intervento umano), esprimendo capacità di adattamento (comportamenti *reattivi*) a dinamiche interne ed esterne, più in generale ai cambiamenti dell'ambiente

Sistemi distribuiti autonomici (2)

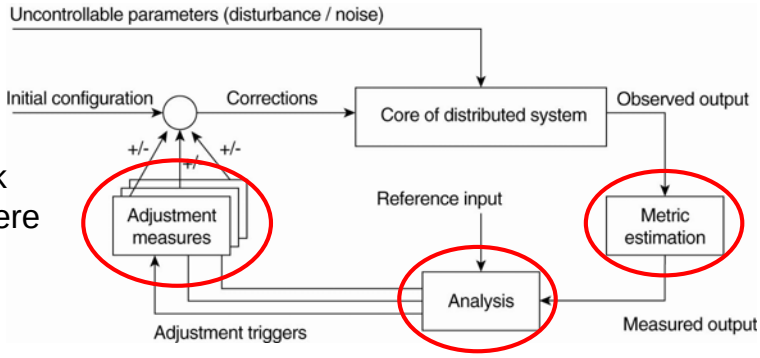
- Un sistema autonomico, a fronte di determinate "policy" impostate dall'amministratore, dovrebbe possedere le seguenti caratteristiche:
- Autogestione
- Auto-configurazione
- Auto-guarizione
  - Garantire sopravvivenza ai guasti (ad es. malfunzionamenti hardware)
- Auto-ottimizzazione
  - Ottimizzare le proprie prestazioni
- Auto-protezione
  - Proteggersi da attacchi esterni
- Complessivamente: essere un sistema auto-* (o self-*)

Riferimento: J.O. Kephart, D.M. Chess, "The vision of Autonomic Computing", *IEEE Computer*, Vol. 36, No. 1, Jan. 2003.

Sistemi a controllo dei feedback

- In molti casi, i sistemi auto-* sono organizzati come sistemi a controllo dei feedback
  - Componente per la stima della metrica
    - Misura il comportamento del sistema
  - Componente di analisi dei feedback
    - Analizzare le misure e le confronta con dei valori di riferimento
    - E' il cuore del ciclo di controllo, poiché contiene gli algoritmi che decidono gli eventuali adattamenti
  - Meccanismi per influenzare il comportamento del sistema

Organizzazione *logica* di un sistema a controllo dei feedback (l'organizzazione *fisica* può essere molto diversa: i vari componenti possono essere distribuiti)



Esempio: Globule

- Globule: CDN collaborativa che analizza tracce per decidere su quali edge server collocare repliche del contenuto Web
- Decisioni guidate da un modello di costo generale:

  $cost = (w_1 \times m_1) + (w_2 \times m_2) + \cdots + (w_n \times m_n)$
- L'origin server raccoglie tracce ed effettua un'analisi di tipo *what-if*
  - Controllando cosa sarebbe successo se la pagina *P* fosse stata replicata nell'edge server *S*
  - Valutate diverse strategie di replica e scelta la migliore