Sistemi Informativi:

Progetto del sistema

Argomenti

- Fino a qui il progetto e' stato indipendente dall' implementazione
- Architettura e progetto del software
  - Architetture distribuite
  - 3-tier architecture
  - Programmazione delle basi di dati
  - Strategie di riutilizzo
  - Componenti
  - installazioni
- Collaborazione – progetto dettagliato
  - Diagramma di collaborazione
  - Realizzazione dei casi d' uso
  - Realizzazione delle operazioni

Design

“There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies.”

- C.A.R. Hoare

Difficolta’ del progetto

- L’analisi centrata sul dominio dell’applicazione
- Progetto e implementazione
  - la conoscenza del progetto e’ un obiettivo in movimento (moving target)
  - i motivi delle scelte cambiano rapidamente
    - “tempo di dimezzamento” della conoscenza nell’ ingegneria SW: 3-5 anni
    - fra 3 anni cio’ che insegno sara’ datato? (swing)
    - discesa costo HW
- finestra di opportunita’ del progetto
- tempo in cui occorre fare le scelte di progetto

Round-trip engineering

- Forward engineering
  - Produzione del codice a partire dal progetto
- Backward engineering
  - Aggiornare il progetto in base all'effettiva implementazione del codice
    - Gli ambienti di CASE forniscono livelli variabili di supporto per questa attivita
    - La gestione del progetto deve essere rigorosa per evitare la progressiva divergenza tra modello ad alto livello ed implementazione
    - funziona quando esistono standard di progetto obbligatori

Reengineering

- Modifica / adeguamento di un sistema preesistente (legacy) ad un nuovo ambiente
- da archivi COBOL a RDB
- Da RDB a ORDB
- Se le applicazioni preesistenti espongono una serie di funzionalita’ semplici e stateless (es anagrafe) si preferisce incapsularle dietro una interfaccia **ponte** che ne permetta l’accesso dal nuovo ambiente evitando il costo di reimplementazione

Progetto del sistema (1)

- molte volte tante scelte sono gia’ fatte per progetti standard

- Obiettivi
  - definizione
  - compromessi
- Organizzazione del sistema
  - livelli/partizioni
  - coerenza/coesione
- concorrenza
  - identificazione dei thread

Progetto del sistema (2)

- Mappatura HW-SW
  - speciale, costruzione /acquisto
  - allocazione
  - connettivita’
- Gestione dati
  - Oggetti persistenti
  - file
  - basi dati
  - strutture dati

Progetto del sistema (3)

- Risorse globali
  - controllo accesso
  - sicurezza
- Controllo
  - monolitico
  - ad eventi
  - concorrenza
- condizioni al contorno
  - inizializzazione
  - termine
  - errori

Influenza dell’analisi nel progetto

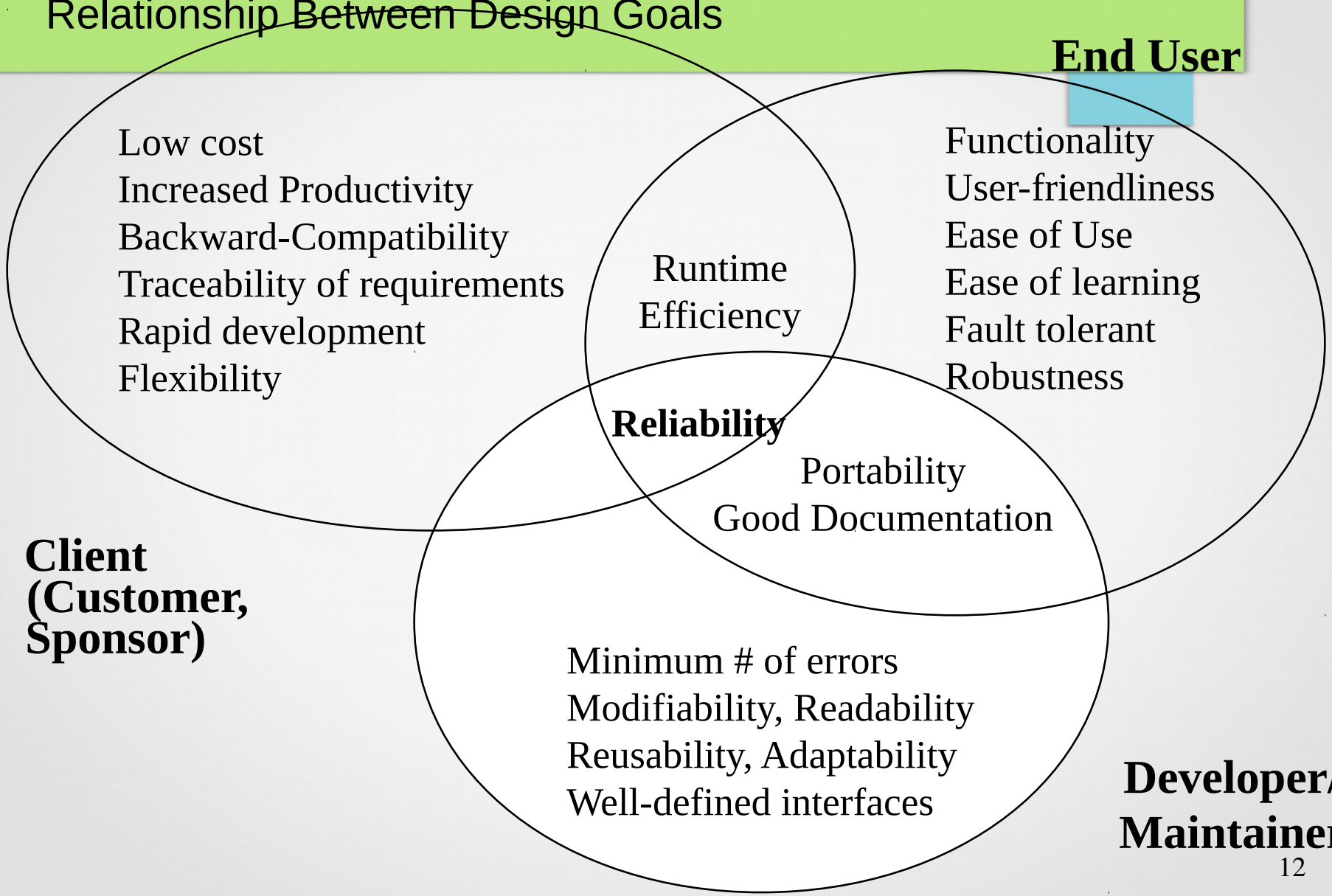
- requisiti non funzionali →
  - obiettivi del progetto
- modello dei casi d’uso →
  - organizzazione in sottosistemi in base a req fun, coerenza, coesione
- modello degli oggetti →
  - mappatura HW/SW
  - gestione dati persistenti
- modello dinamico →
  - concorrenza
  - controllo SW
  - gestione risorse globali
  - condizioni al contorno

obiettivi del progetto

- Reliability
- Modifiability
- Maintainability
- Understandability
- Adaptability
- Reusability
- Efficiency
- Portability
- Traceability of requirements
- Fault tolerance
- Backward-compatibility
- Cost-effectiveness
- Robustness
- High-performance

- Good documentation
- Well-defined interfaces
- User-friendliness
- Reuse of components
- Rapid development
- Minimum # of errors
- Readability
- Ease of learning
- Ease of remembering
- Ease of use
- Increased productivity
- Low-cost
- Flexibility

Relationship Between Design Goals



alcuni compromessi nel progetto

- funzionalita’/usabilita’
- Costo/robustezza
- efficienza/portabilita’
- rapidita’ di sviluppo/funzionalita’
- costo /riusabilita’
- compatibilita’ con pregresso/leggibilita’

organizzazione in sottosistemi

- si parla di decomposizione, ma si puo' vedere come **aggregazione** delle classi individuate nelle fasi precedenti come cicli analisi-sintesi
- il sistema e' organizzato in sottosistemi, ognuno dei quali fornisce dei servizi. Es: servizioPagamento: bonifico(),assegno(),cartacredito,vaglia()
- servizio : gruppo di operazioni fornite da un sottosistema
  - spunto: casi d'uso del sottosistema
  - il servizio e' definito da una *interfaccia del sottosistema*
  - specifica l'interazione e flusso dati da/per il sottosistema, ma non all'interno
  - piccolo e ben definito
  - in fase di implementazione chiamato Application Programmer Interface (API)

scegliere i sottosistemi

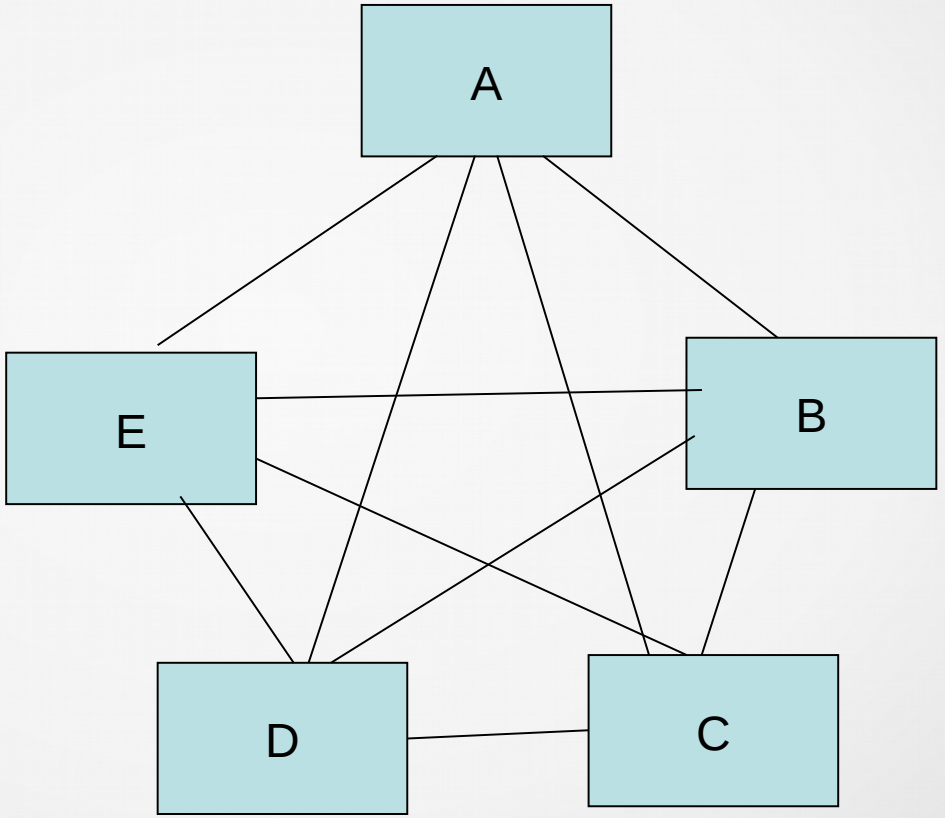
- criterio: gran parte delle operazioni all'interno di **un**, piuttosto che **tra** sottosistemi (coerenza)
  - un sottosistema chiama sempre un altro?
  - due sottosistemi si chiamano a vicenda?
- che tipo di servizio (API) fornisce il sottosistema?
- si riesce a organizzare il sistema a livelli?
- che modello usare per livelli e partizioni?

Stratificazione delle classi

- Un sistema ad oggetti di grandi dimensioni e' una rete complessa complessa di oggetti intercomunicanti
- Occorre un progetto architetturale ben congegnato per gestire la complessita
- Nelle reti, se i nodi comunicano tutti tra essi, il numero di canali e' proporzionale a N**2
- Si puo' ridurre il problema organizzando le classi in gerarchie; solo nodi a livelli adiacenti possono comunicare

Numero di canali

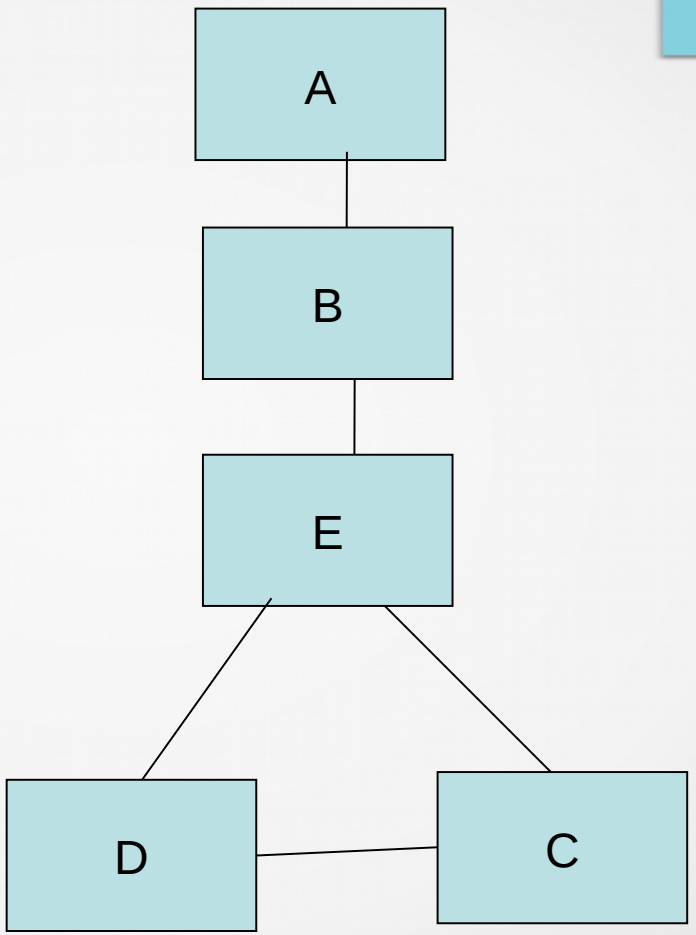
10 archi



Numero di canali

5 archi

4 livelli



indipendenza dei sottosistemi

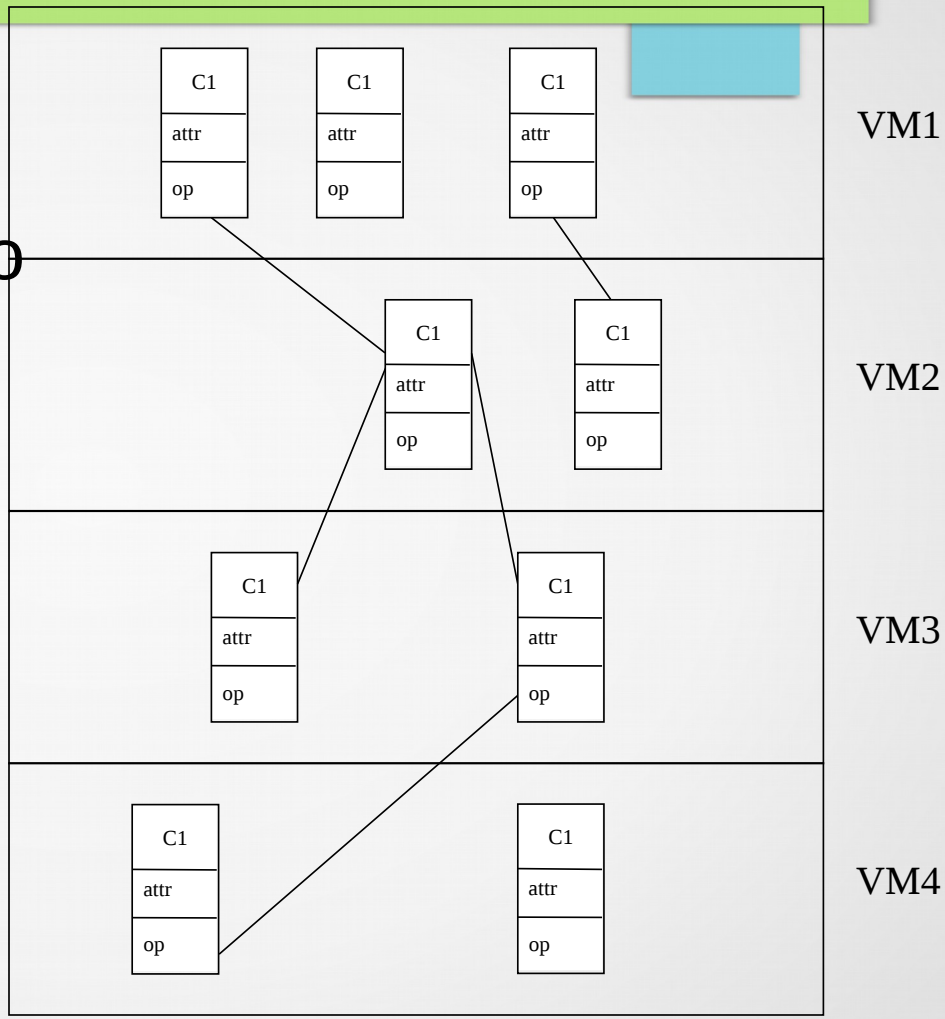
- scopo: ridurre complessita’
  - alta coerenza: le classi svolgono lavori interrelati
  - bassa coerenza: un mix disparato (es utility)
- si vuole evitare ovviamente che modifiche ad un sottosistema si ripercuotano sugli altri
- parallelo tra strade, nazione/regioni/provincie … e sistema/package .. spedire lettera
- Amiat ?

dijkstra (1965)

- un sistema deve essere sviluppato secondo un ordine preciso di macchine virtuali, ognuna delle quali usa servizi di quelle inferiori
- possono implementare due tipi di architetture:
  - chiuse
  - aperte

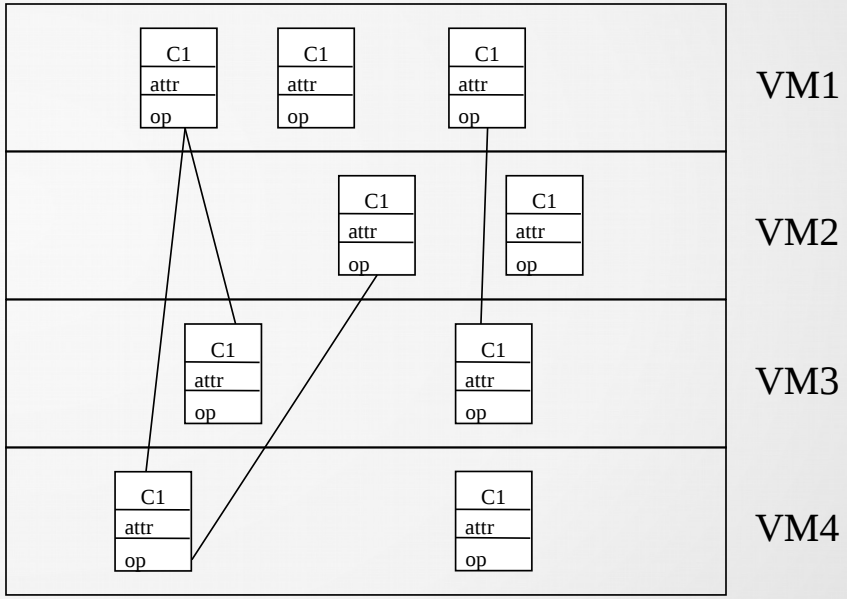
architetture chiuse

- si possono fare chiamate solo al livello inferiore
- obiettivo: alta manutenibilita’



architetture aperte (strati trasparenti)

- si possono fare chiamate a tutti I livelli inferiori
- obiettivo: alta efficienza
- es String vs memcpy



proprieta’ dei sistemi a strati

- sono gerarchici e riducono la complessita’
- le architetture chiuse sono piu’ portabili
- quelle aperte piu’ efficienti
- se un sottosistema e’ uno strato, spesso e’ chiamato macchina virtuale (Virtual Machine o VM)

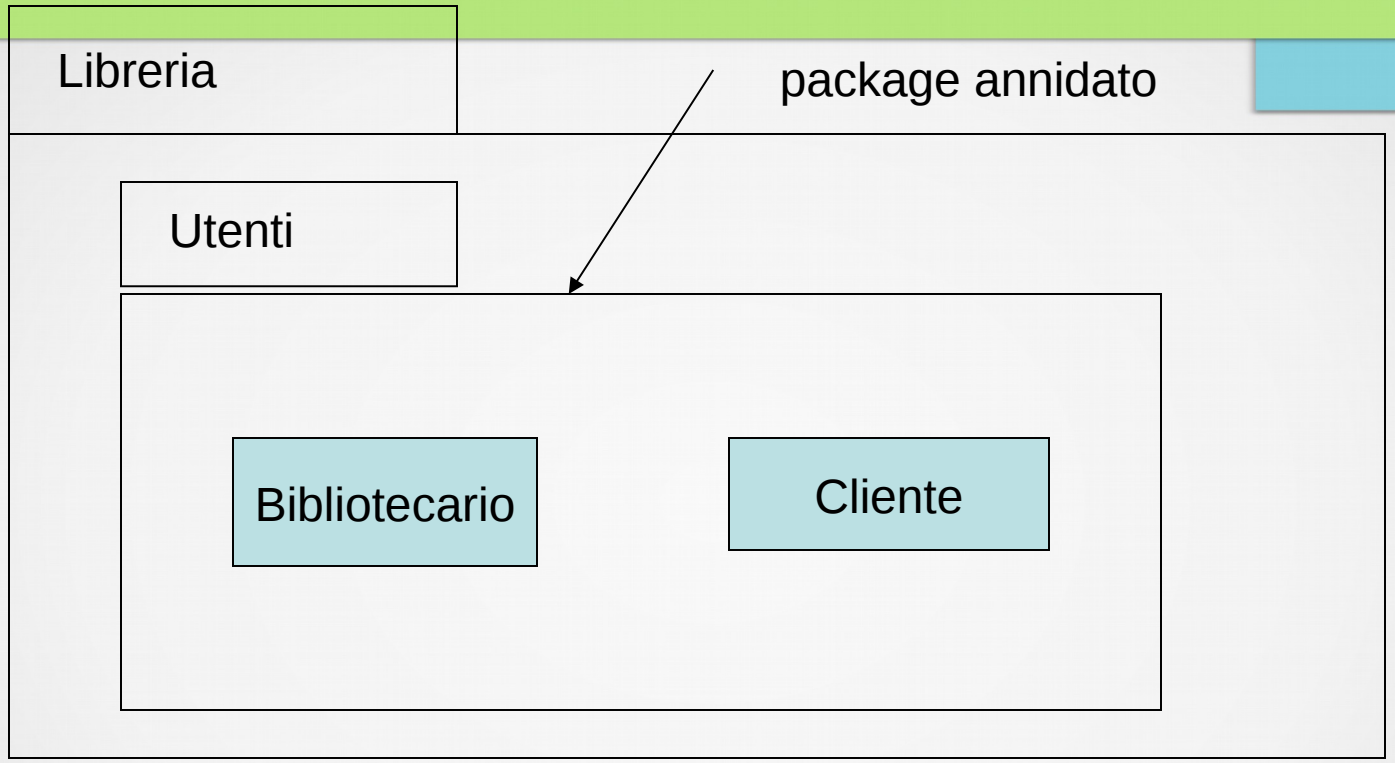
sottosistemi in UML

- package UML : raggruppamento logico di classi, associazioni ,operazioni,eventi, vincoli
  - spunto: classi e oggetti UML
  - serve a dare una organizzazione “logica”, come le cartelle di un file system, ai vari elementi.
  - limita la visibilita’ dei nomi
  - lo stesso nome puo’ apparire in piu’ package
  - i nomi all’interno di un package sono unici
  - fornisce unita’ per lavorare in parallelo e gestire la configurazione

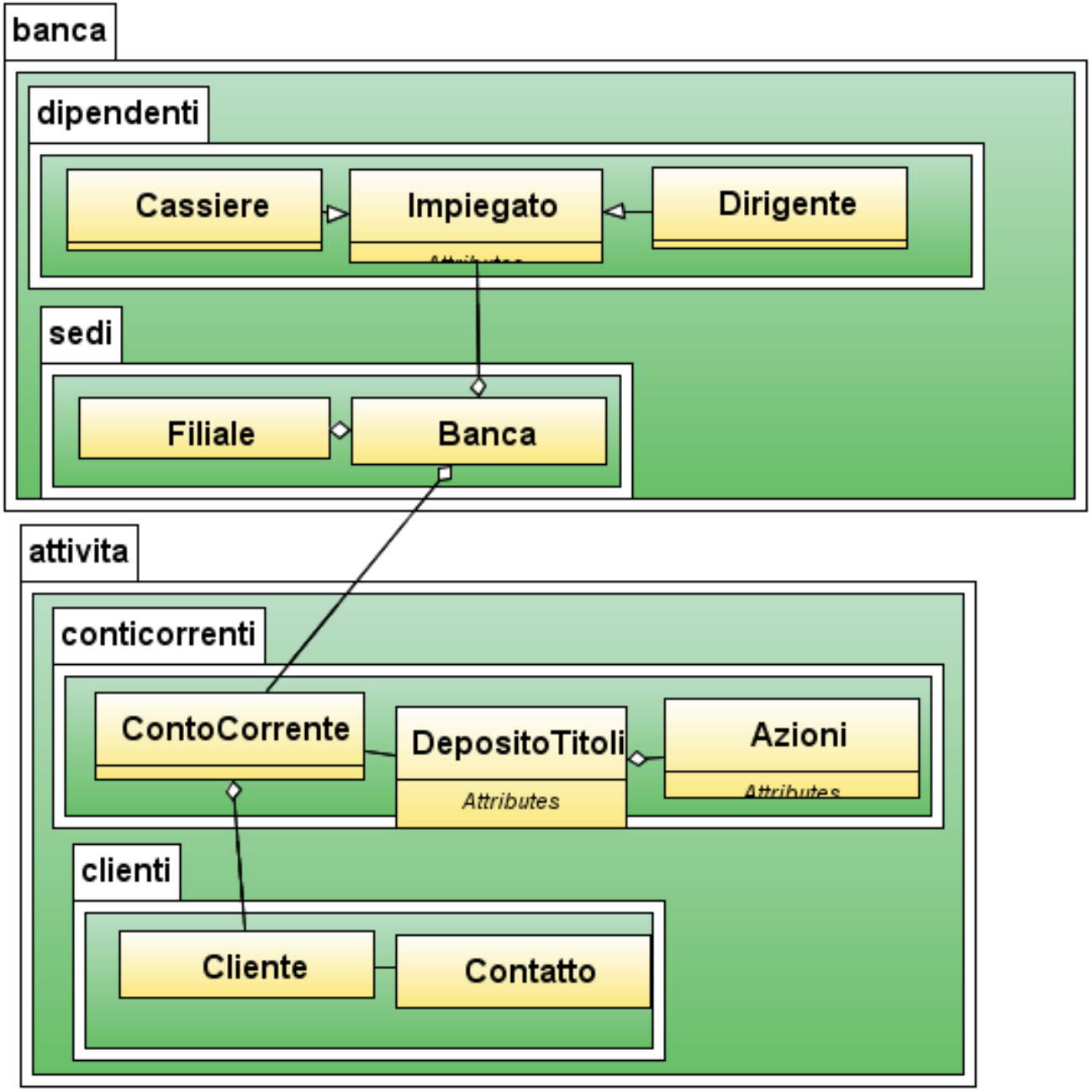
package UML

- La struttura a package/classi e’ simile a quella cartelle/documenti
- Ogni classe ha un solo package proprietario
  - La classe puo’ comunque comunicare ed essere usata da altri package
  - Le regole di visibilita’ classe-package sono simili a quelle applicate ad attributo-classe
- Un package puo’ dipendere da un altro se ad esempio utilizza funzionalita’ di esso (parallelo con le classi)
- le dipendenze non sono transitive (quasi..)

package UML



nome qualificato: Libreria::Utenti::Bibliotecario


banca
dipendenti
Cassiere
Impiegato
Dirigente
sedi
Filiale
Banca
attivita
conticorrenti
ContoCorrente
DepositoTitoli
Attributes
Azioni
clienti
Cliente
Contatto

visibilita’ dei nomi nei package

- i nomi degli elementi di un package possono essere public(+) o private(-)
- un package puo’ importarne un altro, e i nomi pubblici del secondo sono visibili nel primo
- un package annidato puo’ accedere ai nomi del contenitore senza prefisso
  - i nomi fuori dal package devono essere qualificati per poter essere individuati: Se Impiegato avesse un attributo “matricola” il suo nome qualificato sarebbe
    - banca.dipendenti.impiegato.matricola

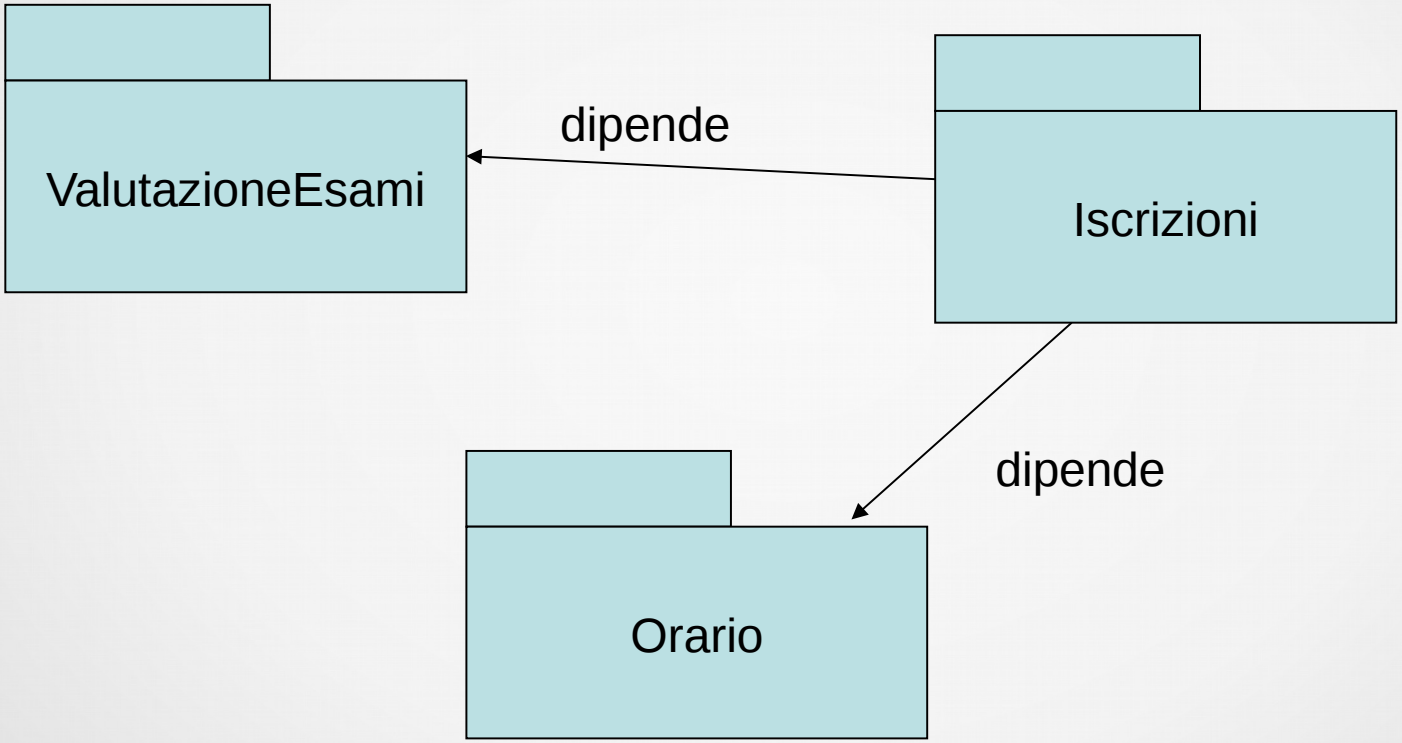
organizzare in package

- in una buona strutturazione, ogni package:
  - ha poca dipendenza dagli altri
  - ha pochi membri pubblici
  - ha molti membri privati
- come vedete siamo sempre alle solite
- importante in progetti grossi
- le dipendenze tra 2 package riassumono le dipendenze tra le loro classi

osservazioni sui package

- tendono ad avere piu’ interfacce
- man mano che un package ha sempre piu’ clienti, la sua interfaccia **dovrebbe** diventare sempre piu’ stabile
- package usati ovunque sono spesso sterotipati **<<global>>**

Packages



Architettura SW

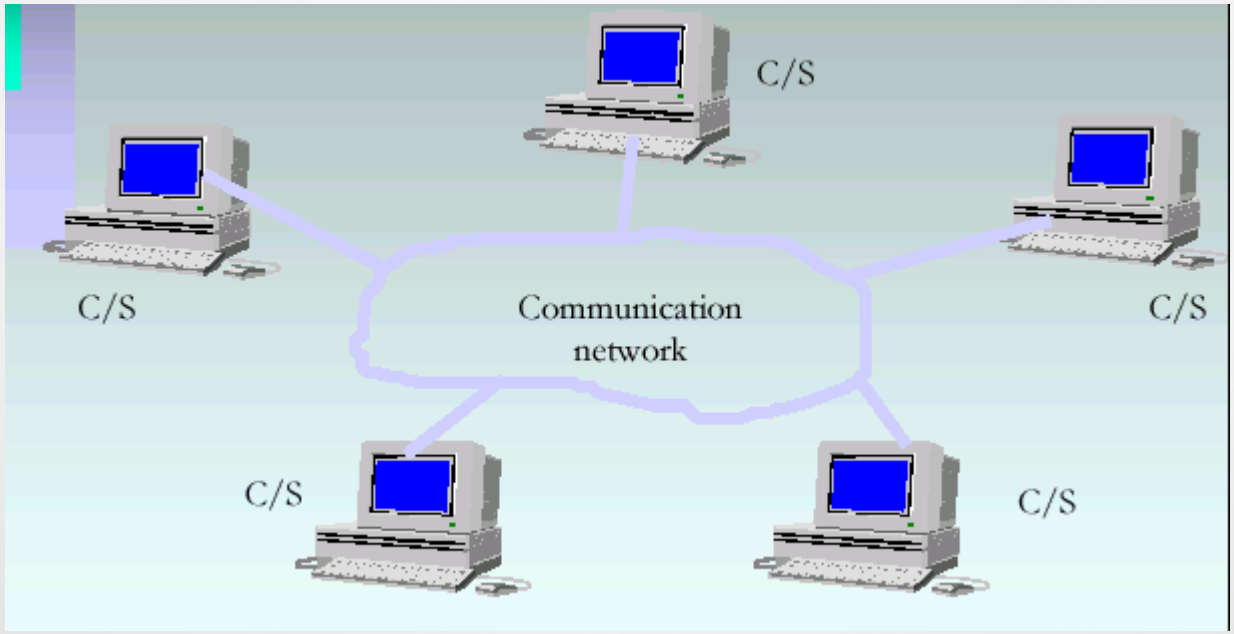
- pattern di utilizzo
- architettura client/server
- peer-to-peer
- repository
- Model/View/Controller
- Filtri e canali

architettura Client-Server

- Uno o piu’ server forniscono servizi a dei sottosistemi detti client
- il client conosce l’interfaccia del server ma non viceversa
- risposta in genere rapida
- gli utenti interagiscono solo col client

Architettura distribuita

- L’architettura client-server puo’ rappresentare un sistema distribuito
- Client e server possono girare sulla stessa macchina o macchine diverse
- Se le richieste a piu’ server dei database sono combinate dai medesimi prima di essere fornite al cliente, si parla di database distribuito



obiettivi per Client-Server

- portabilita’
  - installazione su una varieta’ di sistemi e reti
- trasparenza
  - il server puo’ essere distributo, ma offre una sola interfaccia logica
- prestazioni
  - client lavora sulla interfaccia utente (molti interrupt)
  - server sui task che sfruttano CPU (pochi interrupt)
- scalabilita’:
  - aggiungo server
- flessibilita’
  - fornita dal client per varie interfacce utente
- affidabilita’
  - sopravvivenza a crash di un nodo o di un canale

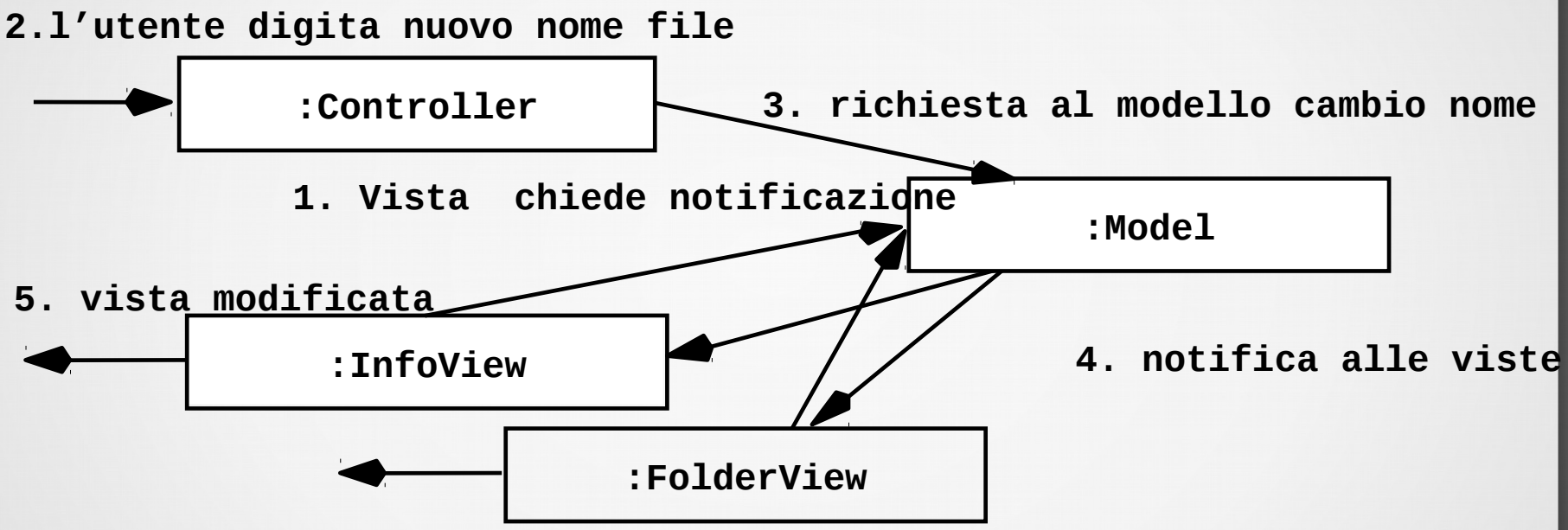
Peer-To-Peer

- molte applicazioni tendono via via a chiedere e fornire servizi
- si passa allora a generalizzare al peer-to-peer, dove sostanzialmente ogni sottosistema puo' essere client e server
- piu' complicata da realizzare, non gerarchica: maggiori interdipendenze tra i sottosistemi

Model-View-Controller (MVC)

- classifica i sottosistemi in 3 categorie:
  - Modello: responsabile per la conoscenza e la organizzazione dei dati del dominio
  - Vista: mostra gli oggetti del dominio all'utente
  - Controllo: gestisce l'interazione con l'utente e lo notifica dei cambiamenti degli oggetti del dominio

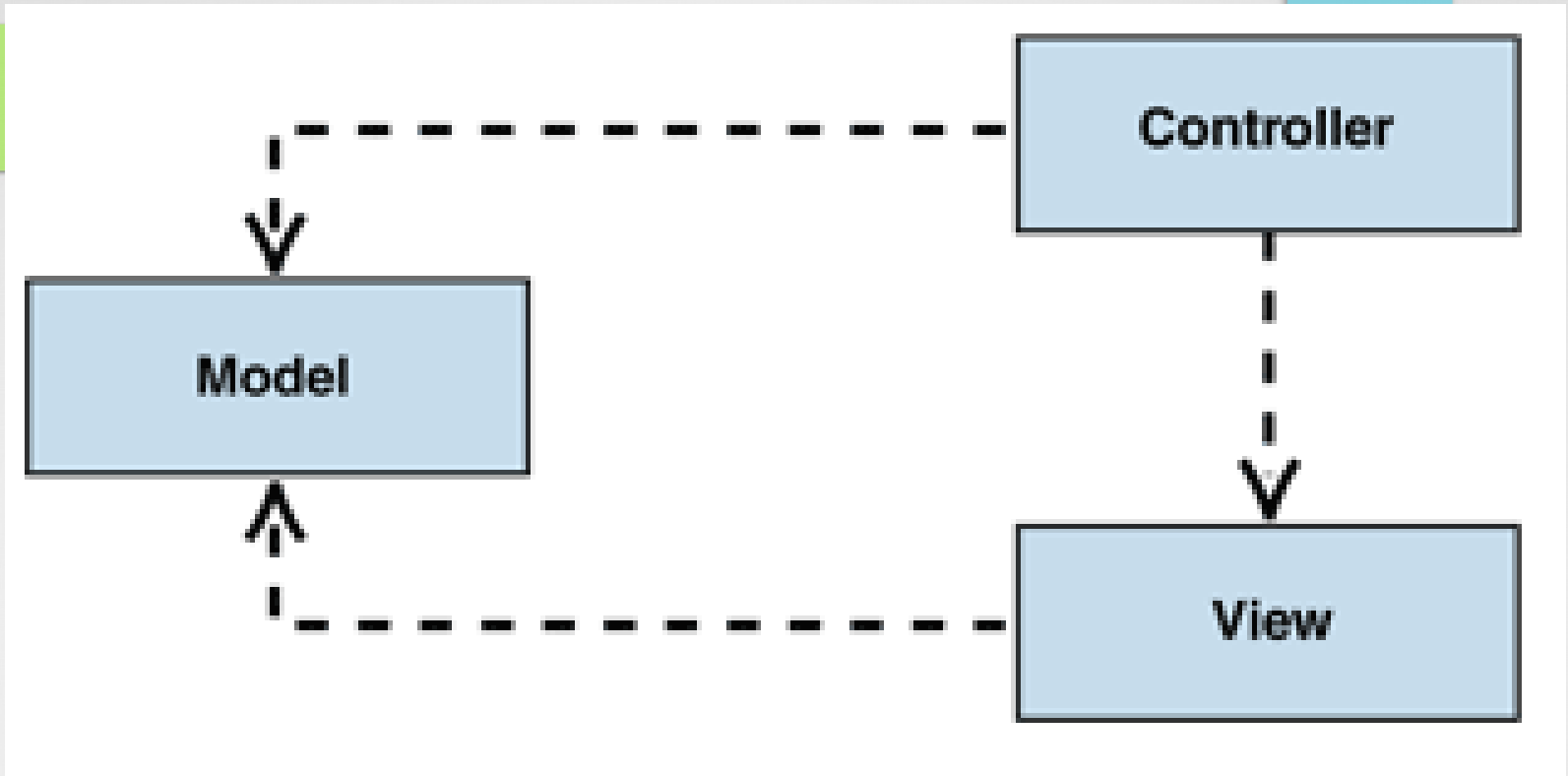
Model-View-Controller (MVC)



Usato in generale per le GUI

Perche' MVC

- La UI cambia piu' frequentemente della logica applicativa, per nuove esigenze, device etc
- Diverse presentazioni degli stessi dati
- Comptetenze diverse per pagine HTML e business logic
- Business logic molto piu' stabile.
- Testing interfaccia separabile dal testing della business logic



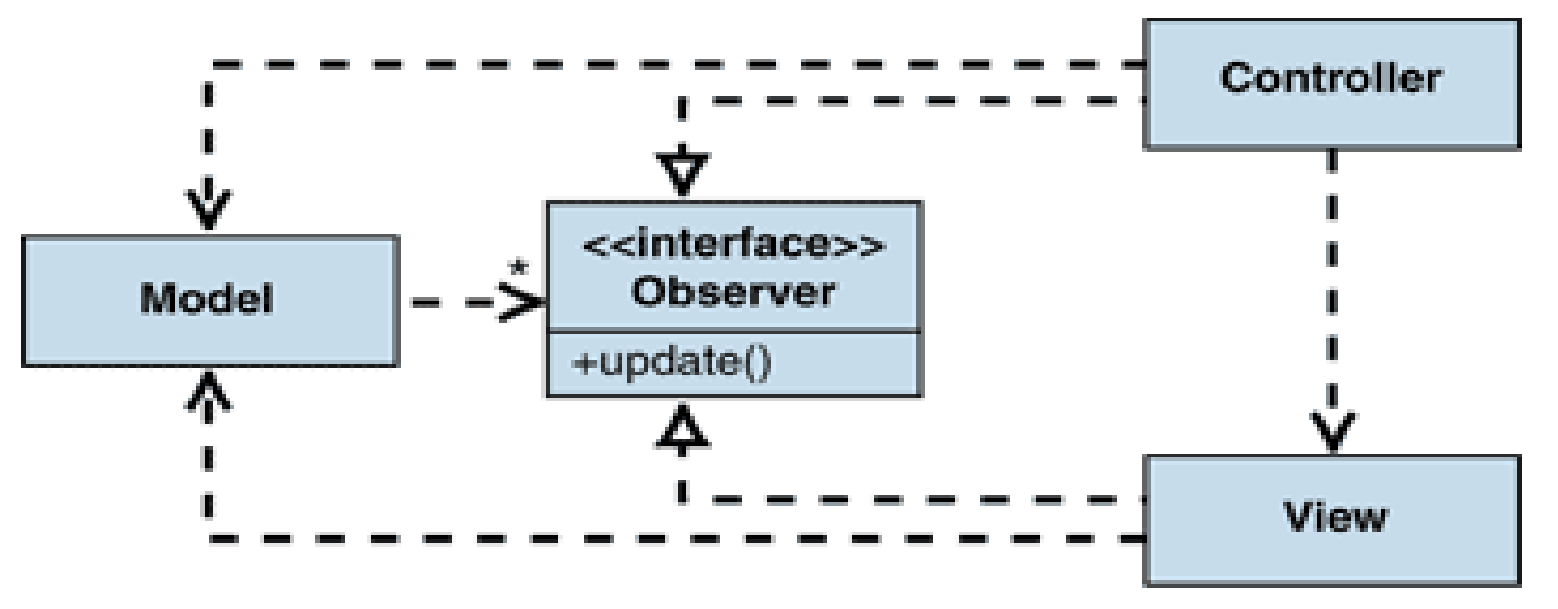
In alcune applicazioni View e Controller sono integrati, in applicazioni Web sono distinte (Browser e Web server)

MVC attivo e passivo

- Model passivo se cambia solo per opera di un Controller (domanda-risposta)
- Model attivo se cambia indipendentemente da azioni del controller (es. quotazioni di borsa)

MVC attivo

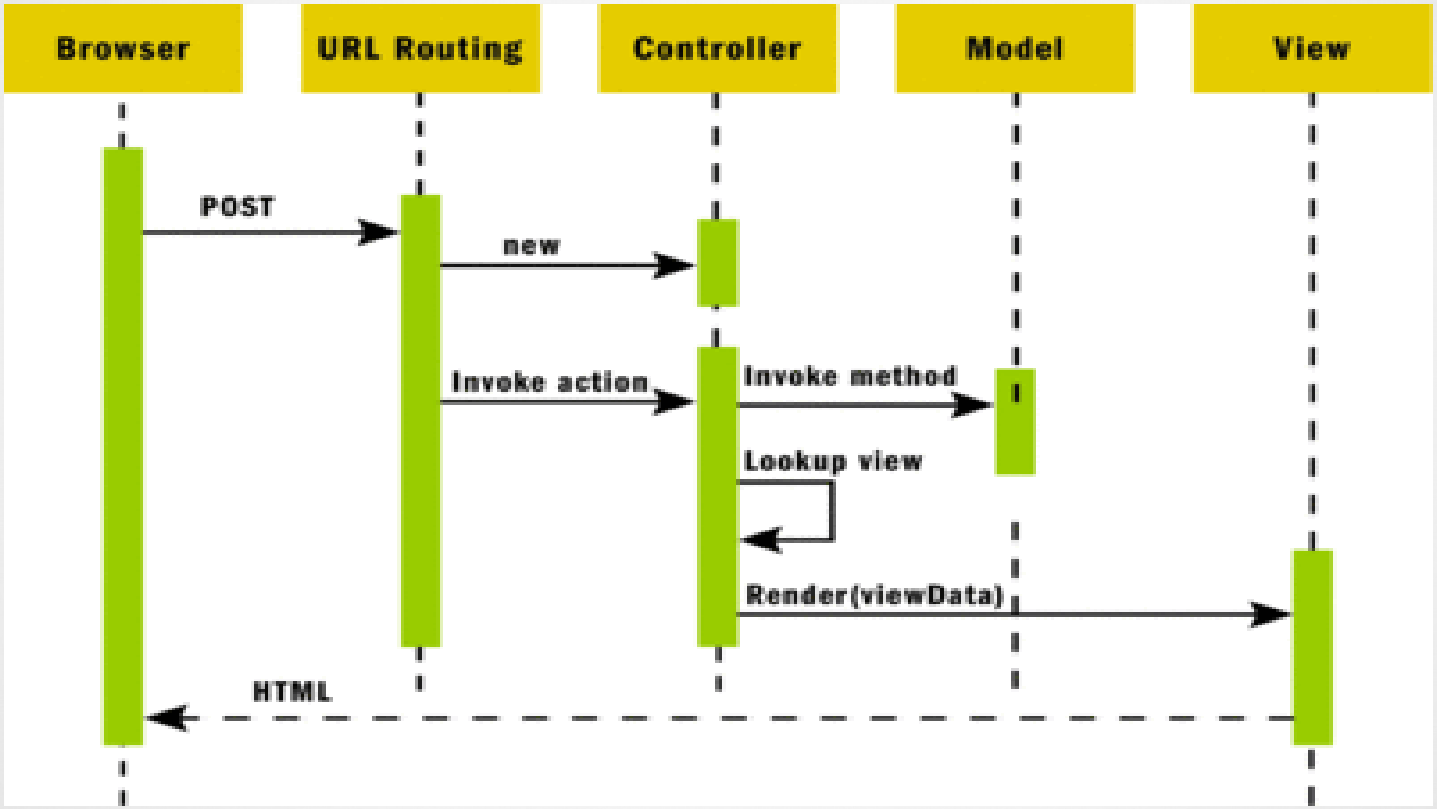
- Il Model deve avvisare le viste di un suo cambiamento, ma non vorreemo che lui le conoscesse, che fare?
- Si usa il pattern Observer (publish & subscribe)

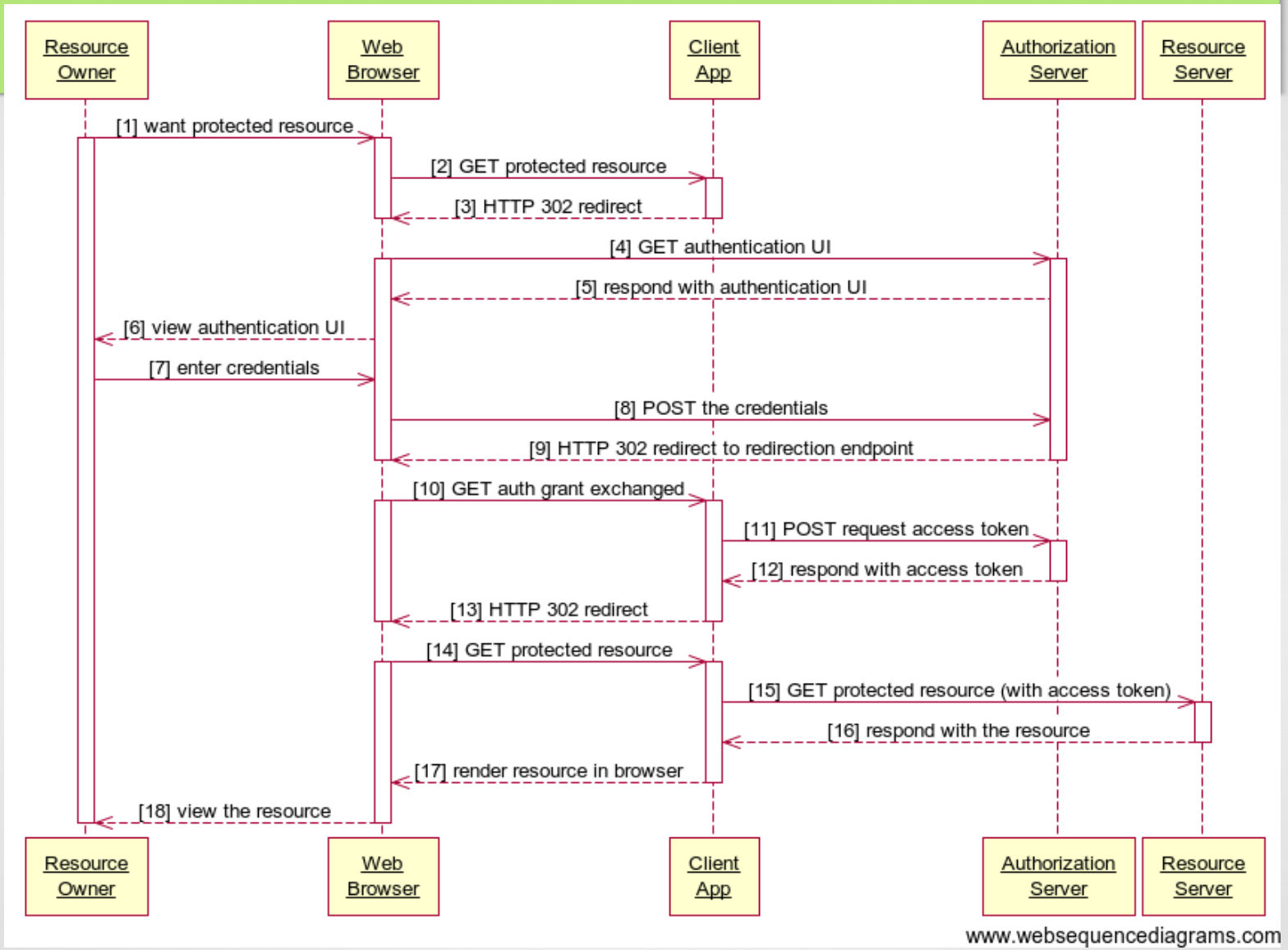


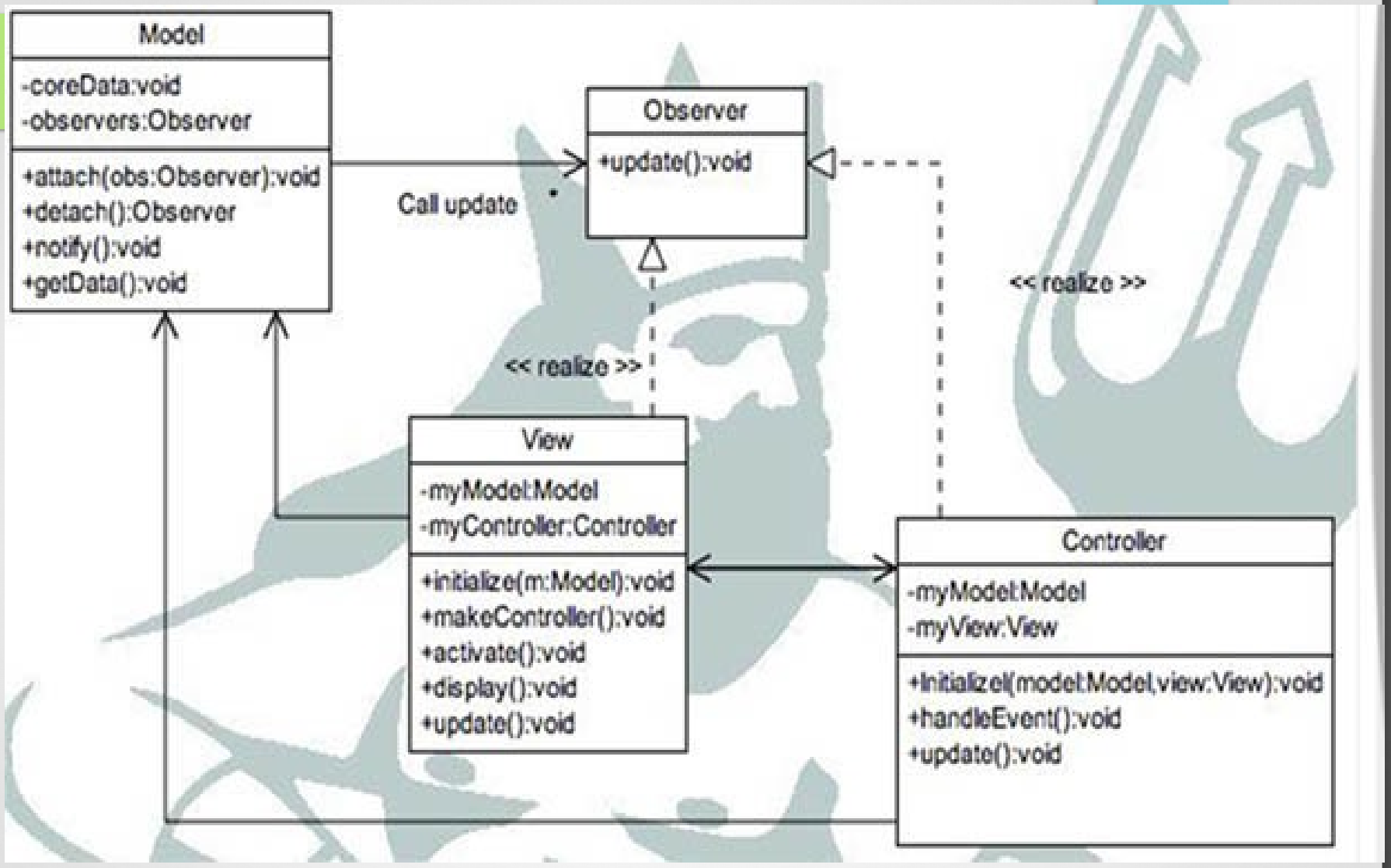
Problemi MVC

- Maggior complessita'
- Logica piu' difficile da seguire
- Valutare la frequenza di aggiornamento delle viste

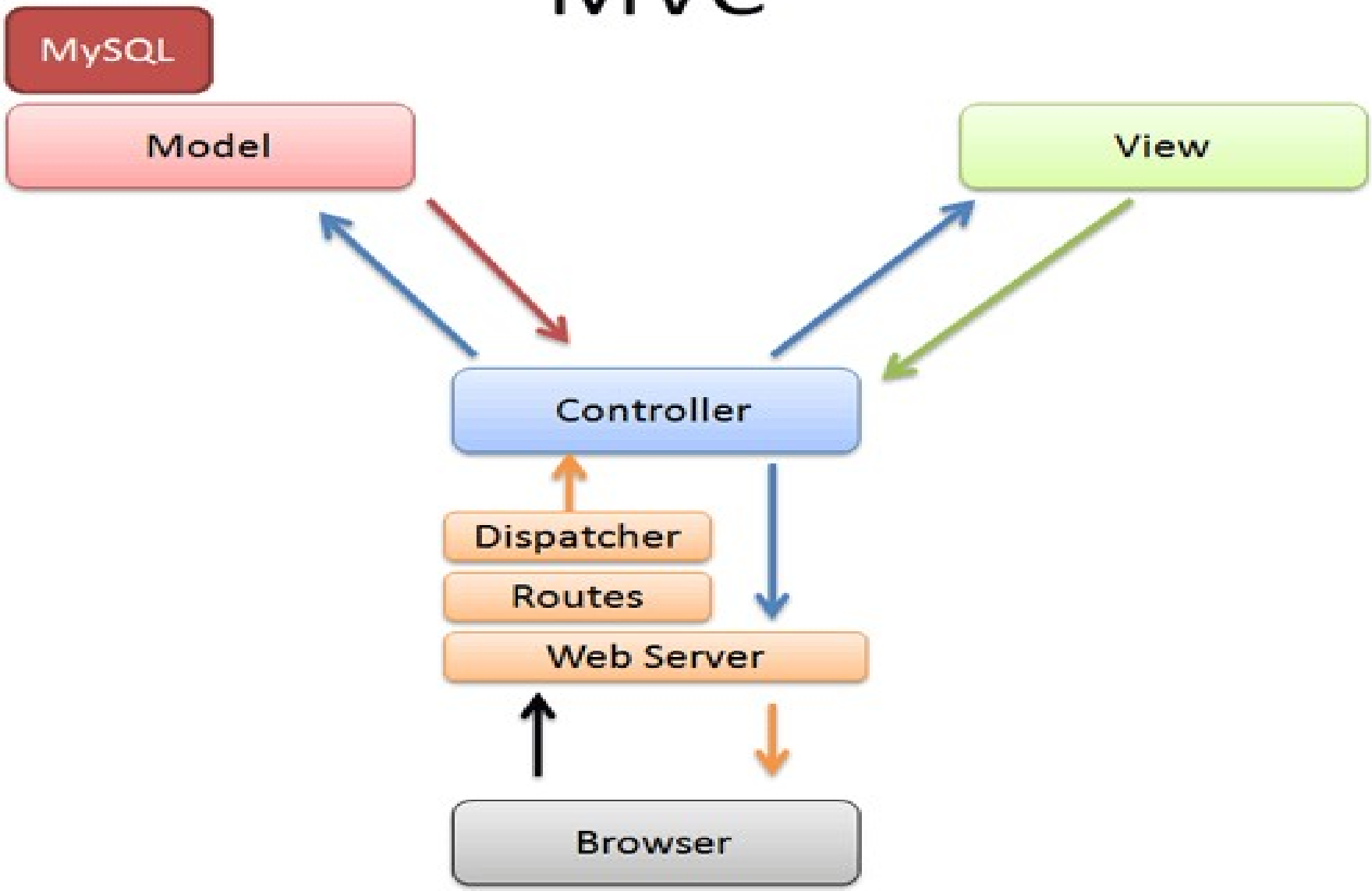
- Esempio MVC


Browser
URL Routing
Controller
Model
View
POST
new
Invoke action
Invoke method
Lookup view
Render(viewData)
HTML


Resource Owner
Web Browser
Client App
Authorization Server
Resource Server
[1] want protected resource
[2] GET protected resource
[3] HTTP 302 redirect
[4] GET authentication UI
[5] respond with authentication UI
[6] view authentication UI
[7] enter credentials
[8] POST the credentials
[9] HTTP 302 redirect to redirection endpoint
[10] GET auth grant exchanged
[11] POST request access token
[12] respond with access token
[13] HTTP 302 redirect
[14] GET protected resource
[15] GET protected resource (with access token)
[16] respond with the resource
[17] render resource in browser
[18] view the resource
Resource Owner
Web Browser
Client App
Authorization Server
Resource Server
www.websequencediagrams.com

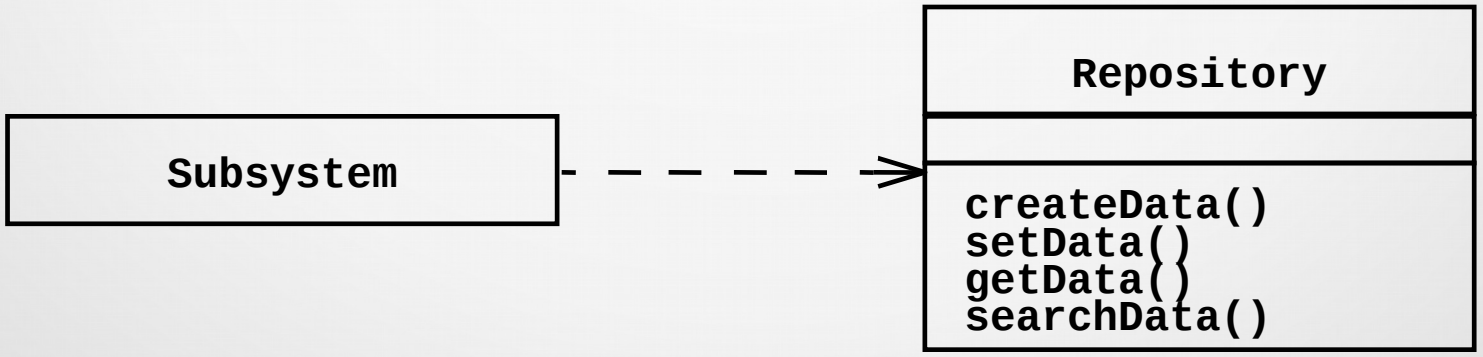

Model
-coreData:void
-observers:Observer
+attach(obs:Observer):void
+detach():Observer
+notify():void
+getData():void
Observer
+update():void
Call update
<< realize >>
<< realize >>
View
-myModel:Model
-myController:Controller
+initialize(m:Model):void
+makeController():void
+activate():void
+display():void
+update():void
Controller
-myModel:Model
-myView:View
+Initialize(model:Model,view:View):void
+handleEvent():void
+update():void

MVC

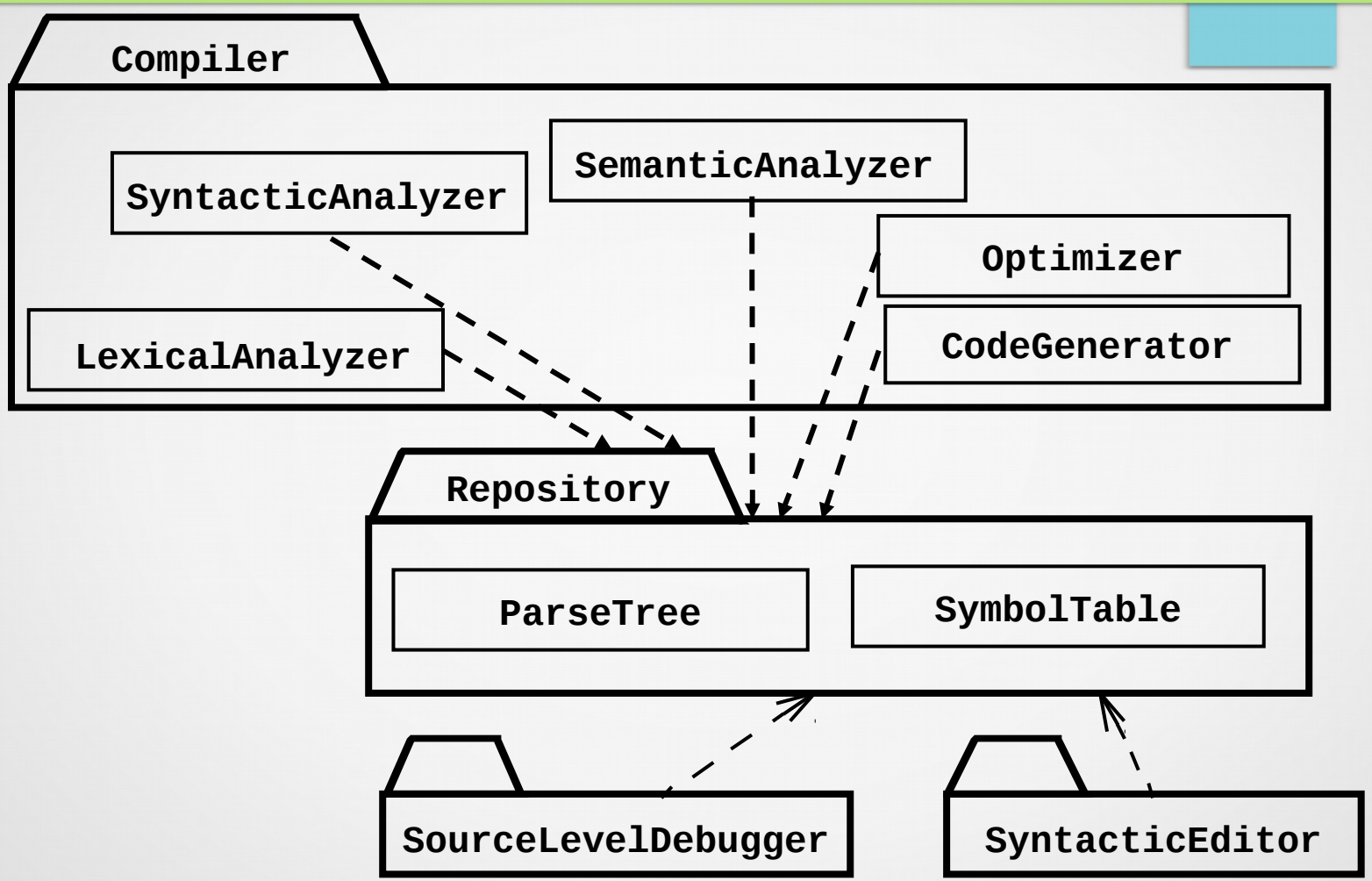

MySQL
Model
View
Controller
Dispatcher
Routes
Web Server
Browser

architettura a Repository

- i sottosistemi accedono e modificano una singola struttura dati
- i sottosistemi interagiscono solo attravero la struttura dati
- il flusso e’ gestito o attraverso triggers della struttura centrale o lock e sincronizzazione dei sottosistemi
- es. repository di documenti (progetto google)



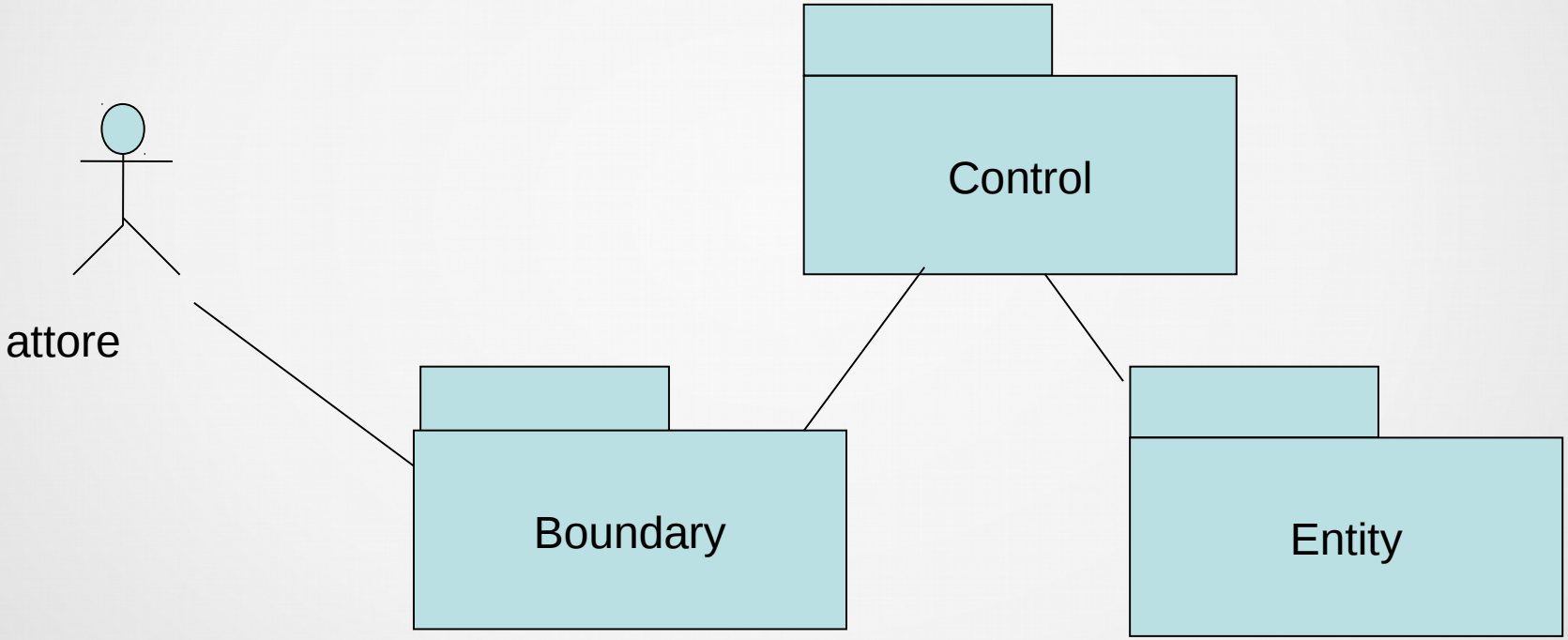
architettura a Repository



Boundary Control Entity (BCE)

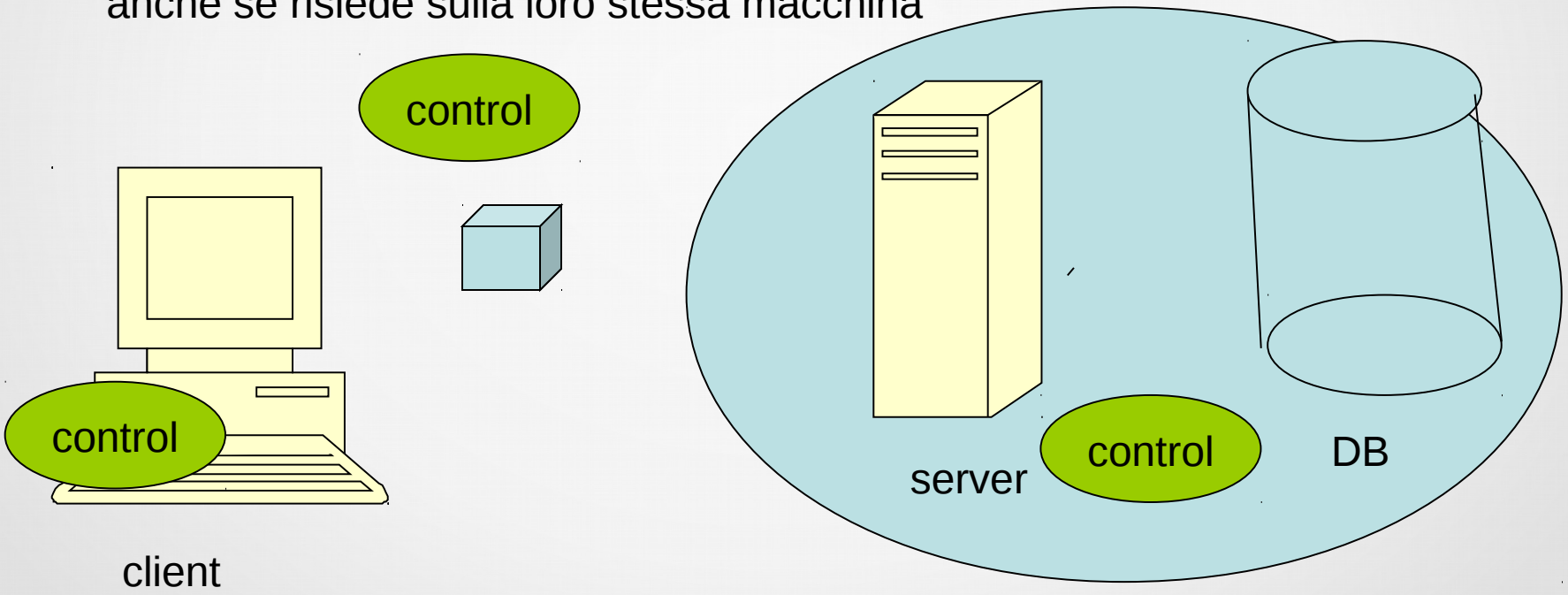
- piu' specifico ai SI di MVC che riguarda di piu' la GUI
- classi di Boundary
  - Oggetti che rappresentano interfacce fra attori ed il sistema
  - Effetti visivi e sonori
  - Durano spesso piu' dell'esecuzione di un use case
- classi di Controllo
  - Catturano gli eventi dell'utente e controllano l'esecuzione della logica dell'applicazione
  - Rappresentano azioni ed attivita di uno use case
  - Terminano spesso assieme allo use case
- classi Entita'
  - Sono oggetti che rappresentano le entita' del database
  - Persistono dopo il termine del programma

BCE Packages



3-tier architecture

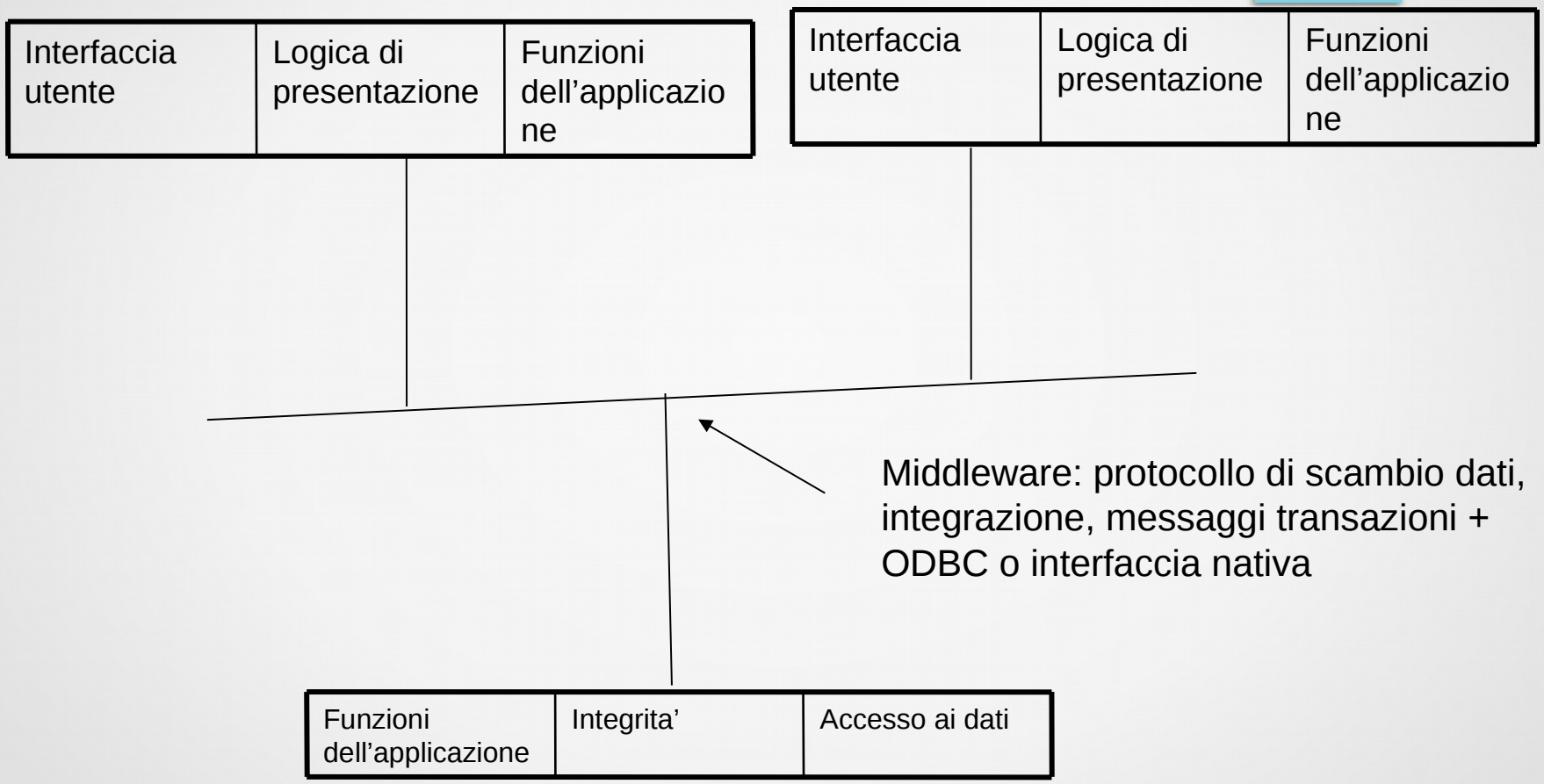
- Come visto nell'approccio BCE, la presentazione (boundary) ovvero l'interfaccia utente sta sul client, mentre le entity sono nel database e gestite dal server.
- La logica (Control) puo' stare sia nel client (cliente pesante) come nel server (cliente leggero solo di interfaccia), come anche in un terzo nodo.
- A livello logico tuttavia il control e' sempre distinto dagli altri componenti anche se risiede sulla loro stessa macchina



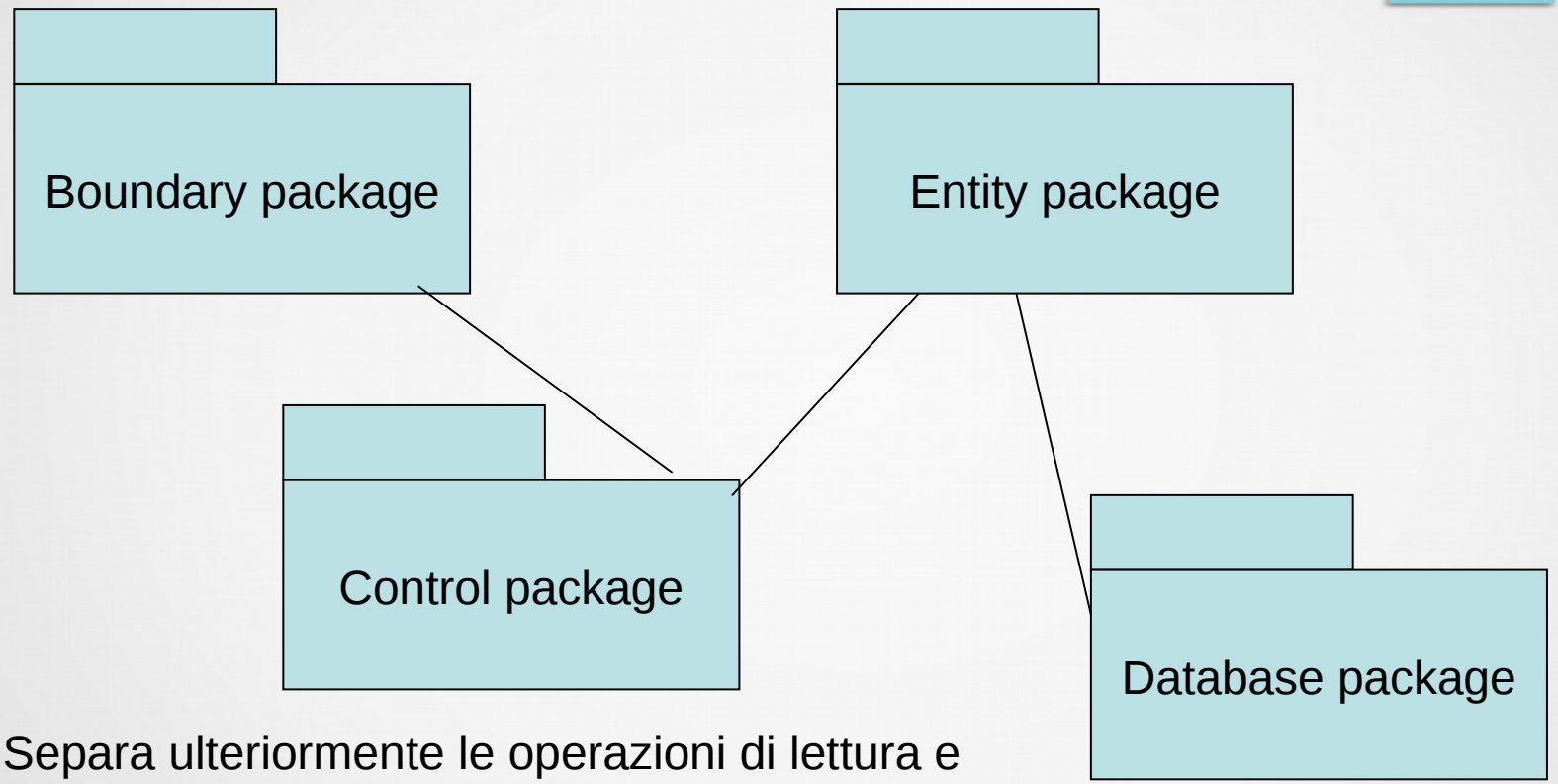
Programmazione del database

- Database attivo
  - Stored procedure (invocabile)
  - Trigger (automatico) verifica integrita’
-

Interazione tra applicazione e database



Approccio BCED



Separa ulteriormente le operazioni di lettura e scrittura dati DB dal resto delle operazioni su entity rendendole piu' indipendenti (puo' essere inefficiente). Puo avere interesse per OODB

Approccio BCDE e connessione

- 4 strati di classi
- Gli oggetti comunicano solo tra livelli adiacenti
- Connessione tra classi dello stesso strato
  - Consente l’evoluzione dei diversi strati in modo indipendente
- Connessione tra classi di strato diverso
  - Da minimizzare e definire tramite interfacce

Coesione e connessione tra le classi

- Coesione
  - Quanto la classe determina il proprio comportamento
  - Misura la sua indipendenza
- Connessione
  - Misura l’interdipendenza tra classi
  - Meglio se limitata
- La coesione e’ antitetica alla interdipendenza

Regole pratiche su coesione e connessione tra le classi

- Due classi sono indipendenti oppure una dipende solo dall’interfaccia pubblica dell’altra
- Gli attributi ed i metodi su loro operanti stanno nella stessa classe
- Ogni classe rappresenta solo un concetto
- Nel sistema l’intelligenza e’ distribuita.

limitare la dipendenza tra classi: la legge di Demeter

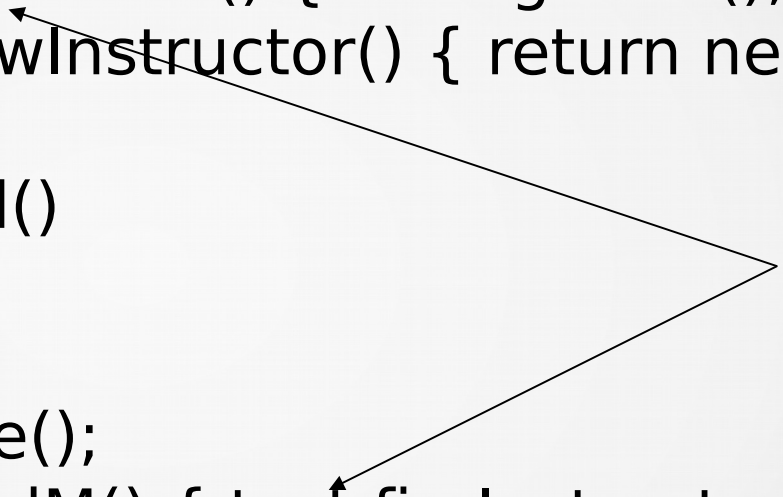
- Un messaggio puo' solo essere inviato a:
  1. L'oggetto mittente
  2. Un oggetto argomento del metodo
  3. Un oggetto attributo del mittente
  4. Un oggetto creato dal metodo
  5. Un oggetto che sia variabile globale
- Versione ristretta
  - Punto 3 limitato ad attributi non ereditati

La legge di Demeter: non fare

```
class Inside {
   Instructor boring = new Instructor();
   public Instructor getInstructor() { return boring; }
   public Instructor getNewInstructor() { return new
Instructor(); }
   // ..... dichiarazione dei metodi hired() e fired()
}
class C {
   Inside test = new Inside();
   public void badM() { test.getInstructor().fired(); }
   public void goodM() { test.getNewInstructor().hired(); }
}
```

La legge di Demeter: fare

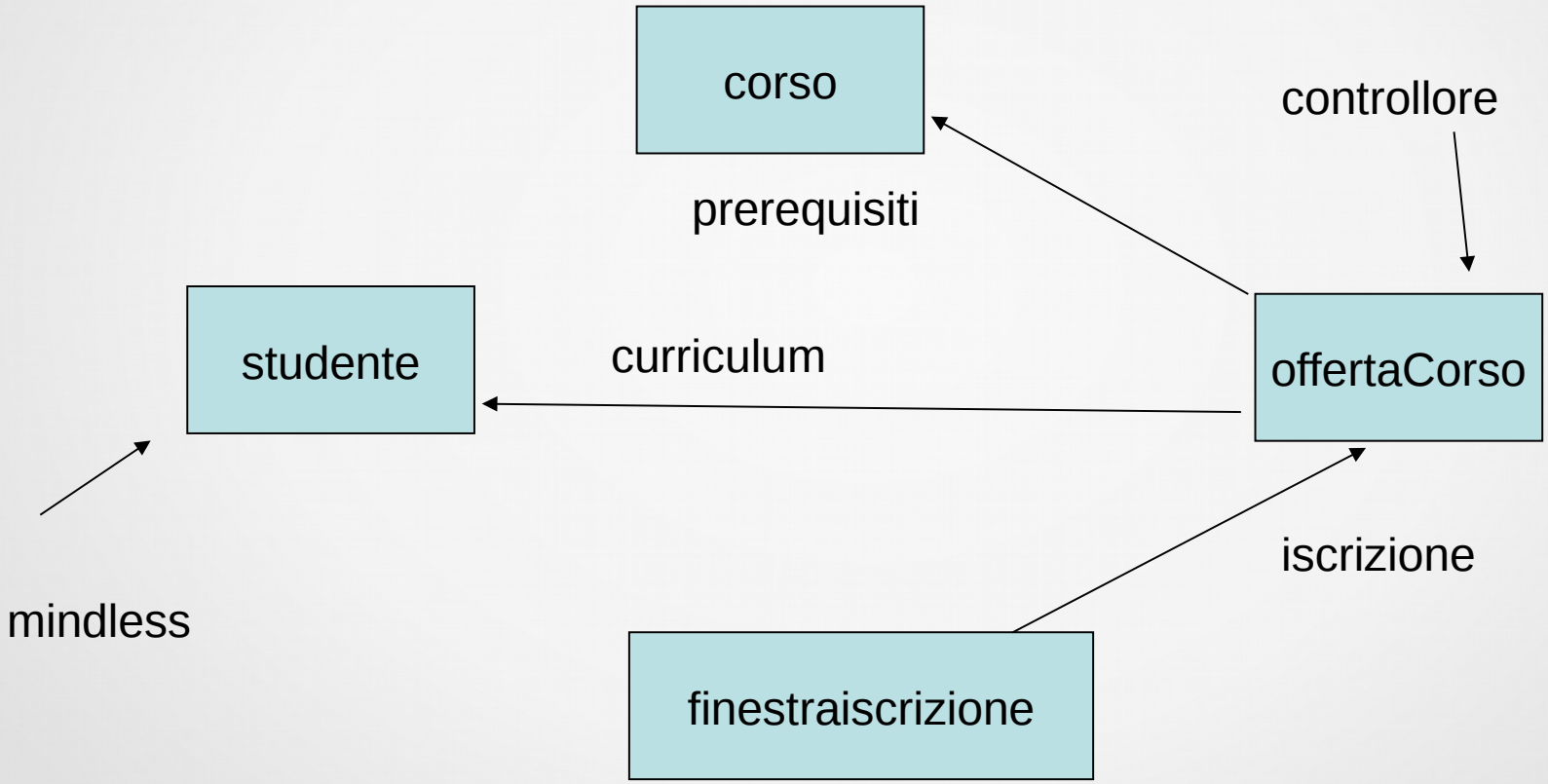
```
class Inside {
   Instructor boring = new Instructor();
   public Instructor fireInstructor() { boring.fired(); }
   public Instructor getNewInstructor() { return new
Instructor(); }
   // dichiarazione di hired()
 }
class C {
   Inside test = new Inside();
   public void reformedBadM() { test.fireInstructor(); }
   public void goodM() { test.getNewInstructor().hired(); }
}
```



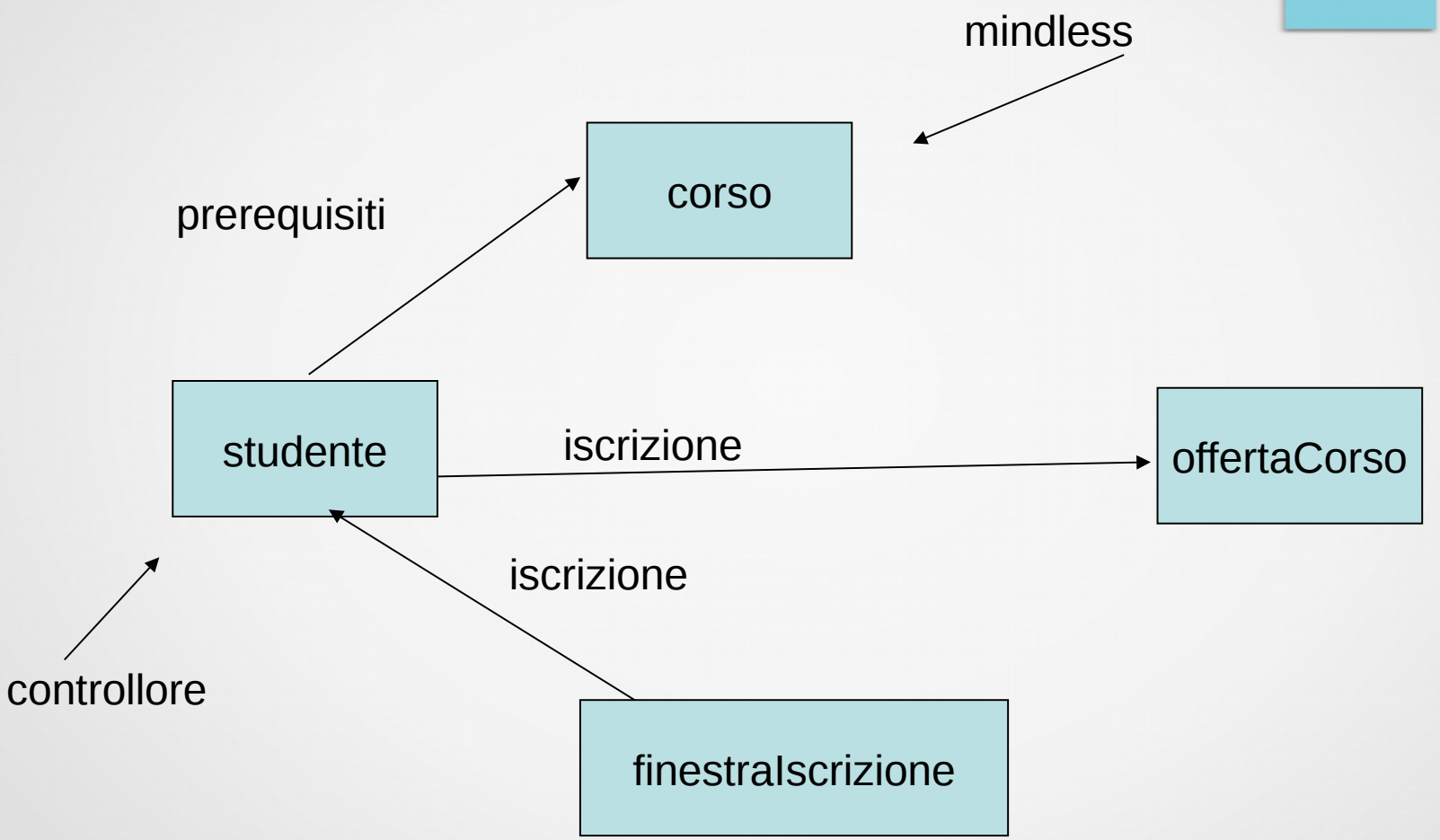
Metodi set/get

- Metodi che accedono agli attributi
  - Gli attributi sono solo acceduti tramite metodi
- La classe “mindless”
  - Ha troppi metodi di accesso
  - Consente alle altre classi di modificare liberamente il suo stato
  - OK se la classe e’ un semplice contenitore
- Se le classi sono relativamente alla pari e’ abbastanza arbitrario decidere quale controllera’ e quale potrebbe essere mindless

Metodi di accesso - 1

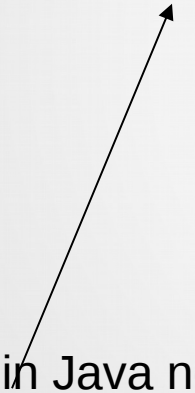


Metodi di accesso - 2



Metodi di accesso

- Supponiamo di aver due classi: Integer e Real. Se vogliamo convertirli possiamo fare in almeno 3 modi:
  - Integer.convertToReal()
  - Real.convertToReal(int)
  - Converter.convertToReal(int)

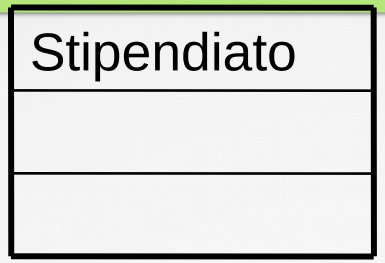


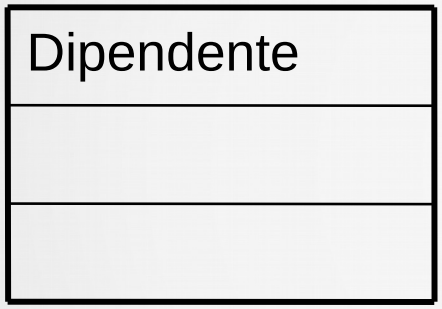
in Java non si puo' fare: int.convertToReal() e' un tipo elementare e non un oggetto esiste tuttavia la classe Integer che e' rappresenta un intero come un Oggetto

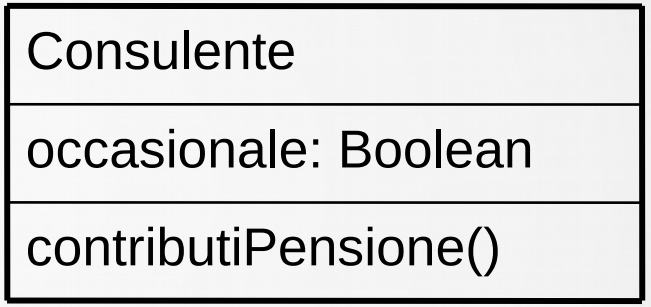
problema della Coesione nelle istanze miste

- Originata dalla mancanza di classificazione multipla
- La classe ha delle proprieta’ non definite per tutte le sue istanze
- Alcuni metodi non possono essere applicati ad alcune istanze della classe
- affrontato con le interfacce e l’aggregazione

Coesione nelle istanze miste

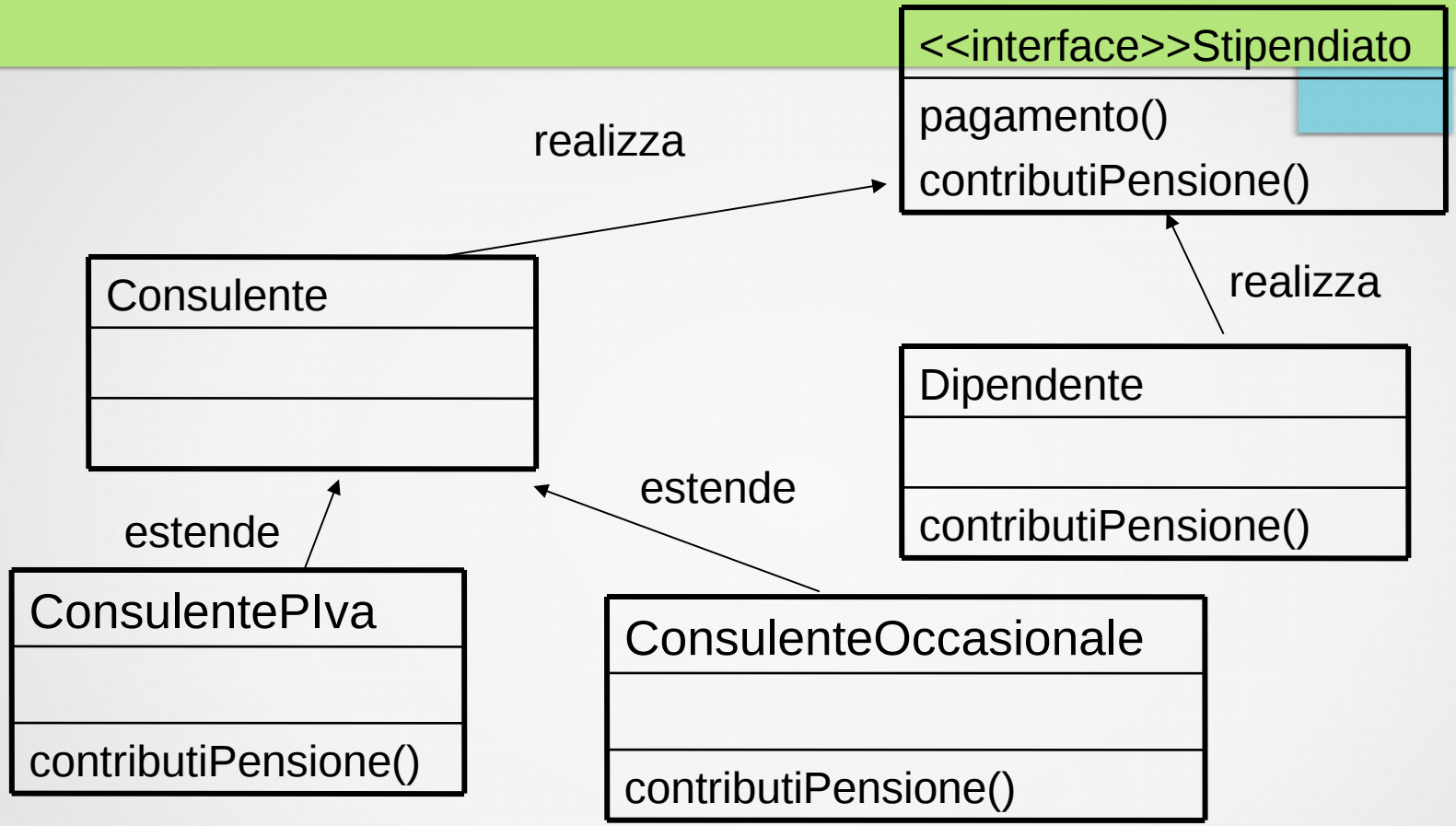






Se ad esempio abbiamo un consulente del tipo a prestazione occasionale, non deve pagare contributi pensione; potremmo dire che il metodo restituisce ‘0’ anche per lui

Coesione nelle istanze miste



Aggiunta di due classi e rimozione del flag “occasionale”; spesso si usano flag multiple per evitare di modellare troppe classi; i valori dei flag modificano il comportamento delle operazioni della classe

Software di sistema

- Connessione al DB
  - Interfaccia nativa di uno specifico ambiente
  - Interfaccia standard (JDBC ODBC)
- Lato server
  - Database relazionale
  - Database relazionale-oggetti
  - Database a oggetti

Modalita’ di riuso: molti sottosistemi si ripetono

- Granularita’
  - Classe
  - Componente
  - Soluzione
- Strategie
  - Toolkit (librerie di software)
    - Classi elementari (es Java Foundation Classes)
    - Architetture (es java-swing)
  - Framework (strutture configurabili) es.SAP
  - Schemi di analisi e progetto funzionanti (design pattern)

Framework

- Nel framework, esiste gia' uno scheletro dell'applicazione
- Sono disponibili molte classi riutilizzabili
- Basta riempire i vuoti
- Non e' detto sia Object Oriented
- Si scrive poco codice ma e' piu' difficile (operazioni complesse)
  - Occorre tenere conto dello stato del sistema

Obiettivi in fase Design

- ridurre il divario tra requisiti e macchina
- valutazione dei compromessi
- prioritizzazione degli obiettivi
- organizzazione in sottosistemi

Modello dominante

- ***Modello ad oggetti*:** il sistema ha strutture dati non banali: es database: molte strutture per accedere ai dati in modo efficiente, grosso costo di verifica della correttezza dati
- ***modello dinamico:*** ha molti tipi di eventi: input, tastiera, errori ecc. es GUI,OS
- ***modello funzionale*:** computazionalmente complesso. esempio compilatore:dinamica nulla: un input, pochi output

nel frattempo I progettisti fanno analisi collaborativa

- I gruppi che analizzano I vari elementi collaborano alla loro integrazione e verificano la coerenza delle scelte
  - **tutte le classi sono menzionate nel dizionario?**
  - **I nomi sono scelti con cura e riflettono la natura dell'elemento?**
  - **controllo della corrispondenza tra i vari elementi**
    - **classi e metodi mancanti**
    - **associazioni senza corrispondenze**

consistenza, completezza, ambiguita’

- consistenza
  - ogni corrispondenza tra le classi e’ verificata
- completezza:
  - classi doppiamente definite
  - riferimenti mancanti
  - classi usate ma non definite
- ambiguita’
  - errori nei nomi
  - classi con nomi uguali e differenti significati

sommario

- Le tipiche applicazioni dei sistemi informatici sono basate sulla architettura client-server
- I sistemi 3-tier estendono la base client –server
- La stratificazione BCE estesa a BCED con package di interfaccia al database
- Il riutilizzo avviene a livello di libreria , di scheletro applicativo, di modello di soluzione
- Puo’ essere difficile allineare l’architettura interna con le esigenze di riutilizzare componenti preesistenti, sia HW che SW
- Il progetto dettagliato si basa sui diagrammi di sequenza/collaborazione
- Gli aspetti strutturali sono modellati nei diagrammi di classe, quelli comportamentali nei diagrammi di sequenza/collaborazione