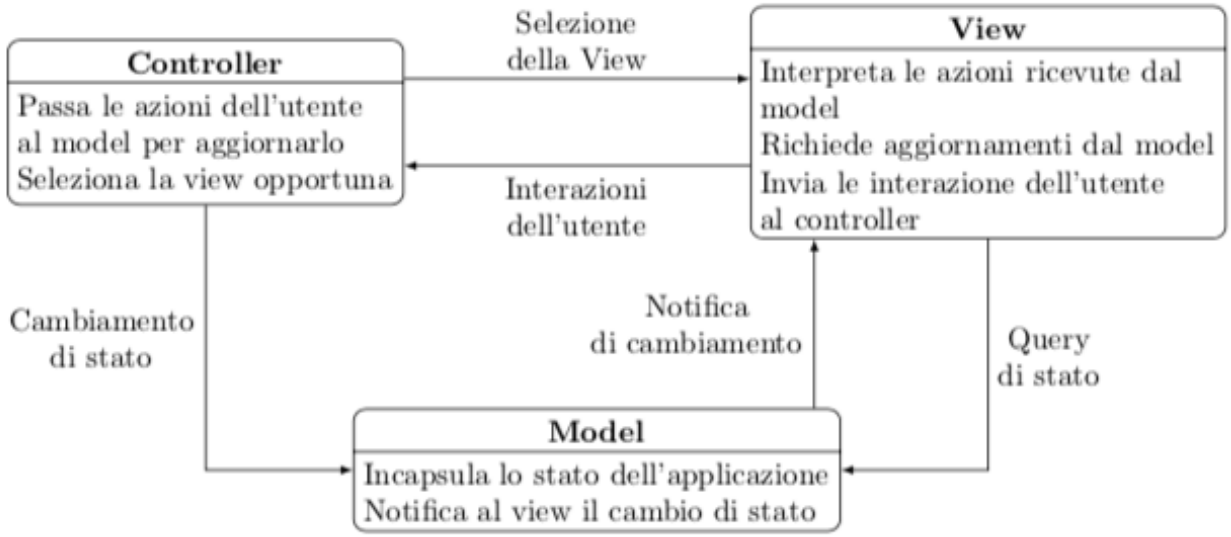
Pattern Architetturali

Sono un mezzo per rappresentare, condividere e riutilizzare le conoscenze software, fornendo linee guida e strutture per organizzare e progettare sistemi complessi.

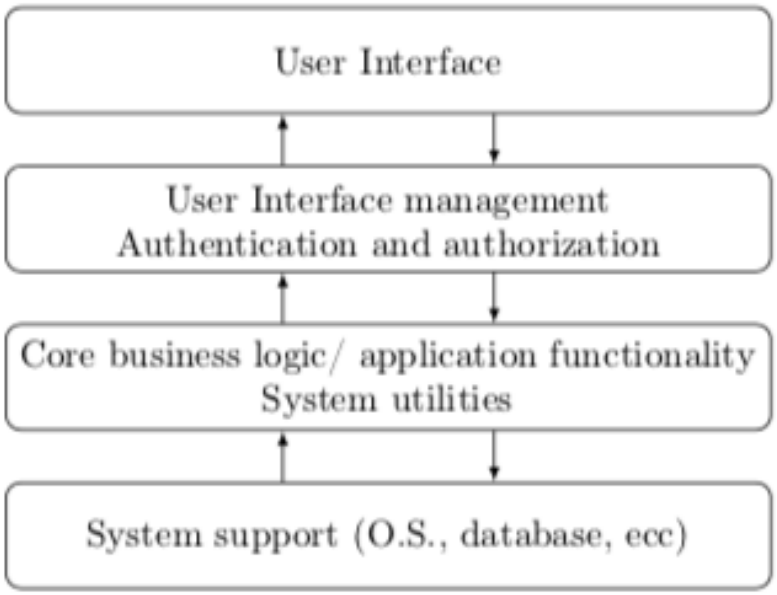
Model View Controller (MVC)

Si compone di 3 componenti logiche che interagiscono fra loro:
-il Model si occupa delle operazioni e della gestione dei dati
-il View definisce l'interfaccia
-il Controller gestisce l'interazione dell'utente con questa interfaccia, e coordina le interazioni a View e Model



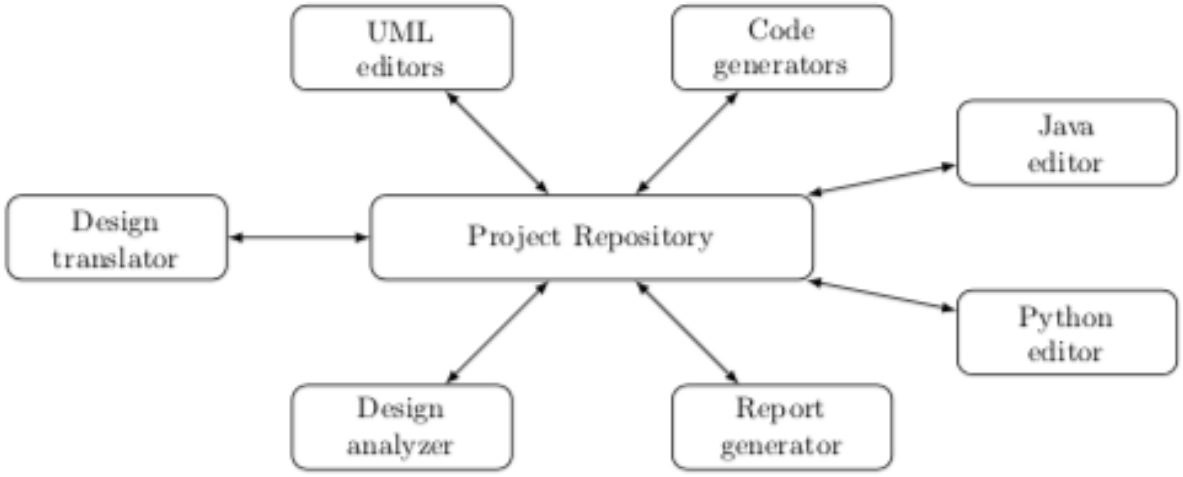
Architettura a strati (Layered)

L'architettura a strati organizza il sistema in un insieme di strati, ciascuno dei quali fornisce e utilizza funzionalità offerte dagli strati sottostanti, promuovendo la modularità e la separazione delle responsabilità.



Architettura repository

Nel caso in cui 2 sotto-sistemi debbano scambiarsi i dati, solitamente ci sono 2 modalità:
1) i dati vengono mantenuti in un database centrale e possono essere acceduti da tutti i sotto-sistemi
2) ogni sotto-sistema mantiene un database separato e consente lo scambio esplicito di dati tra gli altri database degli altri sotto-sistemi, particolarmente efficiente in presenza di grandi volumi di dati.

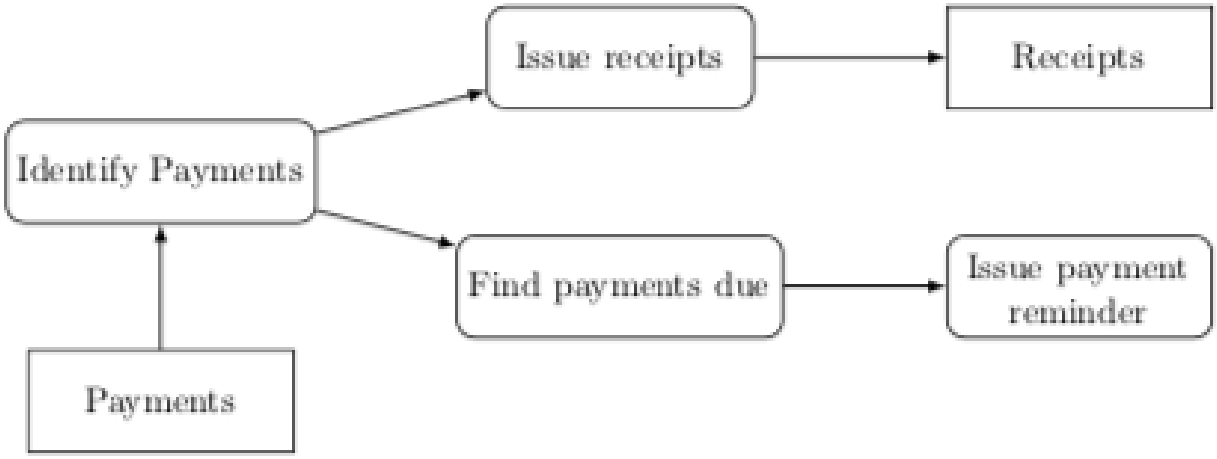


Architettura Client Server

In questa architettura, le funzionalità del sistema sono organizzate in servizi forniti da server distinti, con i client che accedono a tali servizi attraverso le interfacce offerte dai server, promuovendo la scalabilità e la distribuzione delle risorse.

Architettura Pipe and Filter

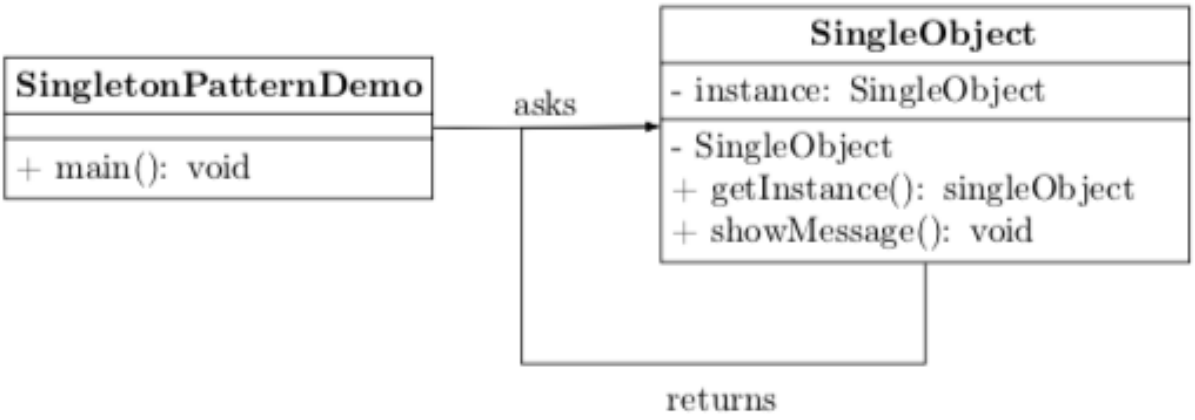
Ogni componente (Filter) genera in output un tipo di trasformazione dei dati. L'output di un componente è l'input del componente successivo (come in una Pipe), facilitando la modularità e la riusabilità delle componenti.



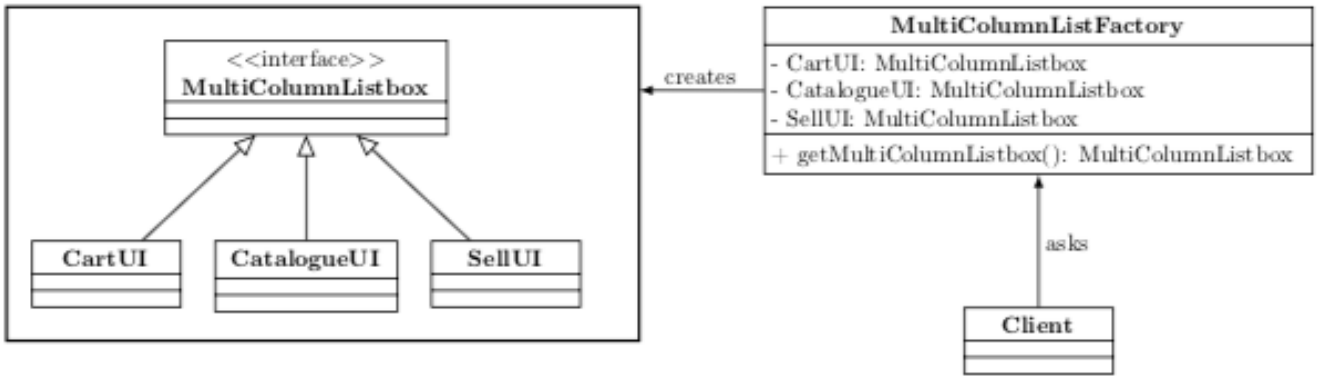
Pattern di progettazione
Costituiscono schemi di base per progettare i software.

Pattern CREAZIONALI

Singleton
Assicura che una classe abbia una sola istanza per tutto il ciclo di vita del software e un unico punto di accesso globale. Le classi Singleton hanno costruttori privati per evitare di istanziare più oggetti della stessa classe; un metodo statico restituisce un riferimento all'unico oggetto di tale classe.



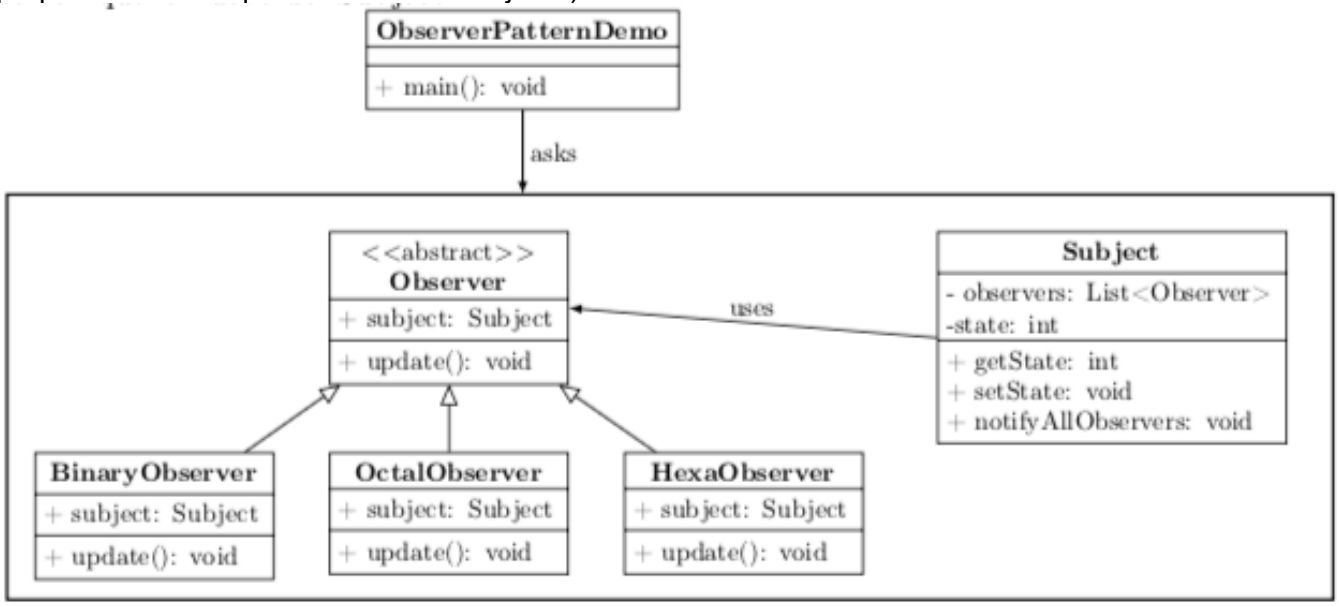
Factory
Il pattern Factory consente di creare oggetti senza esporre la logica della creazione al client: per riferirsi a tall oggetto si usa un'interfaccia comune.



Abstract Factory
In esso, una super-factory crea altre factory; consentendo la creazione di famiglie di oggetti senza specificarne le classi effettive

Pattern COMPORTAMENTALI

Observer
Definisce una dipendenza uno-a-molti tale che quando un oggetto (Subject) cambia stato, tutti gli altri oggetti (Observers) vengono automaticamente aggiornati di conseguenza.(In risposta alla notifica di modifica, gli Observers chiedono al Subject le informazioni necessarie a sincronizzare il proprio strato con quello nuovo del Subject.)



Iterator
Il pattern Iterator permette di scorrere sequenzialmente gli elementi di una collezione senza conoscere la sua rappresentazione sottostante, fornendo un'interfaccia uniforme per accedere agli elementi della collezione.

Template
Il pattern Template definisce la struttura di un algoritmo in una superclasse, delegando l'implementazione di alcuni passi dell'algoritmo alle sottoclassi, promuovendo il riutilizzo del codice e la flessibilità nell'estensione del comportamento

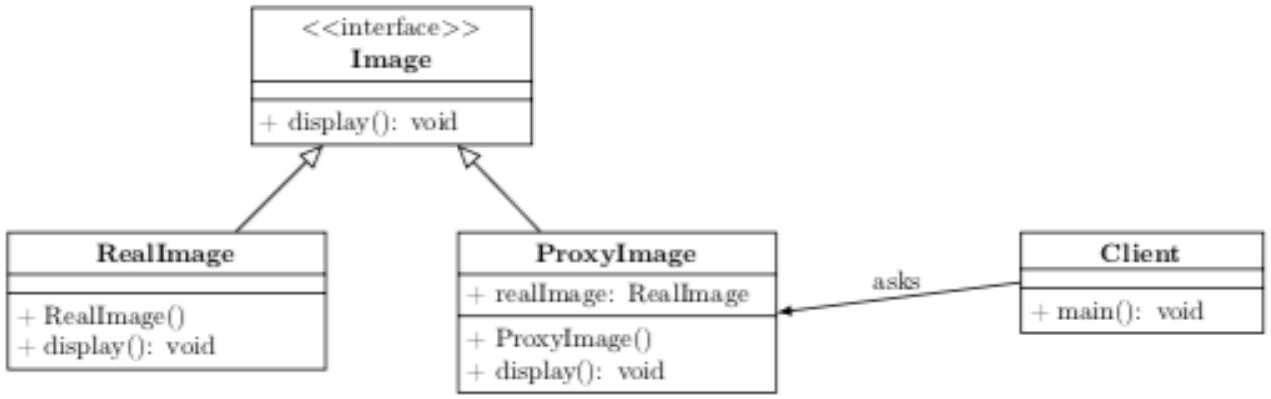
Pattern STRUTTURALI

Decorator
Permette di aggiungere funzionalità a un oggetto esistente, senza modificarne proprietà o struttura: la classe Decorator fa da wrapper all'originale e aggiunge le funzionalità.

Facade
Il pattern Facade nasconde la complessità del sistema fornendo un'interfaccia semplificata al client, che a sua volta chiama i metodi più complessi gestiti internamente, creando così una classe maschera che nasconde al client la complessità del sistema.

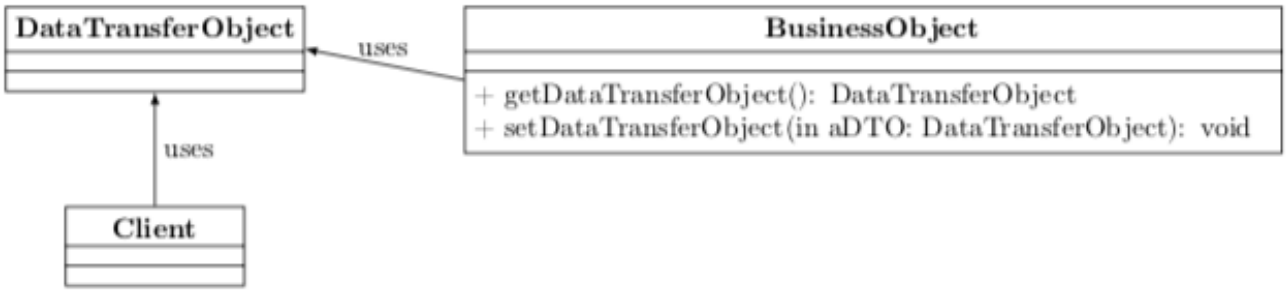
Proxy

Il pattern Proxy rappresenta le funzionalità di un'altra classe, fornendo un surrogato che agisce come sostituto dell'oggetto originale e controlla l'accesso e l'interazione con quest'ultimo da parte del client, gestendo così l'interfacciamento con il mondo esterno.



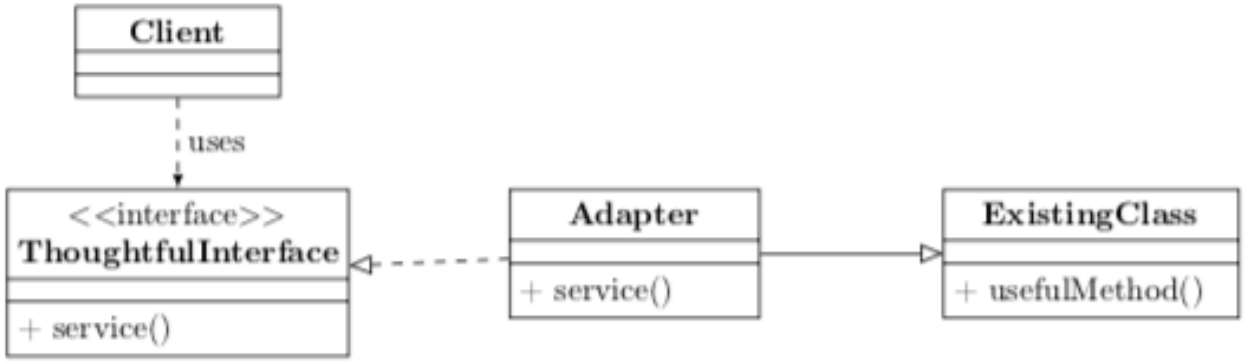
Data transfer Object

E' usato per trasferire dati tra un client e un server, gestendo più oggetti con una singola invocazione, piuttosto che singolarmente. L'oggetto DTO contiene parte dei dati del BusinesObject remoto, non ha metodi comportamentali e permette l'accesso diretto ai dati senza get/set.



Adapter

Converte un interfaccia di una classe in un'altra richiesta dal client, consentendo a diverse classi di collaborare utilizzando interfacce apparentemente incompatibili. La classe Adapter implementa l'interfaccia, estende la classe esistente e ridefinisce il suo metodo chiamando quello richiesto.



Data Transfer Object e Adapter, essendo pattern strutturali, offrono soluzioni specifiche per la gestione e l'organizzazione dei componenti del sistema, risolvendo problemi legati alla compatibilità delle interfacce e alla trasmissione efficiente dei dati.