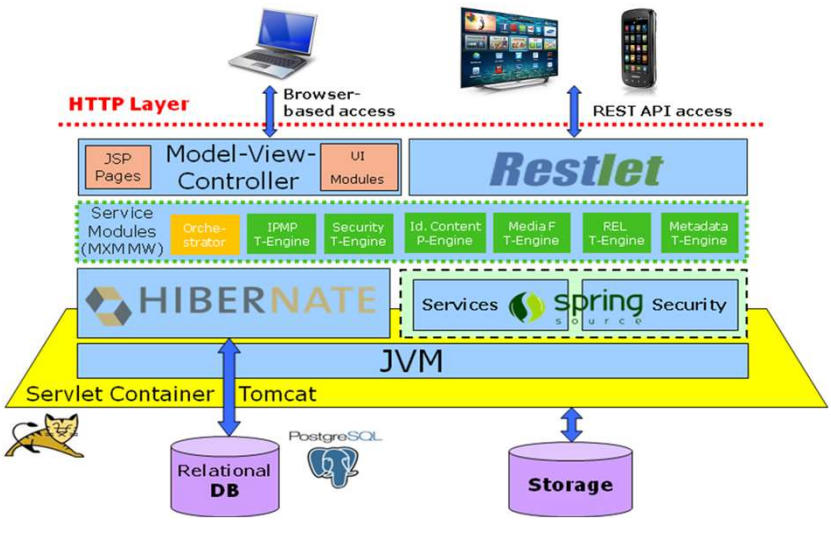
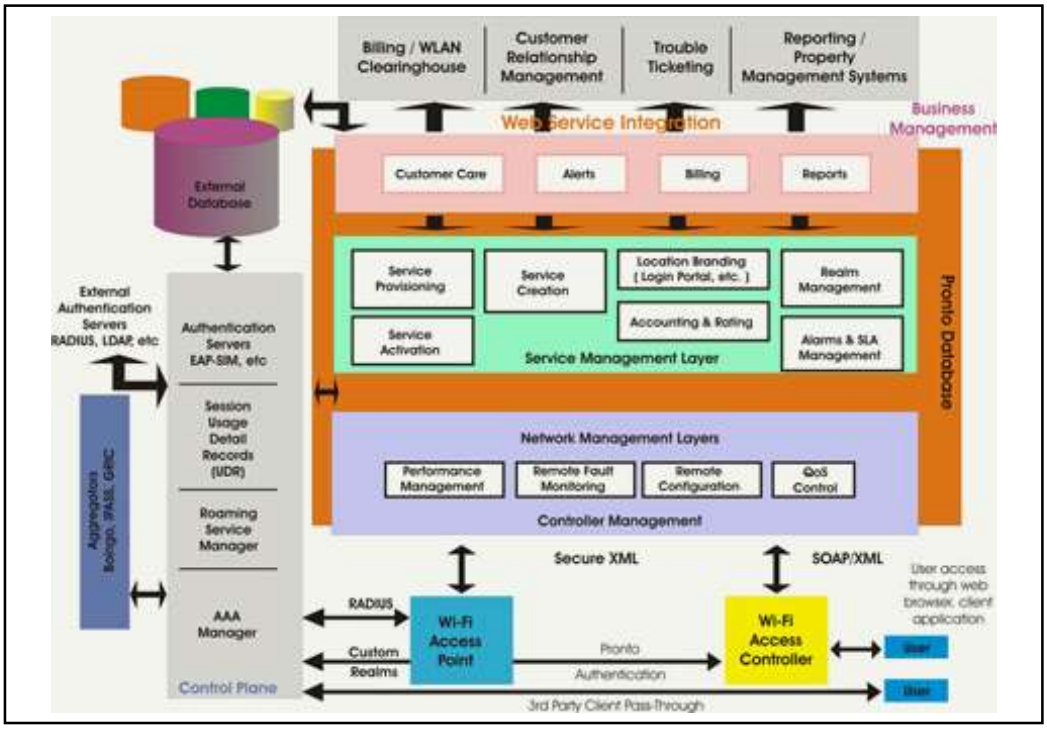
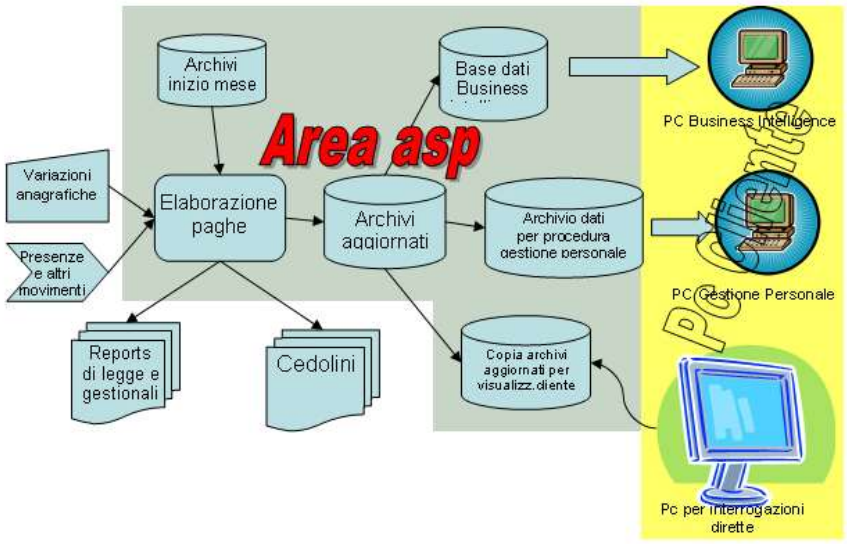
Architetture Software

Roberta Gori, Laura Semini
Ingegneria del Software
Dipartimento di Informatica
Università di Pisa

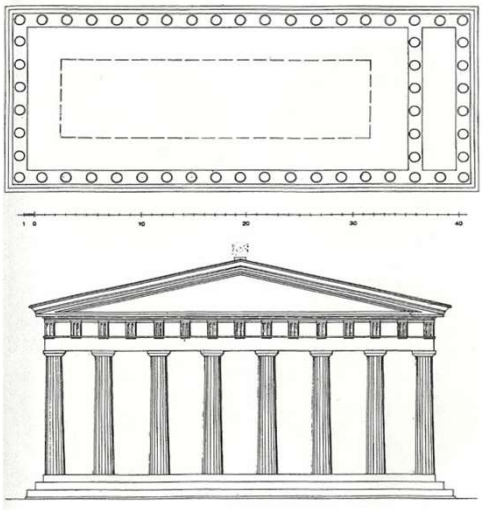
Esempio di "descrizione" di una architettura...




Billing / WLAN Clearinghouse
Customer Relationship Management
Trouble Ticketing
Reporting / Property Management Systems
Web Service Integration
Business Management
External Database
Customer Care
Alerts
Billing
Reports
External Authentication Servers RADIUS, LDAP, etc
Authentication Servers EAP-SIM, etc
Service Provisioning
Service Creation
Location Branding (Login Portal, etc.)
Realm Management
Service Activation
Accounting & Rating
Alarms & SLA Management
Service Management Layer
Pronto Database
Session Usage Detail Records (UDR)
Roaming Service Manager
Network Management Layers
Performance Management
Remote Fault Monitoring
Remote Configuration
QoS Control
Controller Management
Secure XML
SOAP/XML
AAA Manager
RADIUS
Custom Realms
Wi-Fi Access Point
Pronto Authentication
Wi-Fi Access Controller
User access through web browser, client application
Control Plane
3rd Party Client Pass-Through

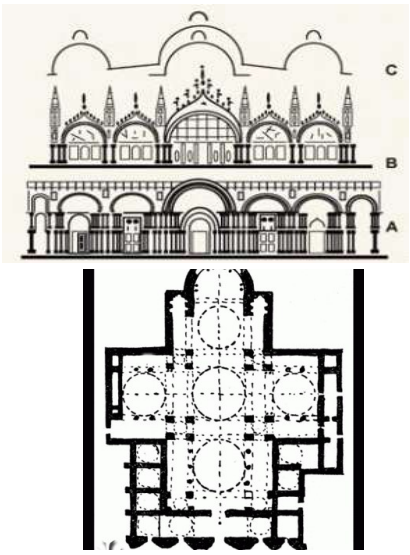

Archivi inizio mese
Base dati Business
Area asp
PC Business Intelligence
Variazioni anagrafiche
Elaborazione paghe
Archivi aggiornati
Archivio dati per procedura gestione personale
Presenze e altri movimenti
PC Gestione Personale
Reports di legge e gestionali
Cedolini
Copia archivi aggiornati per visualizz. utente
Pc per interrogazioni dirette

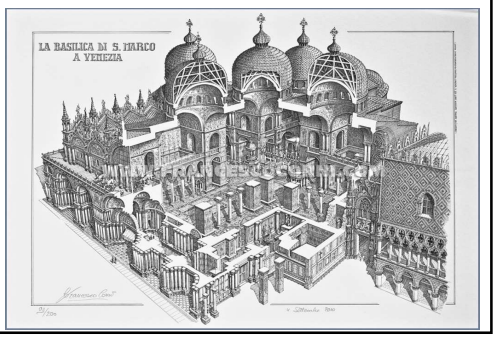
Una descrizione chiara a tutti



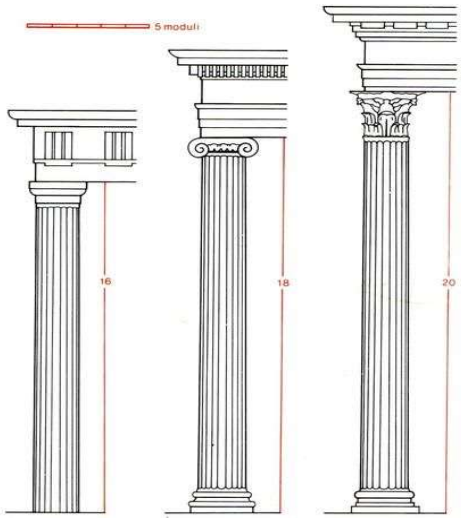
- Pianta
  - vista dall'alto
  - proiezione sul piano xy
- Prospetto
  - vista frontale
  - proiezione sul piano xz (o yz)

Proiezioni (prospetto e pianta) e spaccati: un monumento, diverse viste

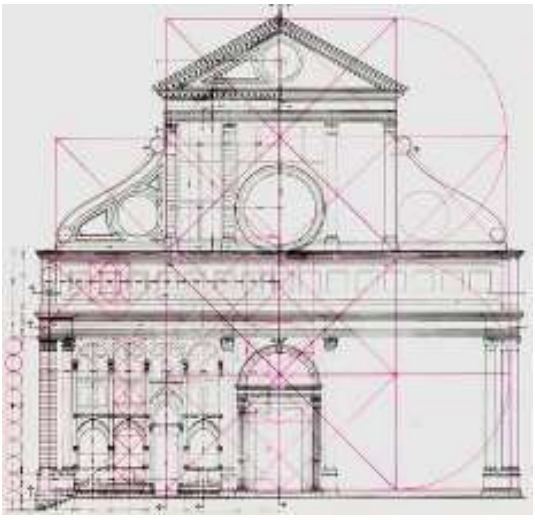




Un componente, la colonna, una vista comune (il prospetto), stili diversi



Pattern da applicare: schemi di progettazione (e.g. schemi proporzionali [Vitruvio, Alberti])



S. Maria Novella
L.B. Alberti

roadmap

- Ruolo e usi (della descrizione) di un'architettura
- Viste, stili e tipi
- Tipi di viste
  - strutturali
  - comportamentali
  - logistiche
- Vista d'insieme

problemi "architettonici"

- Per le prestazioni
  - decomposizione in processi cooperanti
  - controllo del volume di comunicazioni e di accessi ai dati
  - identificazione di colli di bottiglia
- Per la modificabilità e la portabilità
- Per il rilascio incrementale
  - semplicità delle relazioni di dipendenza
- Per la sicurezza
  - controllo delle relazioni d'uso e delle comunicazioni tra le parti
  - identificazione di parti vulnerabili da attacchi esterni
  - introduzione di componenti fidate

architettura software

- […] la progettazione e la descrizione della struttura complessiva del sistema emerge come un nuovo tipo di problema. Queste questioni strutturali includono l'organizzazione di massima e la struttura del controllo; i protocolli di comunicazione, sincronizzazione e accesso ai dati… [Garlan & Shaw 1993]
- L'AS è l'organizzazione di base di un sistema, espressa dai suoi componenti, dalle relazioni tra di loro e con l'ambiente, e i principi che ne guidano il progetto e l'evoluzione [ANSI/IEEE 1471–2000, Recommended Practice for Architecture Description of Software-Intensive Systems, il meno normativo]

[superseded by ISO/IEC/IEEE 42010:2011, *Systems and software engineering — Architecture description*]

ancora l'architettura

- L'architettura software è l'insieme delle strutture del sistema, costituite dalle componenti software, le loro proprietà visibili e le relazioni tra di loro [Bass, Clemens & Kazman 1998]
  - Le "proprietà visibili" di una componente definiscono le assunzioni che le altre componenti possono fare su di essa, come servizi forniti, prestazioni, uso di risorse condivise, trattamento di malfunzionamenti, ecc.
- L'architettura di un sistema software (in breve architettura software) è la struttura del sistema, costituita dalle parti del sistema, dalle relazioni tra le parti e dalle loro proprietà visibili

in altre/altrui parole

- L'architettura
  - definisce la struttura del sistema sw
  - specifica le comunicazioni tra componenti
  - considera aspetti non funzionali
  - è un'astrazione
  - è un artefatto complesso → viste

comunicare tra/per le parti interessate

- tra architetto e cliente: negoziazione di requisiti
- tra architetto e progettisti: negoziazione di risorse (a tempo d'esecuzione)
- per progettisti di sistemi interagenti: definizione di interfacce e protocolli
- per progettisti: definizione di vincoli e possibilità
- per verificatori e integratori: specifica del comportamento delle parti
- per manutentori: base per l'analisi delle conseguenze di modifiche
- per gestori: base per la formazione di gruppi di lavoro, decomposizione del lavoro, allocazione delle risorse, traccia dello stato di avanzamento

ancora sull'uso di AS

- Educazione
  - di nuovi membri, analisti esterni, nuovo architetto
- Analisi
  - delle prestazioni
  - della sicurezza
  - della disponibilità (availability)
  - ...

Viste sull'AS

- Una vista è la rappresentazione di un insieme di elementi del sistema e delle loro proprietà e relazioni
- Una vista espone differenti attributi di qualità in gradi differenti
  - una vista a livelli espone la portabilità
  - una vista di "deployment" espone le prestazioni e l'affidabilità
- Ciascuna vista astrae da informazioni irrilevanti per quel punto di vista

Documentazione di un'AS

- Documentazione di ciascuna vista rilevante
  - visione complessiva, spesso grafica
  - catalogo degli elementi
  - la specifica delle interfacce e del comportamento degli elementi
  - razionale
- Documentazione globale
  - Introduzione e guida alla documentazione
  - Relazioni tra le viste
  - Razionale e vincoli globali
- La rilevanza dipende dagli obiettivi

Tipi di vista (Viewtypes)

- Un tipo di vista definisce i tipi degli elementi e delle relazioni usati per descrivere l'AS da un particolare punto di vista
- Tre punti di vista simultanei sul sistema
  - la struttura come insieme di unità di realizzazione
    - tipo di vista strutturale
  - la struttura come insieme di unità con comportamenti e interazioni, a tempo d'esecuzione
    - tipo di vista comportamentale (component-and-connector (C&C))
  - le relazioni con strutture diverse dal software nel contesto del sistema
    - tipo di vista logistico (problemi di allocazione)

Stili (o schemi)

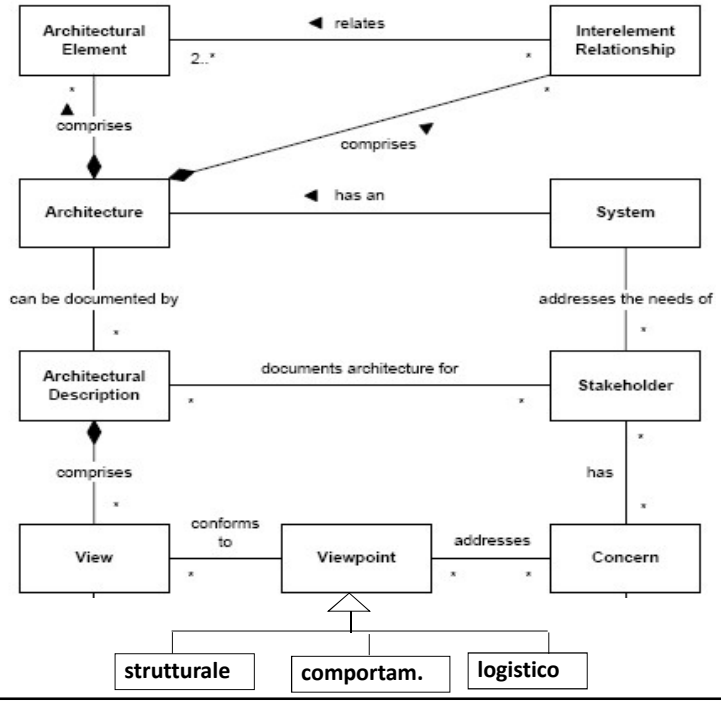
- Uno stile è una proprietà di una architettura
- Uno stile caratterizza una famiglia di architetture con caratteristiche comuni
  - stile a macchine virtuali: i moduli definiscono macchine virtuali
  - stile client-server: caratterizzato da particolari interazioni tra componenti
- Vedremo gli stili di uso comune

La definizione di ciascuna vista è strutturata come segue

- Elementi e Relazioni.
- Proprietà
  - Di elementi e relazioni
- Usi
  - Utilità della vista
- Notazione
  - Costrutti UML utilizzati

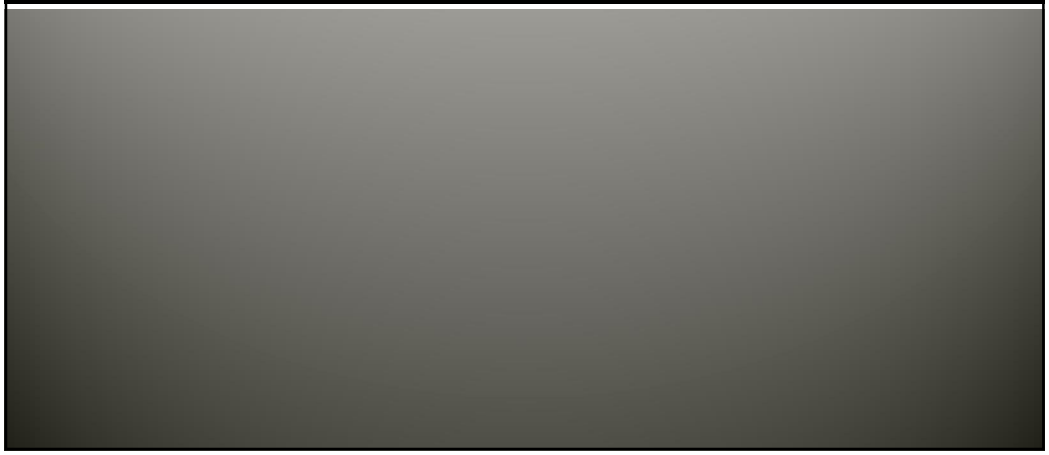
Vista d'insieme: la documentazioe completa

- Organizzazione della documentazione
  - Guida alla documentazione
    - descrizione delle parti
    - elenco delle viste
    - usi per le parti interessate
  - Schema di presentazione di una vista
- L'architettura
  - Descrizione del sistema
  - Corrispondenze tra viste
  - Elenco degli elementi, con riferimento alla definizione
  - Glossario e lista degli acronimi
- Motivazioni
  - Contesto, vincoli di progettazione, giustificazioni delle scelte



Viste di tipo comportamentale

Aka C&C aka componenti e conettori



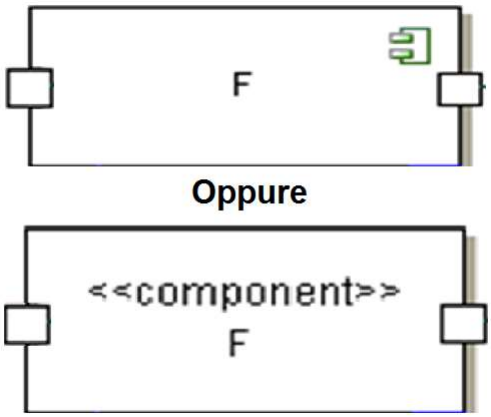
Viste di tipo comportamentale

- Elementi
  - Componenti: entità presenti a tempo d'esecuzione
    - processi, oggetti, serventi, depositi di dati, …
    - A component represents a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment.
  - Connettori: canali di interazione
    - protocolli, flussi d'informazione, accessi ai depositi, …
- Proprietà
  - Componenti
    - nome, tipo (numero e tipo dei porti), altre informazioni specializzate: prestazioni, affidabilità, …
  - Connettori
    - nome, tipo (numero e tipo dei ruoli) , protocollo, prestazioni, …
- Relazioni
  - connessione di ruoli a porti

Viste di tipo comportamentale(2)

- Usi
  - analisi delle caratteristiche di qualità a tempo d'esecuzione
    - prestazioni, affidabilità, disponibilità, sicurezza, ...
  - comunicazione della struttura del sistema in esecuzione
    - flusso dei dati, dinamica, parallelismo, replicazioni, ...

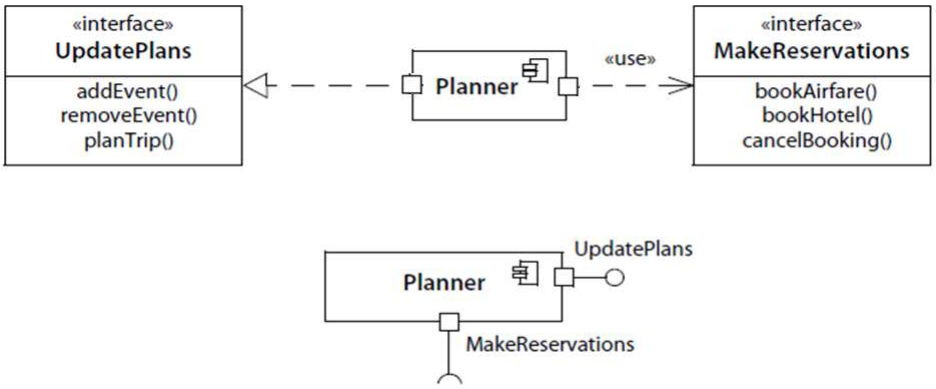
Componenti: notazione UML



Porti

- I quadratini sono "porti", che identificano i punti di interazione di un classificatore (classe, componente), e sono caratterizzati dalle interfacce che forniscono o richiedono.
- Possono avere associata una molteplicità con l'usuale sintassi (1..n).
- Le interfacce sono descritte in modo sintetico con lollipop e forchette, oppure in modo esteso, per mostrare le operazioni richieste/offerte.

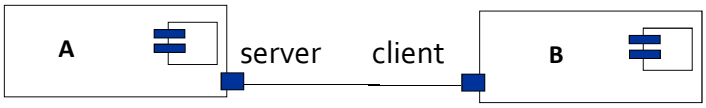
Interfacce: definizione estesa vs sintetica



Conettori: notazione UML

- Un connettore definisce un protocollo di interazione tra le componenti e contribuisce a definire lo stile di una architettura.
- Non ha un descrittore specifico, si usa il simbolo di associazione (linea tra due porti). Per documentare lo stile architetturale si usa:
- uno stereotipo:
  - <<clientServer>>
  - <<dataAccess>> (specializzazione di clientServer)
  - <<pipe>>
  - <<peer2peer>> (<<p2p>>)
  - <<publish-subscribe>>

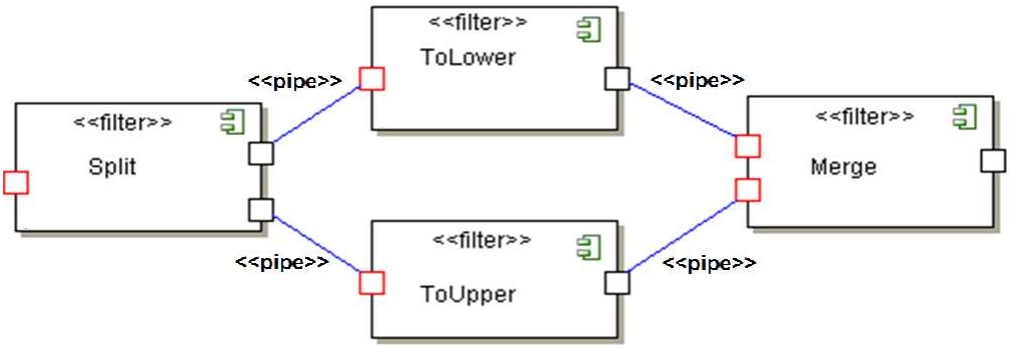
- Oppure si indicano i ruoli delle componenti:



Stili comportamentali: condotte e filtri

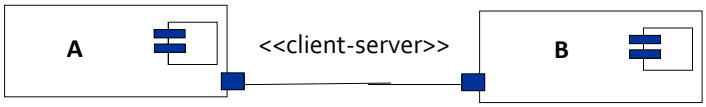
- Componenti
  - filtro: trasforma uno o più flussi di dati dai porti d'ingresso in uno o più flussi sui porti d'uscita
- Connettori
  - condotta (pipe): canale di comunicazione unidirezionale bufferizzato che preserva l'ordine dei dati dal ruolo d'ingresso a quello d'uscita
- Usi
  - pre-elaborazione in sistemi di elaborazione di segnali
  - analisi dei flussi dei dati, e.g. dimensioni dei buffer

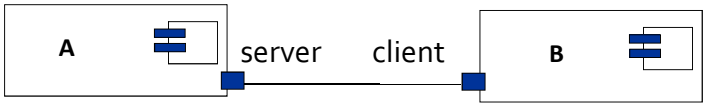
Condotte e filtri: esempio



Stili comportamentali: client server

- Cliente-servente (Client-server)
  - le componenti clienti chiedendo servizi alle componenti serventi



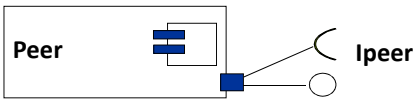


Stile client-server

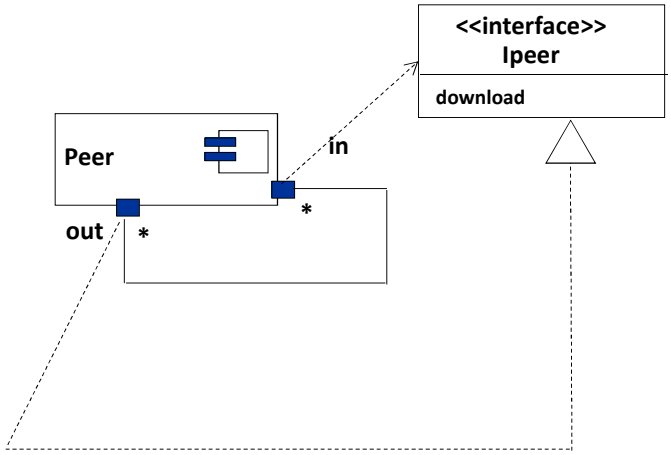
- l sistema è formato da due componenti : il client e il server (spesso, ma non necessariamente, eseguiti su macchine diverse collegate in rete)
- Il server svolge le operazioni necessarie per realizzare un servizio: ad esempio gestisce una banca dati, gestisce l'aggiornamento dei dati e la loro integrità,....
  - Aspetta le richieste dei client a un porto
- Il client provvede ad inviare al server le richieste ed attende una risposta

Stili comportamentali: peer2peer, caso particolare di client-server

- Da pari a pari (peer to peer)
  - tutti i programmi agiscono sia da client sia da server. Es: i programmi di scambi audio e video (WinMx, Kazaa, eMule, ecc.)
  - Scambio di servizi alla pari



P2P



Stili comportamentali: master-slave

- Non lontano da client-server, ma risponde a esigenze diverse
- Il servente (slave) serve un solo cliente (master)
- Architettura usata per esempio nella replica di database, il database master è considerato come fonte autorevole e i database slave sono sincronizzati con esso.

Stili comportamentali: Publish-subscribe

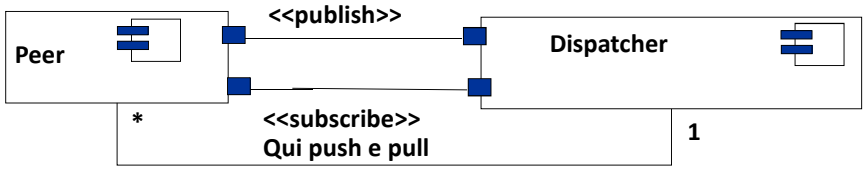
- Le componenti interagiscono annunciando eventi: ciascuna componente si "abbona" a classi di eventi rilevanti per il suo scopo
- Ciascuna componente, volendo, può essere sia produttore che consumatore di eventi
- Disaccoppia produttori e consumatori di eventi e favorisce le modifiche dinamiche del sistema

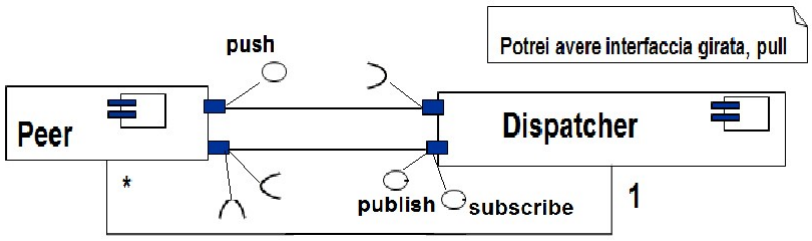
Publish-subscribe

- In questo stile, mittenti e destinatari di messaggi dialogano attraverso un tramite, che può essere detto dispatcher o broker. Il mittente di un messaggio (detto publisher) non deve essere consapevole dell'identità dei destinatari (detti subscriber); esso si limita a "pubblicare" (in inglese to publish) il proprio messaggio al dispatcher. I destinatari si rivolgono a loro volta al dispatcher "abbonandosi" (in inglese to subscribe) alla ricezione di messaggi. Il dispatcher quindi inoltra ogni messaggio inviato da un publisher a tutti i subscriber interessati a quel messaggio.
- In genere, il meccanismo di sottoscrizione consente ai subscriber di precisare nel modo più specifico possibile a quali messaggi sono interessati. Per esempio, un subscriber potrebbe "abbonarsi" solo alla ricezione di messaggi da determinati publisher, oppure aventi certe caratteristiche.
- Questo schema implica che ai publisher non sia noto quanti e quali sono i subscriber e viceversa. Questo può contribuire alla scalabilità del sistema.

Publish-subscribe

Operazioni: subscribe, unsubscribe, publish, notify (push), letMeKnow (pull)





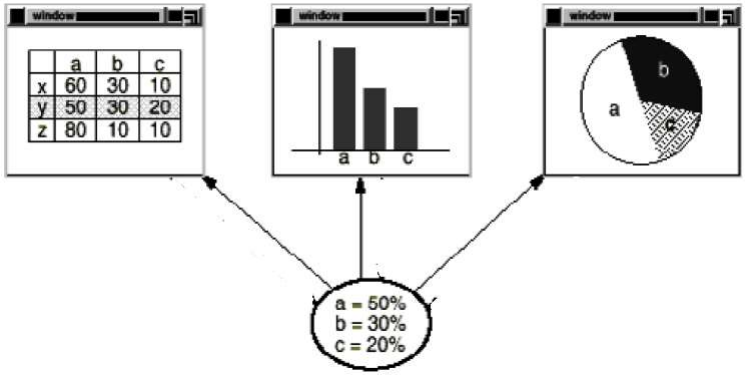
Publish-subscribe: middleware e reti

- Reti: multicast con algoritmi di flooding
- Il Data Distribution Service for Real Time Systems (DDS) è uno standard emanato dall'Object Management Group (OMG) che definisce un middleware per la distribuzione di dati in tempo reale secondo il paradigma publish/subscribe.

Stili comportamentali: Model–View–Controller (MVC)

- Stile in cui si isola la logica di business dal controllo sull'input e dalla presentazione (vista sui dati), consentendo sviluppo indipendente, test e manutenzione di ciascuno.
- Modello
  - È la rappresentazione del dominio dei dati su cui opera l'applicazione. Quando un modello cambia il suo stato, notifica le sue viste associate in modo che si possano aggiornare.
- Vista
  - Rende il modello in una forma adatta all'interazione, in genere un elemento dell'interfaccia utente. Ci possono essere più viste per un singolo modello, per scopi diversi.
- Controllore
  - Riceve l'input e effettua chiamate agli oggetti del modello.

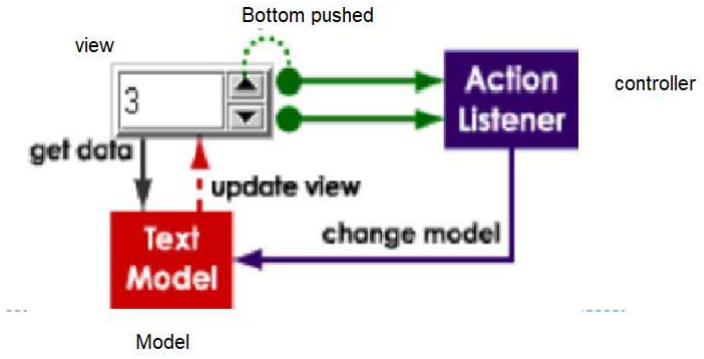
Un modello, diverse viste



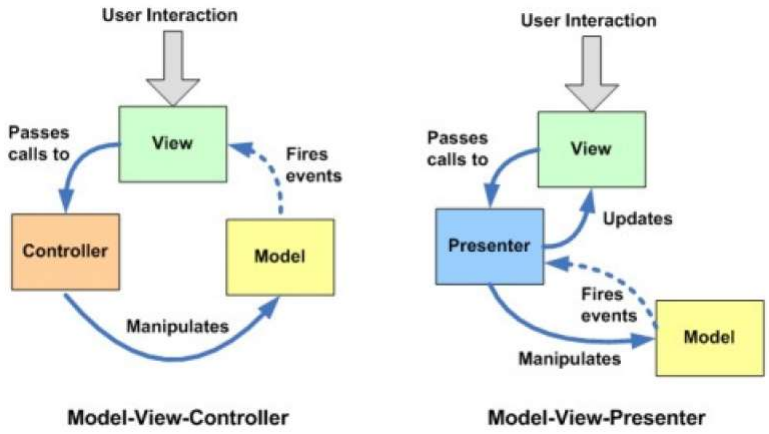
Model-View-controller

- L'utente interagisce con la vista
- Il controller riceve le azioni dell'utente e le interpreta
  - Se si fa clic su un pulsante, è compito del controllore capire che cosa significhi e come il modello dovrebbe essere manipolato in base a tale azione.
- Il controller chiede al modello di cambiare il suo stato
- Il modello notifica la vista quando il suo stato è cambiato
  - Quando qualcosa cambia nel modello, in risposta a qualche azione (es. fare clic su un pulsante) o per altri motivi (es. è iniziato il brano successivo nella playlist), i modello notifica alla vista che il suo stato è cambiato.
- La vista chiede lo stato al modello
  - Per esempio, se il modello notifica la vista che è iniziata un nuovo brano, la vista richiede il nome del brano al modella e lo mostra.

Esempio di MVC



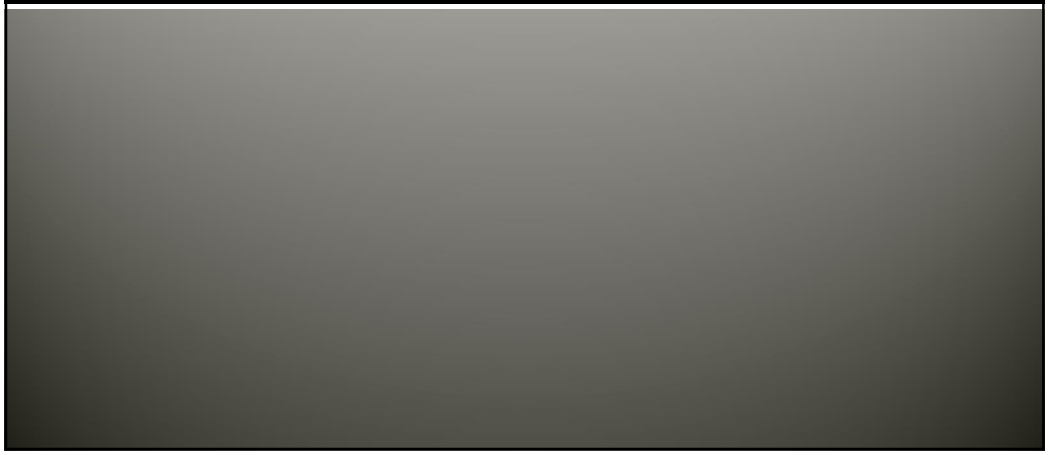
MVC vs Model-view-presenter



Stili Comportamentali: Coordinatore di processi

- Il Coordinatore di processi conosce la sequenza di passi necessari per realizzare un processo.
- Riceve la richiesta, chiama i servers secondo l'ordine prefissato, fornisce una risposta
- Normalmente usato per realizzare processi complessi
- Disaccoppiamento: i server non conoscono il loro ruolo nel processo complessivo né l'ordine dei passi del processo. Ogni server semplicemente definisce un servizio
- Comunicazione flessibile: sincrona o asincrona.

Viste di tipo strutturale



Viste di tipo strutturale

- Elementi
  - Modulo: unità di software che realizza un insieme coerente di responsabilità
  - classe, collezione di classi, un livello.
- Proprietà
  - responsabilità, visibilità, autore,
  - informazioni sulla realizzazione
    - codice associato, informazioni sul test, informazioni gestionali, vincoli sulla realizzazione
- Relazioni
  - parte di, eredita da, dipende da, può usare

Viste di tipo strutturale (2)

- Usi
  - costruzione
    - la vista può fornire lo schema del codice e directory e file sorgente hanno una struttura corrispondente
  - analisi
    - tracciabilità dei requisiti
    - analisi d'impatto per valutare eventuali modifiche
  - comunicazione
    - se la vista è gerarchica, offre una presentazione top-down della suddivisione delle responsabilità nel sistema ai novizi
- Non usi
  - analisi dinamiche
    - cfr viste comportamentali e logistiche
- Notazione: classi e packages + notaz. per relazioni

Notazione: packages

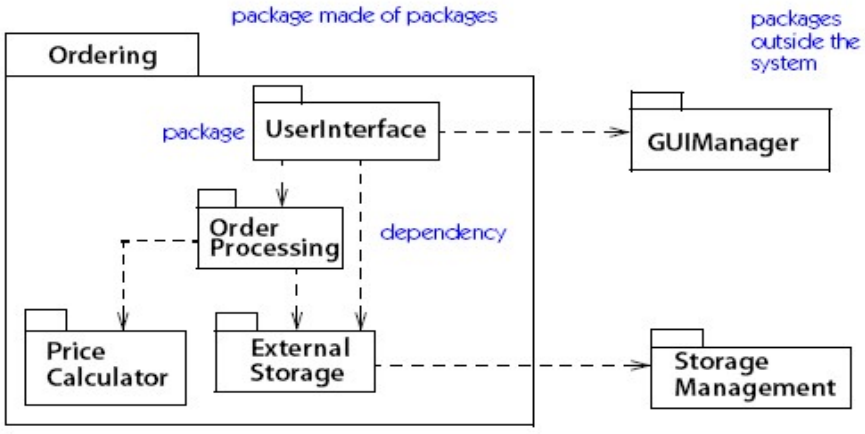
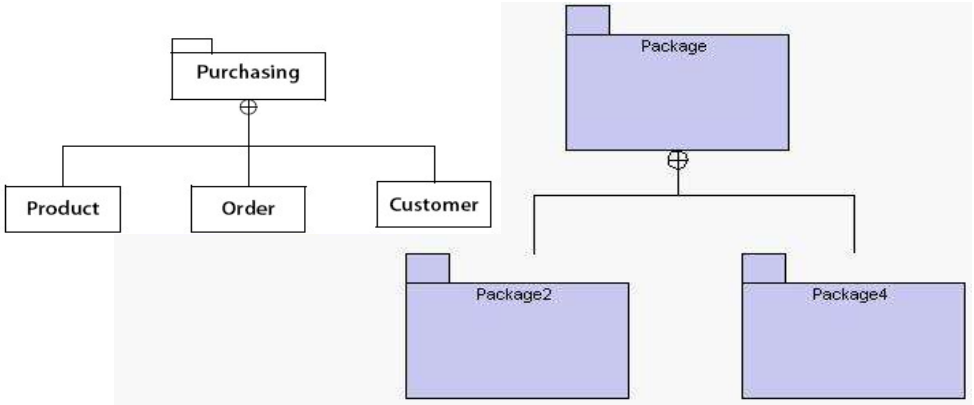


Figure 14-203. *Packages and their relationships*

Vista strutturale di decomposizione

- Relazione "parte di"
  - moduli e sottomoduli
  - registra la campagna di divide-and-conquer
- Criteri
  - incapsulamento per modificabilità
  - supporto alle scelte costruisci/compra
  - moduli comuni in linee di prodotto
- Usi
  - apprendimento del sistema
  - punto di partenza per l'allocazione del lavoro

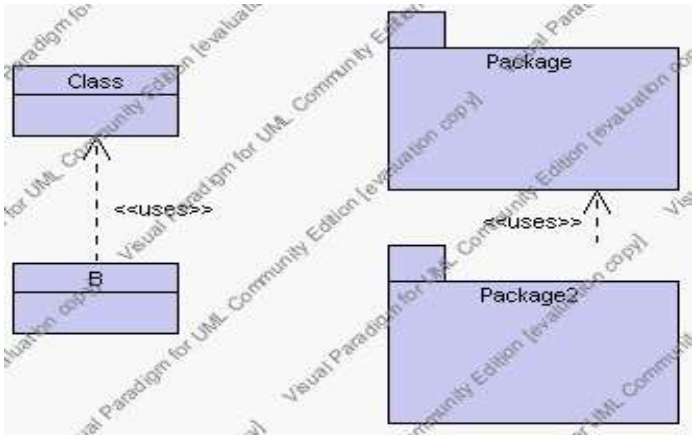
Notazione (oppure contenimento)



Vista strutturale d'uso

- Relazione "usa"
  - il modulo A usa il modulo B se dipende dalla presenza di B (funzionante correttamente) per soddisfare i suoi requisiti
    - un modulo che segnala un errore funziona correttamente indipendentemente da cosa fa il modulo che riceve la segnalazione, per cui lo invoca, ma non lo usa (Non è suo cliente in una dipendenza).
  - abilita lo sviluppo incrementale e il rilascio di sottosistemi utili
  - cicli permessi ma pericolosi
- Usi
  - pianificazione di sviluppo incrementale, di estensioni e ritagli del sistema, del testing incrementale
  - analisi d'impatto

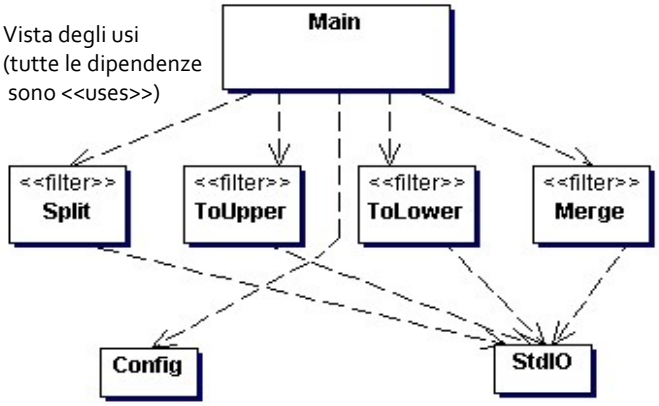
Notazione



Uso

Vista strutturale degli usi (dell'esempio di cui abbiamo già dato vista C&C)

Vista degli usi
(tutte le dipendenze
sono <<uses>>)

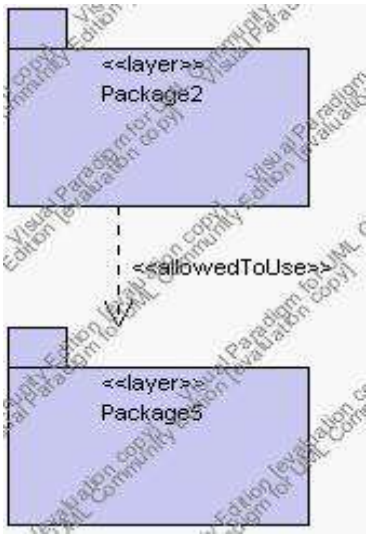


- i filtri si chiamano tra loro, ma non si usano…
- e il main li configura per metterli in comunincazione via StdIO (realizzazione del connettore pipe)
- suggerisce anche una vista a strati…

Vista strutturale a strati (macchine virtuali)

- Elementi: strati
  - uno strato è un insieme coeso di moduli
    - a volte raggruppati in segmenti
  - offre un'interfaccia pubblica per i suoi servizi (macchina virtuale)
- Relazione: può usare
  - antisimmetrica (a meno di eccezioni)
  - transitiva (se esplicitamente detto)
- Usi
  - modificabilità e portabilità
  - controllo della complessità
  - analisi d'impatto

Notazione



Un esempio noto di architettura a stati: vista strutturale o C&C ? (presentazione che cerca di rappresentare entrambe le viste)

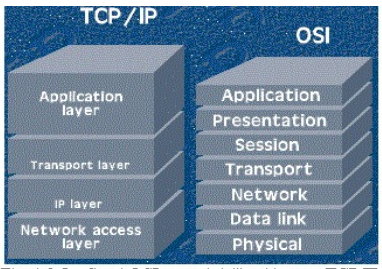
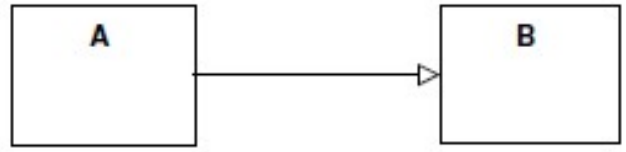


Fig. 1.2.3 - Strati OSI e strati dell'architettura TCP/IP

Telnet	FTP	SMTP	DNS	Application Layer (Layer OSI 5)
TCP			UDP	Transport Layer (Layer OSI 4)
IP ICMP ARP				InterNetwork Layer (Layer OSI 3)
X.25	Ethernet	ISDN	Token-Ring	Network Access (Layer OSI 2)

Fig. 1.2.3 - Strati OSI relativi all'architettura TCP/IP

Vista strutturale di generalizzazione



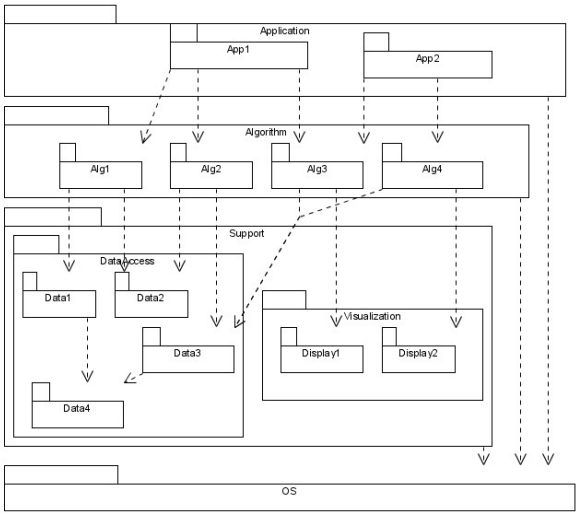
- Framework

Esercizio

Module	Module Type	Invokes Module
APP-1	Application	ALGO-1, ALGO-2, ALGO-3
APP-2	Application	ALGO-3, ALGO-4
ALGO-1	Algorithm	DATA-1, DATA-2
ALGO-2	Algorithm	DATA-2, DATA-3
ALGO-3	Algorithm	DATA-3, DISP-1
ALGO-4	Algorithm	DATA-3, DISP-2
DATA-1	Data Access	DATA-4
DATA-2	Data Access	
DATA-3	Data Access	DATA-4
DATA-4	Data Access	
DISP-1	Display Output	
DISP-2	Display Output	
All Modules use Modules in the Operating System		

a.Sketch a graphical module view that illustrates the uses relationships implied by this table. Could these modules be grouped into packages of modules? Could these modules (or the packages) be grouped into layers? If so, try to incorporate your groupings into your view.
b.Sketch the dependency tree for the APP-1 application.
c.Suppose that we want to create a separate runtime component for each of the applications. Which modules would participate when the APP-2 component is running?
d.Assume that the original plan was to have the data access modules use the file system of the operating system for data management. Now, suppose that we decide to purchase a separate database system for data storage and retrieval. The database, of course, uses modules in the operating system. How would you incorporate this change into your graphical module view?

Soluzione



Viste di tipo logistico

Viste di tipo logistico

- Elementi
  - software: moduli, componenti e connettori
  - dell'ambiente: hardware, struttura di sviluppo, file system
- Relazione
  - elemento software allocato a elemento dell'ambiente
- Proprietà
  - richieste dal software, fornite dall'ambiente
- Usi
  - analisi delle prestazioni (vista di dislocazione - deployment)
  - pianificazione dello sviluppo (vista di assegnamento del lavoro)
  - gestione delle configurazioni (vista di realizzazione)

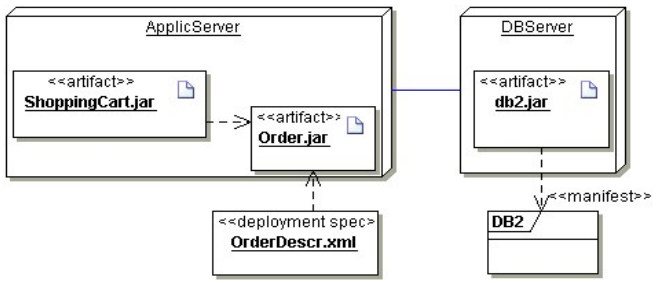
Vista logistica di dislocazione

- Elementi
  - software: componenti e connettori
  - dell'ambiente: hardware, ambiente di esecuzione
- Proprietà del software:
  - consumo di risorse, criticità
- Proprietà dell'ambiente:
  - CPU: frequenza di clock, numero di processori, ...
  - memoria: dimensioni, velocità, dischi, ...
  - canali di comunicazione: ampiezza di banda

Artifact (artefatto)

- Un artefatto è un'informazione fisica che viene utilizzata o prodotta da un processo di sviluppo software o dal funzionamento di un sistema.
- Esempi di artefatti includono
  - codice sorgente, script, file eseguibili binari,
  - la tabella di un database,
  - un deliverable di progetto,
  - un documento word,
  - un messaggio di posta.

UML: diagrammi di dislocazione



- Nodo: rappresenta un tipo di hardware; le relazioni tra nodi sono connessioni fisiche. Un nodo può anche rappresentare un abiente di esecuione
- Artefatto: informazione prodotta dallo sviluppo o esecuzione di sw
- Deployment spec: Parametri d'installazione e d'esecuzione
- Manifestazione: dipendenza verso il modello realizzato

Vista logistica di realizzazione

- Elementi
  - software: moduli
  - ambiente: elementi di configurazione (configuration items)
    - spesso gerarchici
- Usi
  - gestione dei file che corrispondono agli elementi software
  - gestione delle versioni

Vista di realizzazione, esempio

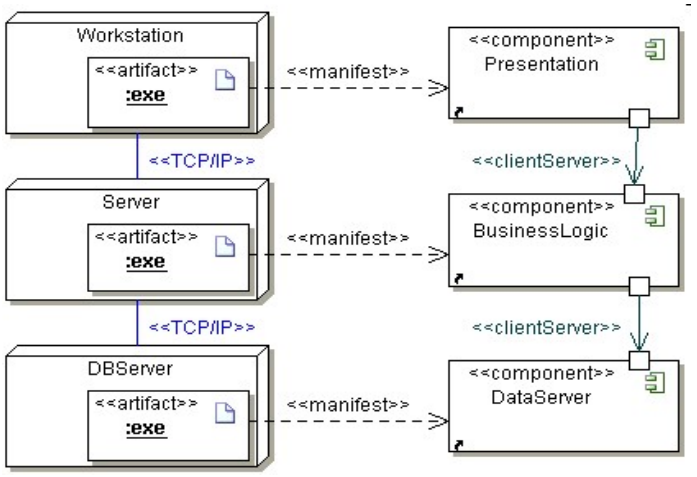
- La directory che realizza il modulo nomemodulo contiene
  - make
  - src
    - NomeModulo.cc
  - include
    - NomeModulo.h
  - doc
    - NomeModulo.doc
  - config
    - NomeModulo.conf

Stili logistici: assegnamento del lavoro

- Elementi
  - software: moduli
  - ambiente: unità organizzativa
    - persona, gruppo, dipartimento, sotto-contraente, …
- Proprietà dell'assegnamento
  - completezza e non sovrapposizione
- Usi
  - comunicazione della struttura del progetto
  - base della WBS (struttura di decomposizione del lavoro) e delle stime dettagliate di costi e tempi

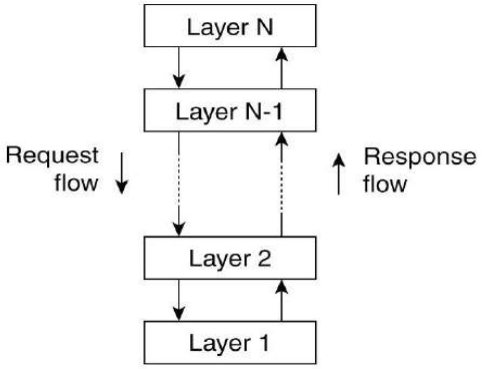
Viste ibride (C&C e dislocazione)

- 3-tier



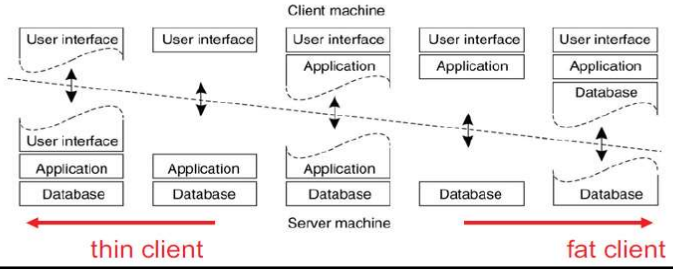
Architettura a livelli

- Componenti organizzati in livelli (*layer*)
- 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



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

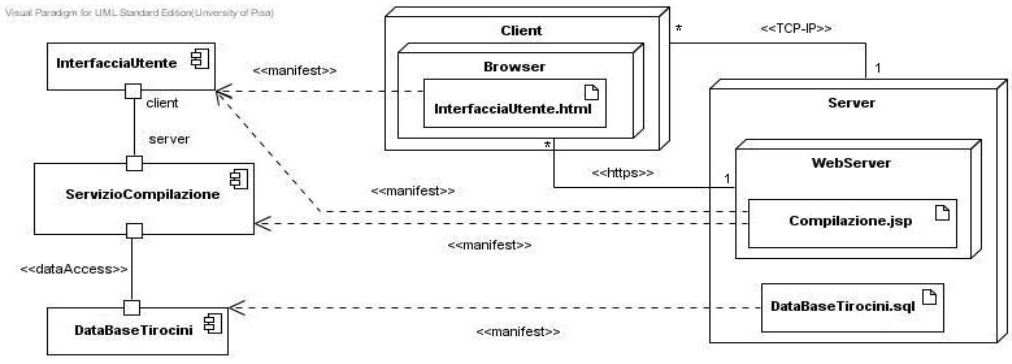


Architetture multilivello (cont'd)

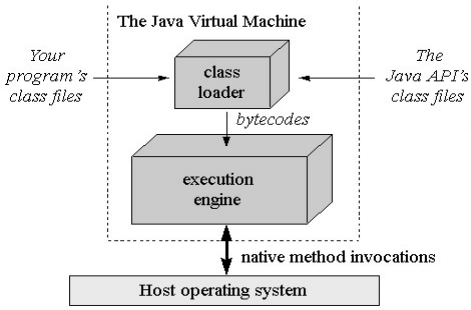
- Da un livello ad N livelli
- Con l'introduzione di ciascun livello
  - L'architettura guadagna in flessibilità, funzionalità e possibilità di distribuzione
- Ma l'architettura multilivello potrebbe introdurre un problema di prestazioni
  - Aumenta il costo della comunicazione
  - Viene introdotta più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

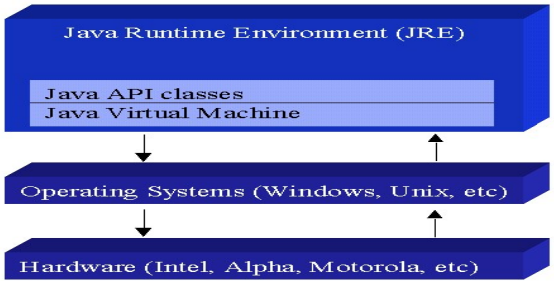
Esempio
Tirocini: 3-layers, 2-tier

Vista ibrida (C&C e dislocazione) dell'architettura del sotto-sistema di Compilazione, assumendo che gli artefatti che manifestano le componenti citate siano: Compilazione.html, visualizzato da un browser di una macchina client e Compilazione.jsp (dislocata su un web server) e DataBaseTirocini.sql, mantenute su una macchina server.



Un altro esempio : capire quali viste sarebbero adatte

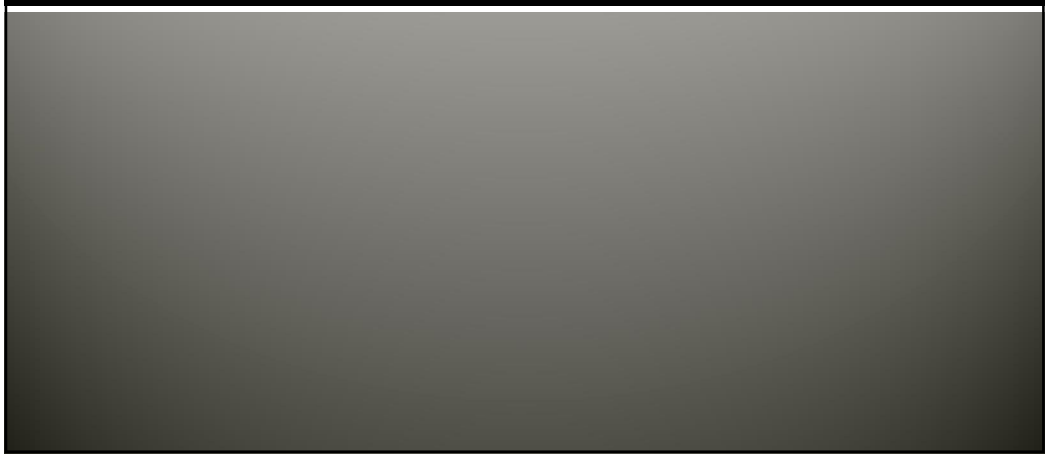




- Componente (JVM)
  - Sotto-Componente (loader) +
  - Sotto-Componente (engine)
- Ambiente di esecuzione (SO)

- Codice (API classes) +
- Componente (JVM) +
- Ambienti di esecuzione (JRE, SO, HW)

Stili dell'architettura e qualità del software



Qualità analizzate

- Valutiamo le caratteristiche di alcuni stili architetturali in base alle seguenti caratteristiche di qualità:
  - Disponibilità
    - Capacità di offrire un servizio in modo il più possibile costante
  - Fault tolerance
  - Modificabilità
  - Performance (efficienza)
  - Scalabilità

Scalabilità

- La scalabilità di un'applicazione può essere definita come la capacità di aumentare il throughput dell'applicazione in proporzione all'aumento dell'hardware utilizzato per ospitare l'applicazione.
- In altre parole, se un'applicazione è in grado di gestire 100 utenti con su un singolo nodo hardware, l'applicazione dovrebbe essere in grado di gestire 200 utenti quando il numero di nodi hw è raddoppiato.

Scale up, scalabilità verticale

- La scalabilità verticale si riferisce all'aggiunta di memoria e CPU a una singolo nodo.
- La scalabilità verticale è particolarmente utile con i database. I database hanno le seguenti necessità:
  - Grande spazio di memoria condivisa,
  - Molti thread dipendenti,
  - Alto grato di interconnessione interna

Scale out, scalabilità orizzontale

- Con scalabilità orizzontale si intende l'aggiunta di più nodi hw. La scalabilità orizzontale è ideale per applicazioni Web, che presentano alcune delle seguenti caratteristiche:
  - Poco spazio di memoria condiviso
  - Molti thread indipendenti
  - Bassa interconnessione
- Possibile anche con sistemi operativi diversi

Client-server, 2 o N-tier

Disponibilità	I server di ogni tier(ordine, fila) possono essere replicati, quindi se uno fallisce c'è solo una minor QoS.
Fault tolerance	Se un cliente sta comunicando con un server che fallisce, la maggior parte dei server reindirizza la richiesta a un server replicato in modo trasparente all'utente.
Modificabilità	Il disaccoppiamento e la coesione tipici di questa arch. favoriscono la modificabilità
Performance (efficienza)	Performance ok, ma da tenere sott'occhio: numero di threads paralleli su ogni server, velocità delle comunicazioni tra server, volume dati scambiato
Scalabilità	Basta replicare i server in ogni tier (quindi ok scale out). Unico collo di bottiglia l'eventuale base di dati che scala male orizzontalmente

Pipes and Filters

Disponibilità	Avendo "pezzi di ricambio" (componenti e possibilità di connetterle) sufficienti a formare una catena.
Fault tolerance	Occorre riparare una catena interrotta usando componenti replica.
Modificabilità	Si, se le modifiche interessano una o comunque poche componenti
Performance (efficienza)	Dipende dalla capacità del canale di comunicazione e dalla performance del filtro più lento.
Scalabilità	Ok anche scale out.

Publish-subscribe

Disponibilità	Si possono creare clusters di dispatcher
Fault tolerance	Si cerca un dispatcher replica
Modificabilità	Si possono aggiungere publisher e subscribers a piacere. Unica attenzione al formato dei messaggi.
Performance (efficienza)	Ok. Ma compromesso tra velocità e altri requisiti tipo affidabilità e/o sicurezza.
Scalabilità	Ok scale out: con un cluster di dispatchers si può gestire un volume molto elevato di messaggi.

P2P

Disponibilità	Dipende dal numero di nodi in rete, ma si assume si.
Fault tolerance	Gratis
Modificabilità	Si, se dell'architettura interessa solo la parte di comunicazione
Performance (efficienza)	Dipende dal numero di nodi connessi, dalla rete, dagli algoritmi. Per esempio BitTorrent ottimizza scaricando per primo il file/pezzo più raro.
Scalabilità	Gratis

Coordinatore di processi

Disponibilità	Il coordinatore è un punto critico: deve essere replicato se si vuole garantire disponibilità.
Fault tolerance	Occorre specificare compensazione: cosa fare se un server fallisce Se fallisce il coordinatore: occorre ridirigere su un coordinatore replica.
Modificabilità	Posso modificare liberamente i servers purché non cambino le funzionalità esportate.
Performance (efficienza)	Il coordinatore deve essere in grado di servire più richieste concorrenti. La performance del processo è limitata dal server più lento. Se non tutti i server sono necessari, si usa un time-out.
Scalabilità	Replicando il coordinatore. Scala sia up che out.

Architetture basate su comunicazione asincrona

Disponibilità	Basta replicare le code
Fault tolerance	Se una coda fallisce, si cerca una replica
Modificabilità	Unico vincolo il formato dei messaggi
Performance (efficienza)	Si possono spedire migliaia di messaggi al secondo. Compromesso tra affidabilità/sicurezza e performance
Scalabilità	Le code possono essere ospitate presso origine e destinazione della comunicazione, ma anche da clusters grandi a piacere di servers.

Syllabus

- Dispensa di architettura, alias Carlo Montangero e Laura Semini, *Architetture software e Progettazione di dettaglio, Note per il Corso di Ingegneria del software, 2014*
- Arlow:
  - cap 11 package di analisi
  - cap 17 interfacce e componenti
  - cap 22 Deployment
- Fuggetta: cap 7, pp129-142

Riferimenti

- Shaw, M e Garlan, D. Software Architecture. Prentice-Hall, 1996.
- Soni, D. et al. Software Architecture in Industrial Application. Proc. 17° Int. Conf. on Software Engineering, 1995.
- Parnas, D. Designing Software for the ease of Extention and Contraction. IEEE Transaction on SE 5(2), 1979.
- Krutchen, P. The 4+1 View Model of Architecture. IEEE Software 12(6), 1995.
- Magee, J. et al. Specifying Distributed Software Architecures. Proc. Fifth Europ. Software Eng. Conf. LNCS 989, Springer Verlag, 1995.
- Clemens, P. et al. Documenting Software Architectures. Addison Wesley, 2003.