



# Le architetture software

---

D. Garlan, M. Shaw (1994)

1



## Introduzione

---

- Con la crescita delle dimensioni e della complessità di un sistema software, il design non può limitarsi ad algoritmi e strutture dati:
  - Organizzazione di massima del sistema;
  - Protocolli di comunicazione;
  - Sincronizzazione;
  - Distribuzione delle funzionalità;
  - Distribuzione fisica dei componenti e loro composizione;
  - Scalabilità e performances;
- Tutto questo costituisce l'architettura

2



## Cosa c'e' a disposizione?

---

- Linguaggi di interconnessione tra i moduli;
- Template e frameworks domain-oriented;
- Modelli formali;
- Principi per caratterizzare le strutture architettonali;
- Patterns.

3



## Importanza della scelta

---

- Scegliere l'architettura più adeguata è importante per la buona riuscita di un progetto;
- Ciò consente di indirizzare al meglio le decisioni progettuali;
- Inoltre, rende possibile il riuso
  - Design patterns

4



## Dal linguaggio macchina ai linguaggi di alto livello

---

- Inizialmente il software era scritto in linguaggio macchina (anni 50);
- Successivamente, riferimenti simbolici alla memoria e agli operatori (assembly);
- Macro processors;
- Identificazione di pattern ricorrenti (es. stampa una stringa, funzioni matematiche) e nascita dei primi linguaggi di alto livello (es. Fortran)

5



## Dai linguaggi di alto livello alle architetture software

---

- Identificati pattern più avanzati (strutture dati), -> Algol
- Tipi di dati astratti (fine anni '60)
  - Struttura software (rappresentazione e operatori);
  - Specifiche;
  - Issues del linguaggio;
  - Integrità dei risultati;
  - Regole per combinare tipi;
  - Information hiding.

6



## Le architetture software

---

- Così come negli anni '60 sono state identificate le strutture dati ricorrenti, è possibile identificare organizzazioni ricorrenti nei sistemi software.
- Esempi di descrizione di architetture:
  - Modello client-server;
  - Layering;
  - Sistemi object-oriented;
  - Pipelines.

7



## Architetture Software

---

- Pipeline e filtri
- Organizzazione object-oriented
- Invocazione implicita event-based
- Layering
- Repository
- Interpreti
- Sistemi distribuiti e client/server
- Altre architetture
- Architetture eterogenee

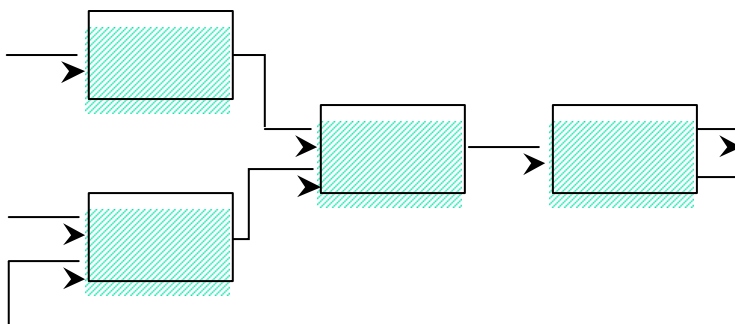
8

## Pipeline e filtri

- Ciascun componente ha un insieme di input e un insieme di output;
- Il componente legge stream di dati dai suoi input, li trasforma e produce stream di output;
- I componenti (*filtri*) possono essere collegati tra loro mediante connettori (*pipe*)
- E' importante che i componenti siano entità indipendenti
  - Specializzazioni: Pipelines, bounded-pipes, typed-pipes

9

## Schema



10



## Esempi

---

- L'esempio classico è l'utilizzo in pipe dei comandi UNIX

```
cat miofile.h | grep "#if" | sort >output
```
- Altro esempio, un compilatore (sia pur rudimentale):
  - Lexer | parser | analisi semantica | generazione di codice
- Le pipeline possono degenerare in sistemi batch, se i filtri processano i dati come entità atomiche

11



## Pros and cons

---

- I/O di ogni componente ben identificati;
- Identificazione dei filtri
- Riutilizzo facilitato
- Manutenzione semplice
- Throughput e deadlock analysis
- Esecuzione concorrente
- Organizzazione batch
- Inadatti ad applicazioni interattive
- Lavoro aggiuntivo per il parsing dello stream

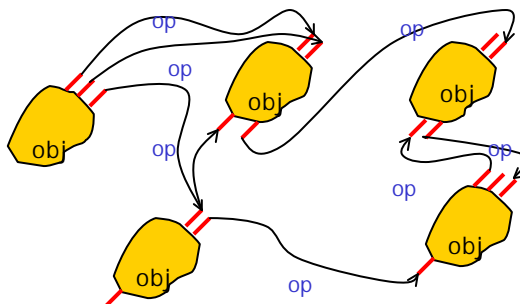
12

# Astrazione dati e organizzazione object-oriented

- Le rappresentazioni dei dati e le operazioni associate sono incapsulate in oggetti o tipi di dati astratti;
- I componenti di questo stile sono oggetti, o istanze di tipi di dati astratti;
- Interazione mediante invocazione di metodi
- Aspetti importanti:
  - Ogni oggetto deve preservare l'integrità della propria rappresentazione;
  - La rappresentazione deve essere nascosta.

13

## Esempio



14



## Pros and cons

---

- Proprietà ben note (vedere UML)
- Possibile modificare l'implementazione senza impattare sui client di un oggetto
- Decomposizione del problema in collezione di agenti interagenti
- Ogni oggetto deve conoscere gli oggetti con i quali interagisce

15



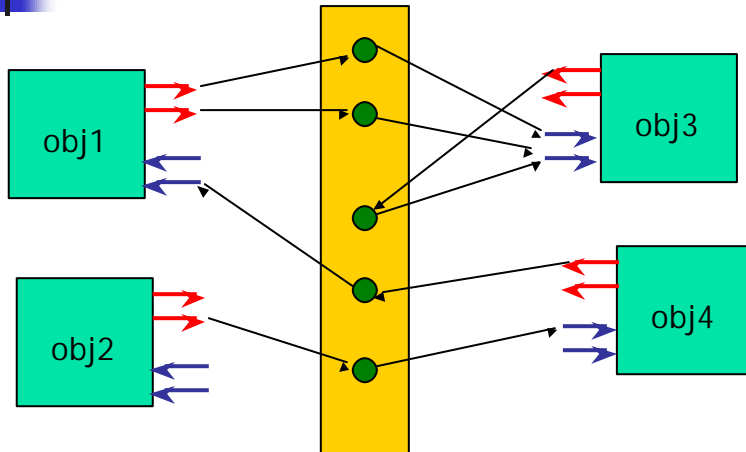
## Invocazione implicita event-based

---

- Tradizionalmente i componenti interagiscono mediante invocazione di routines
- Alternativa:
  - Invece di invocare una procedura, un componente annuncia (broadcast) uno o più eventi
  - Altri componenti possono registrare un evento, associando ad esso una procedura
  - In definitiva, il broadcast di un evento causa l'invocazione (implicita) delle procedure registrate per tale evento

16

## Schema



17

## Esempio

- Consideriamo un frontend per un debugger simbolico (es. *xxgdb*)
  - Quando il debugger interrompe l'esecuzione ad un breakpoint, invia un evento all'interfaccia utente, sulla quale la linea di codice viene evidenziata
  - Il debugger annuncia l'evento, non sa che qualcuno lo intercetta
- Event listener su GUI widgets (es. Java)
- Trigger sui database
- Editors con syntax checking/coloring

18



## Pros and cons

---

- Supporto al riuso: componenti possono essere aggiunti registrandoli per gli eventi del sistema
- Supporto all'evoluzione: i componenti possono essere rimpiazzati senza influenzare l'interfaccia di altri
- Quando un componente annuncia un evento, non sa se e chi risponderà
- Scambio dati: con l'evento, area shared

19



## Sistemi layered

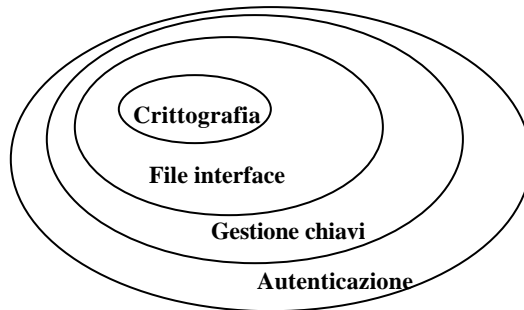
---

- Un sistema a layer è organizzato in maniera gerarchica
- Ogni layer mette a disposizione servizi a quello superiore, utilizzando quelli del layer sottostante
- Un layer può fungere da "macchina virtuale" per il livello superiore
- Possibilità di mettere a disposizione servizi a layer oltre il livello direttamente superiore (struttura parzialmente opaca)
- I connettori sono definiti dai protocolli di interazione tra layer

20

## Schema

**Esempio:  
Sistema per  
la sicurezza  
dei files**



21

## Esempi

- Protocolli di comunicazione (pila OSI, TCP/IP)
- Protezione files
- Sistemi operativi
- DBMS

22



## Pros and cons

---

- Design basato su livelli di astrazione crescenti
- Possibilità di modificare l'implementazione di un layer senza modificarne le interfacce
- Supporto al riuso (es. X-Window)
- Anche se un sistema è logicamente layered, potrebbe non esserlo fisicamente
- Cosa fa ciascun layer? (ISO/OSI)
- Problemi di prestazioni

23



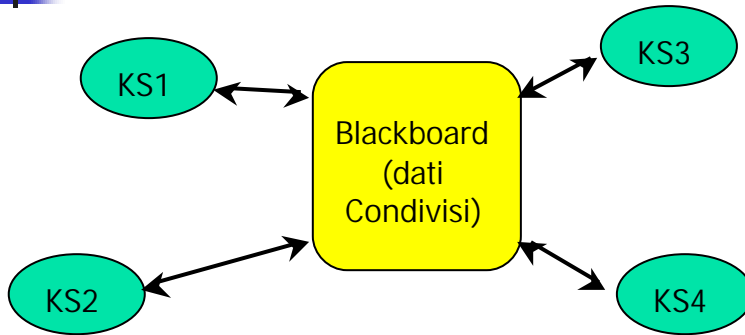
## Repository

---

- Il sistema è composto da un data store centrale, e da un insieme di componenti che vi interagiscono
- Se lo stato del data store attiva i componenti, il data store andrebbe visto come una blackboard
- Modello a blackboard:
  - Sorgenti di conoscenza, che interagiscono tramite la blackboard (KS)
  - Struttura dati della blackboard (stato dell'applicazione)
  - Controllo: guidato dallo stato della blackboard

24

## Schema



25

## Esempi

- Signal e speech recognition;
- Programming environments;
- CASE, CAD/CAM;
- Sistemi batch con database centralizzati;
- Architettura di un moderno compilatore (dati condivisi: AST, tabella dei simboli)

26



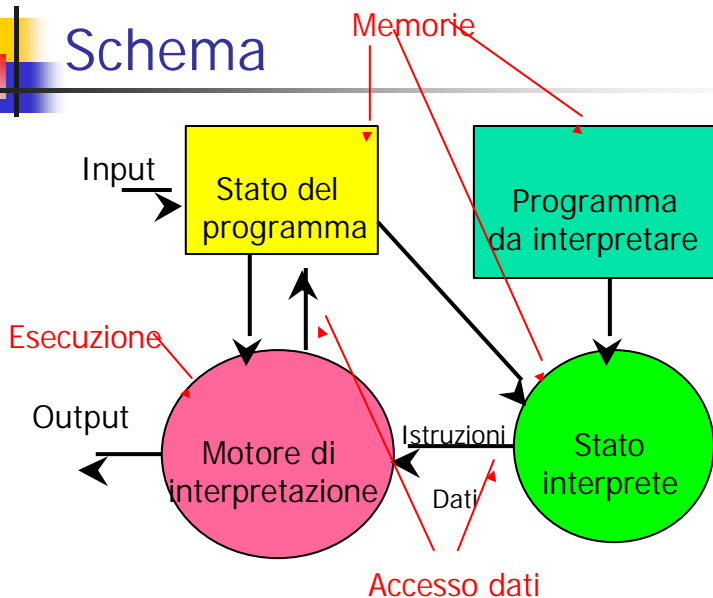
## Interpreti table-driven

- Un interprete è costituito da uno pseudo-programma da interpretare e dal motore dell'interprete stesso
- Lo pseudo-programma comprende il codice e lo stato del programma (record di attivazione)
- Il motore comprende le regole di interpretazione e lo stato dell'interprete
- Utilizzati per costruire macchine virtuali

27



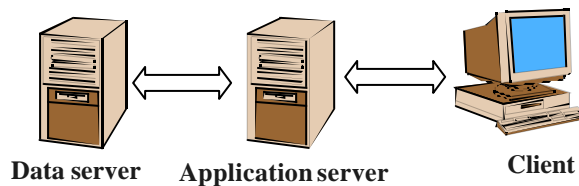
## Schema



28

## Processi distribuiti

- Topologie ad anello o stella
- Protocolli di comunicazione (CORBA, IPC)
- Sistemi client/server: il server non conosce l'identità e il numero dei client, che viceversa conosce il server e vi accede mediante chiamata di procedura remota
- Sistemi multi-tier



29

## Altri stili architetturali (I)

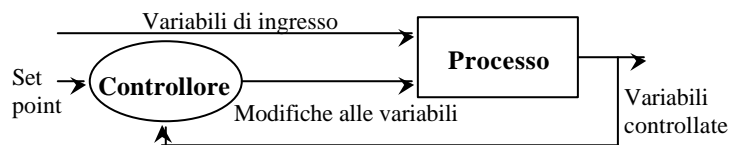
- Main program-subroutine: in linguaggi che non supportano la modularizzazione, è l'organizzazione più intuitiva
- Architetture domain-oriented
  - Es. avionica, veicoli
  - Maggiore potenza espressiva
  - Generazione automatica del codice

30



## Altri stili architetturali (II)

- Macchine a stati
- Sistemi di controllo dei processi
  - Elementi computazionali;
  - Dati e sensori dai quali acquisire i dati;
  - Schema di controllo: open-loop, closed loop, feedback o feedforward.



31



## Architetture eterogenee

- Spesso non esiste una silver bullet, ma è necessario combinare tra loro più stili diversi
  - Organizzazione gerarchica: es. ogni filtro UNIX al suo interno può avere la propria organizzazione;
  - Connettori misti: un componente comunica con un repository condiviso, ma anche con altri componenti per mezzo di pipe o messaggi;
  - Active database: in esso i componenti si registrano, e sono poi attivati al verificarsi di eventi (ibrido blackboard-even driven)

32



## Esercizio

---

- *KWIC (Key Word in Context)* [Parnas, 1972] accetta in input un insieme ordinato di linee;
- Ciascuna linea è costituita da un insieme ordinato di parole, ogni parola da un insieme ordinato di caratteri;
- Ogni linea può essere sottoposta a "shift circolare" rimuovendo la prima parola e "appendendola" in coda
- KWIC produce in output la lista di tutti i possibili shift circolari delle linee in input, in ordine alfabetico.

33



## To Do

---

- Ottimizzare l'architettura per prevedere:
  - Modifiche nell'algoritmo (shifting delle linee appena lette da console, o su tutte le linee memorizzate)
  - Modifiche nella rappresentazione dei dati;
  - Non effettuare shift che iniziano con stop words ("a", "an", "and", "ecc.")
  - Ottimizzare prestazioni e risorse
  - Riutilizzo dei componenti

34



## To Do (2)

---

- Progettare il sistema utilizzando un'architettura main-subroutine-shared data
- Scegliere alcuni stili architetture e proporre soluzioni alternative