Copyright © 2004 Moreno Marzolla

This work is licensed under the Creative Commons Attribution-Noncommercial-Share Alike 2.5 Italy License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/2.5/it/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

Progettazione Architetturale

L'Architettura del Software

- La definizione dell'architettura del software non deve creare software operativo ma piuttosto:
  - Analizzare l'efficacia del progetto nel soddisfare i requisiti
  - Consentire la valutazione di tutte le possibili alternative architetturali del sistema in una fase in cui le modifiche sono relativamente poco costose
  - Ridurre i rischi tecnici nella successiva fase di implementazione del codice
- L'architettura NON E' la stesura del codice ma…
  - …una buona architettura rende (molto) più semplice la stesura del codice
- Architettura del software = Progetto dei Dati + Progetto Architetturale (o dei Componenti)

Perché l'architettura del software è importante?

- La rappresentazione dell'architettura è il mezzo tramite cui le parti interessate allo sviluppo del sistema comunicano
- L'architettura mette in evidenza le decisioni progettuali preliminari che avranno impatto sul lavoro di sviluppo successivo
- L'architettura costituisce un modello relativamente compatto e afferrabile del modo in cui il sistema è strutturato e di come i suoi componenti collaborano fra di loro

Progetto dei dati

- *"La progettazione dei dati a livello dei componenti si concentra sulla rappresentazione delle strutture dati il cui accesso avviene direttamente da parte di uno o più componenti software"*
- Sono stati individuati una serie di principi per la progettazione dei dati

Principi di progettazione dei dati / 1

- I principi di analisi sistematica che si applicano alla funzionalità e al comportamento devono essere applicati anche ai dati
- Si devono individuare tutte le strutture di dati e le operazioni da svolgersi su ciascuna
- Un dizionario dei dati deve essere compilato e utilizzato per definire il progetto dei dati e del programma
- Le decisioni di basso livello sul progetto dei dati devono essere rimandate alle ultime fasi della progettazione

Principi di progettazione dei dati / 2

- La rappresentazione delle strutture dati deve essere nota solo ai moduli che la utilizzano direttamente
  - Concetto di *information hiding*
- E' utile sviluppare una libreria di strutture dati utili e di operazioni
  - Progettare, ove possibile, le strutture dati e le operazioni in modo tali da essere riutilizzate
- La specifica e realizzazione dei tipi di dati astratti deve poggiare su un progetto del software e su un linguaggio di programmazione
  - Implementare una struttura dati sofisticata può essere difficile se non si dispone di supporto adeguato da parte del linguaggio di programmazione utilizzato

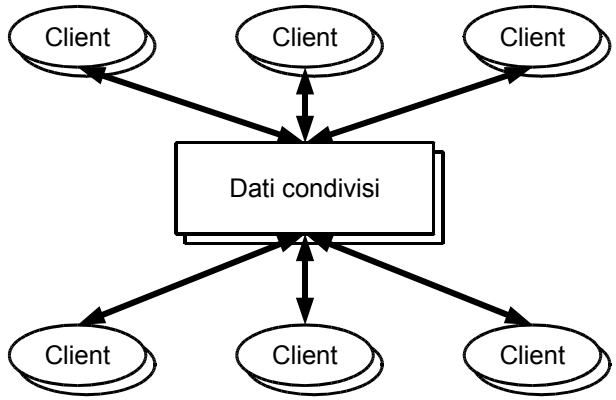
Stili di architettura

- Uno stile architetturale è composto da:
  - Un insieme di componenti che implementano le funzionalità richieste dal sistema
  - Un insieme di connettori che consentono comunicazione e cooperazione tra i componenti
  - I vincoli che definiscono i modi in cui i componenti possono essere integrati per costituire il sistema
  - Modelli semantici che consentono al progettista di comprendere il funzionamento globale del sistema in funzione delle sue componenti

Architettura basata sui dati
(Architettura a *Repository*)

- Il sistema è centrato su un archivio di dati
- Le altre componenti accedono a questo archivio di dati operando in modo indipendente
- L'archivio può essere *passivo* (tutte le operazioni sono in mano ai client) o *attivo* (l'archivio notifica ai client le variazioni nei dati)
- Il vantaggio sta nella sostanziale indipendenza tra i moduli: ogni modulo può essere aggiornato senza influenzare gli altri
- L'accoppiamento tra le componenti può avvenire attraverso un meccanismo a blackboard

Architettura a repository



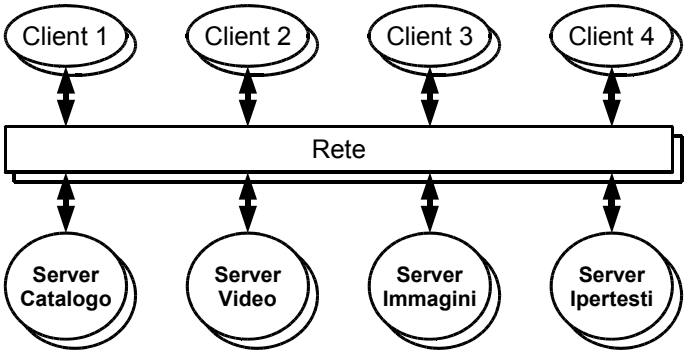
Caratteristiche dell'architettura a repository

- Vantaggi
  - E' un modo efficiente per condividere grandi quantità di dati
  - I sottosistemi (client) non necessitano di sapere in che modo i dati sono condivisi. La gestione dei dati è centralizzata (backup, sicurezza...)
  - Lo schema dei dati può venire pubblicato nel repository condiviso
- Svantaggi
  - I sottosistemi (client) devono basarsi sulla stessa struttura dei dati condivisi. Ossia, tutti i client devono usare la stessa rappresentazione dei dati; inevitabilmente, questo è un compromesso
  - Modificare la struttura (schema) dei dati è difficile e costoso
  - Difficile impostare delle politiche diverse di accesso ai dati
  - Bassa scalabilità: il repository centrale è un buon candidato ad essere un collo di bottiglia

Architettura client-server

- E' un modello distribuito, in cui viene evidenziato come i dati sono elaborati e distribuiti tra una famiglia di componenti
- Si compone di:
  - Un insieme di server autonomi che forniscono determinati servizi (stampa, gestione dei dati, ecc.)
  - Un insieme di client che utilizzano questi servizi
  - Una rete di comunicazione che permette a client e server di interagire

Esempio client-server: libreria digitale



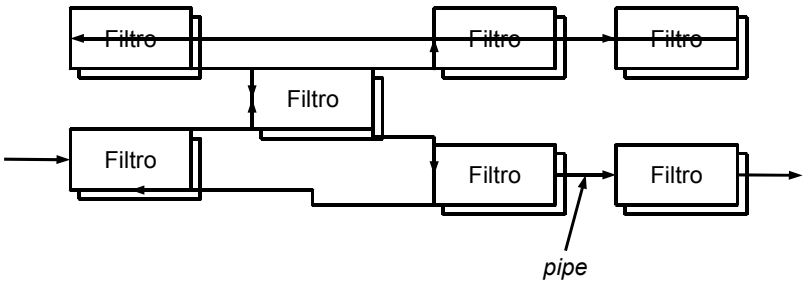
Caratteristiche dell'architettura client-server

- Vantaggi
  - La distribuzione dei dati è molto semplice
  - L'uso della rete permette di collegare molti nodi con bassa potenza di calcolo per effettuare computazioni complesse
  - Risulta facile aggiungere nuovi server o aggiornare server esistenti
- Svantaggi
  - Non esiste un unico modello di dati condivisi; ogni sottosistema fa uso di un proprio modello dei dati. Di conseguenza lo scambio di informazioni può essere inefficiente
  - Alcune attività di gestione dei dati (ad esempio backup) devono essere replicate in ciascun server
  - In generale, non c'è alcun registro centrale di nomi e servizi; può risultare difficile individuare quali server e servizi sono disponibili

Architettura a flusso di dati
(Architettura *pipe-and-filter*)

- Il sistema è modellato sul flusso che porta i dati dalla fase di input a quella di output
- I moduli si comportano da filtri connessi da *pipe* di dati
- Ogni filtro si attende solo dati in input con un certo formato e produce dati in output di formato fisso
- Ogni filtro lavora senza occuparsi dei filtri che lo precedono e lo seguono
- Se il flusso dei dati degenera in un'unica catena di filtri allora l'architettura si dice "a filtro batch sequenziale"

Esempio di architettura a flusso di dati



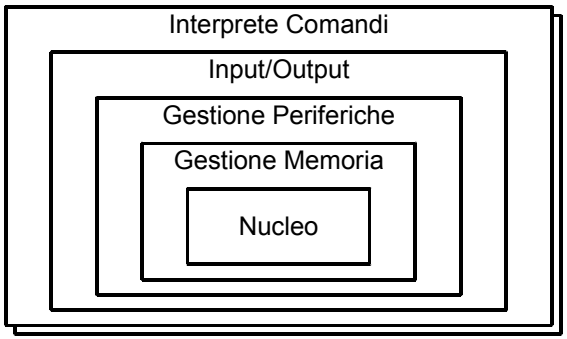
Caratteristiche dell'architettura a flusso di dati

- Vantaggi
  - E' facile costruire computazioni complesse tramite composizione di filtri semplici
  - Ogni filtro può essere considerato come una “scatola nera”, e riutilizzato per applicazioni differenti
  - Se ben programmati, i filtri non condividono lo stato (basso accoppiamento)
- Svantaggi
  - I formati dei dati di input e output di filtri collegati devono essere compatibili
  - Se la struttura di controllo non può essere “linearizzata”, questo modello non è adeguato

Architettura a macchina astratta
(Architettura a livelli)

- Viene utilizzata per modellare l'interfacciamento di sottosistemi
- Il sistema viene organizzato in un insieme di strati (o *macchine astratte*) ciascuno dei quali fornisce dei servizi
- Ciascun livello comunica solo con quelli adiacenti

Esempio di architettura a livelli



Caratteristiche dell'architettura a livelli

- Vantaggi
  - Supporta lo sviluppo del sistema in modo incrementale, livello dopo livello.
  - Se si cambia l'interfaccia di un livello, solo quello adiacente ne risente
- Svantaggi
  - Spesso può essere complicato strutturare il sistema in questo modo.
  - In particolare, è molto restrittivo supporre che ciascun livello possa interagire **solo** con quelli adiacenti

Modelli di controllo

- Descrivono il modo con cui fluisce il controllo tra i sottosistemi
- Controllo centralizzato
  - Un sottosistema ha la responsabilità ddel controllo globale, e avvia e disattiva i sottosistemi
- Controllo basato a eventi
  - Ciascun sottosistema può rispondere ad eventi generati da altri sottosistemi, o dall'ambiente esterno

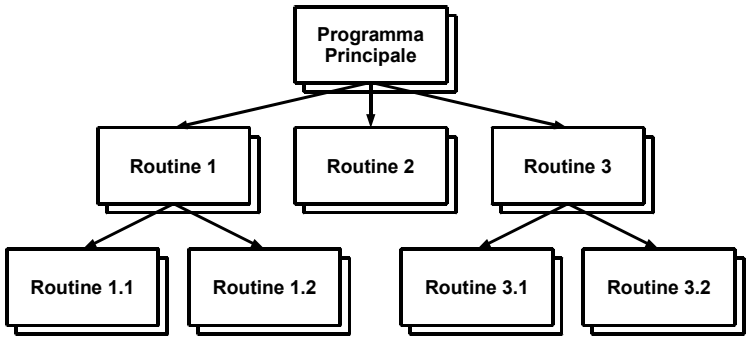
Controllo centralizzato

- Esiste un sottosistema che ha la responsabilità di gestire il controllo complessivo
- Modello *call-and-return*
  - Il controllo inizialmente parte dalla routine alla radice della gerarchia e si sposta verso il basso. Questo tipo di controllo si può applicare a sistemi sequenziali
- Manager model
  - Può essere applicato a sistemi concorrenti. Una componente del sistema coordina l'attivazione, l'arresto e il coordinamento delle altre componenti. Nei programmi sequenziali si implementa come un blocco "case"

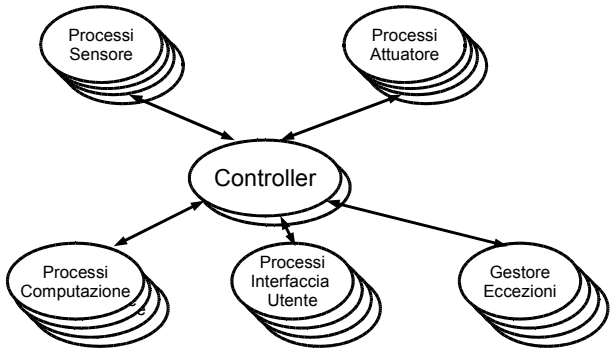
Architettura call-and-return

- E' basata su una struttura gerarchica dove un programma principale richiama una serie di procedure
- Nella sua modalità classica consiste in una struttura a subroutine annidiate
- Le varie componenti possono essere attive su nodi di calcolo distribuiti: in questo caso l'architettura è di tipo "a chiamata di procedure remote"
- Un'evoluzione di questo schema architetturale ha portato alle architetture orientate agli oggetti

Esempio di architettura call-return



Modello basato sul manager



Controllo basato su eventi

- Eventi esterni attivano i sottosistemi che sono responsabili della loro gestione
- Modelli basati su broadcast
  - Lo stesso evento viene inviato a tutti i sottosistemi contemporaneamente. Ciascuno dei sottosistemi esamina l'evento, e se è in grado di trattarlo lo fa
- Modelli basati su interrupt
  - Utilizzati in sistemi real-time, dove gli interrupt sono raccolti da un gestore delle interruzioni e passati alle componenti del sistema responsabili della loro gestione

Modello broadcast

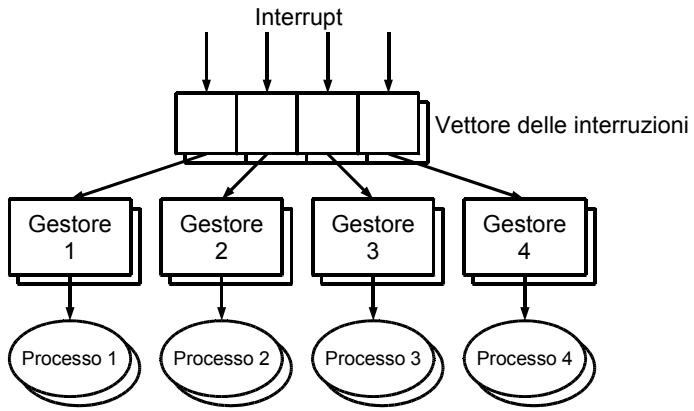
- E' molto utile per integrare diversi sistemi collegati in rete
- Ciascun sottosistema si registra per ricevere particolari tipi di eventi. Quando questi eventi si verificano, il controllo viene trasferito ai sottosistemi registrati
- I sottosistemi decidono gli eventi di interesse. Il gestore degli eventi è solo responsabile di registrare l'interesse dei vari sottosistemi a ricevere certi tipi di eventi

Modello broadcast

Modelli basati su interrupt

- Sono utilizzati in sistemi real-time dove è essenziale che il sistema reagisca in tempi rapidi
- Devono essere definiti tutti i tipi di interrupt, e a ciascun tipo occorre associare un modulo in grado di gestire quel particolare tipo di interrupt
- Questo modello garantisce un rapido tempo di risposta
  - Tuttavia sistemi basati su interrupt possono essere difficili da programmare e da validare

Modello basato su interrupt



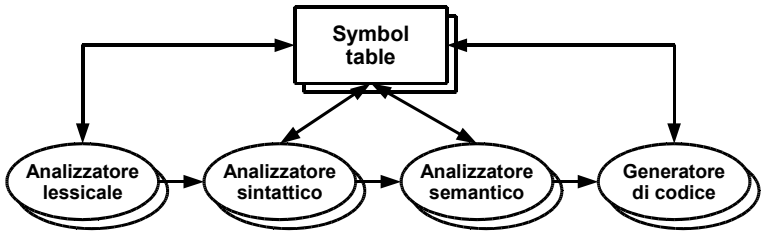
Architetture per domini applicativi specifici

- Si tratta di modelli architetturali che sono specifici a determinati domini applicativi
- Esistono due tipi di modelli per domini specifici
  - *Modelli Generici* che rappresentano astrazioni di un numero di sistemi reali, e rappresentano le principali caratteristiche di tali sistemi
  - *Modelli di Riferimento* che sono modelli più astratti e idealizzati. Forniscono informazioni su una particolare classe di sistemi e vengono utilizzati per confrontare architetture diverse
- I modelli generici sono solitamente bottom-up. I modelli di riferimento sono solitamente top-down

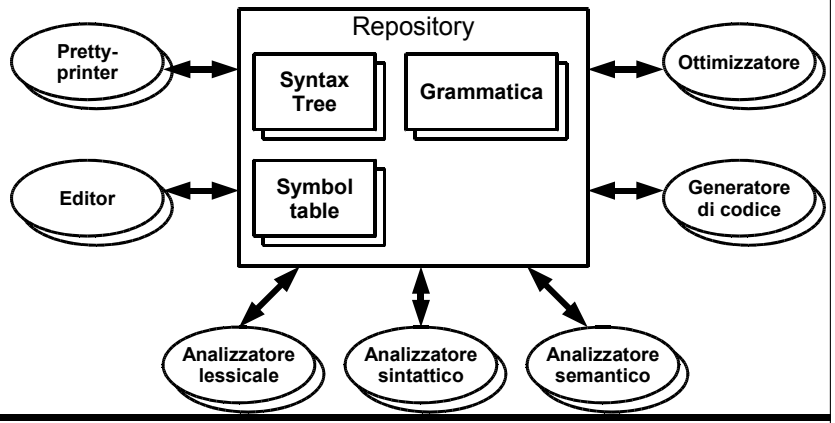
Modelli generici

- Il modello di un compilatore è un esempio ben noto
  - Analizzatore lessicale
  - Tabella dei simboli (symbol table)
  - Analizzatore sintattico
  - Albero di derivazione
  - Analizzatore semantico
  - Generatore di codice
- Il modello generico di compilatore può essere organizzato in base a diversi modelli architetturali

Compilatore basato su architettura a filtri



Compilatore basato sul modello a repository



Modelli di riferimento

- I modelli di riferimento derivano dallo studio del determinato dominio applicativo piuttosto che dallo studio di sistemi esistenti
- Possono essere utilizzati come base per una implementazione, o per confrontare sistemi diversi.
  - I modelli di riferimento possono rappresentare degli "standard" verso cui i sistemi sono valutati
- Esempio classico: il modello OSI di comunicazione

Modello OSI di riferimento