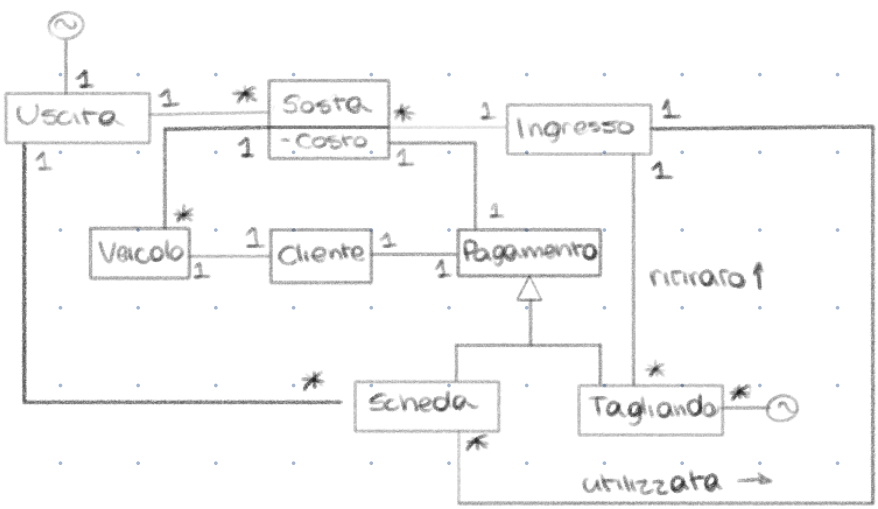
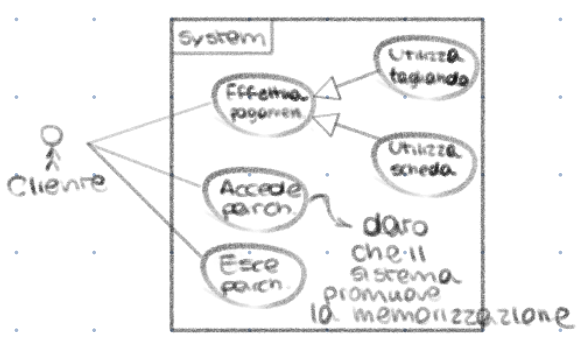
Domain model



Use case



UC: Effettua pagamento
Attore: Cliente

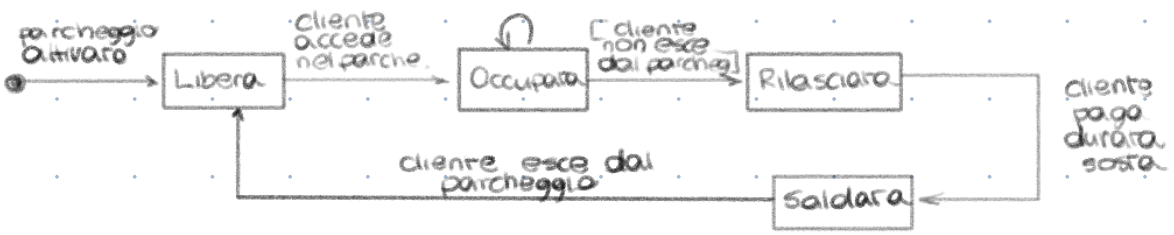
Pre: Cliente abbia sostato nel parcheggio
Cliente ha ritirato tagliando

1. Cliente mostra tagliando
2. Sistema richiede immissione dati veicolo
3. Cliente inserisce dati
4. Sistema mostra totale costo sosta
5. Cliente conferma e provvede al pagamento
6. Sistema mostra maschera di conferma pagamento

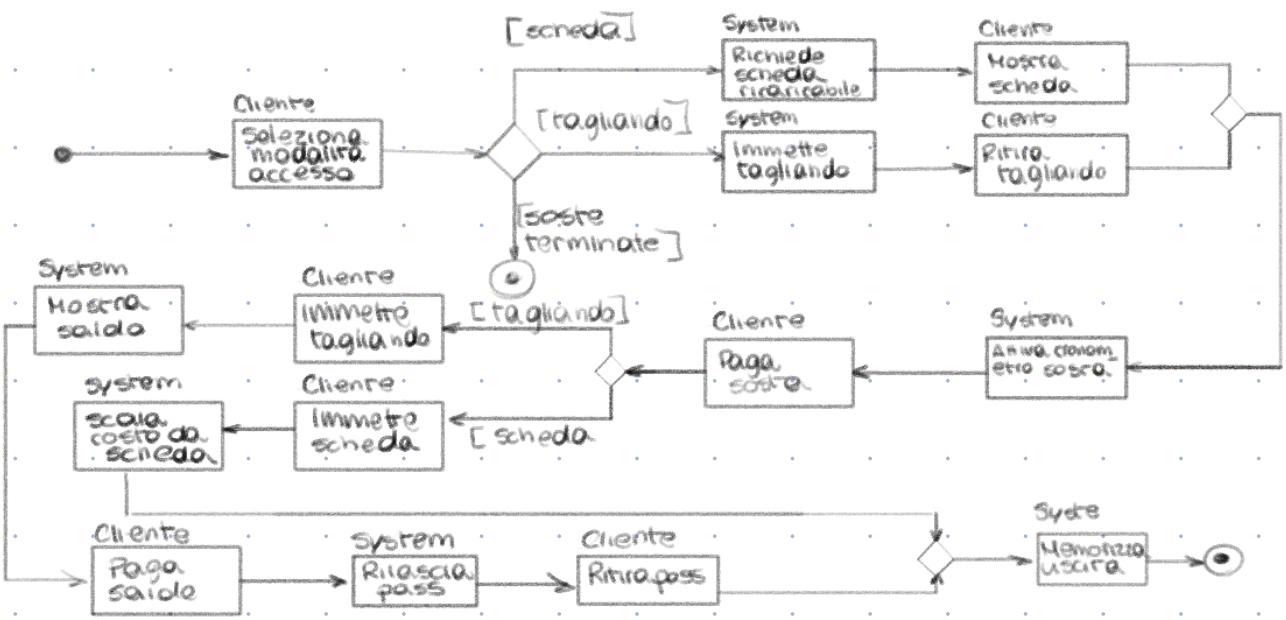
Post: Cliente ottiene pass di uscita.

State machine diagram

1. Occupata
2. Saldata
3. Libera
4. Rilasciata

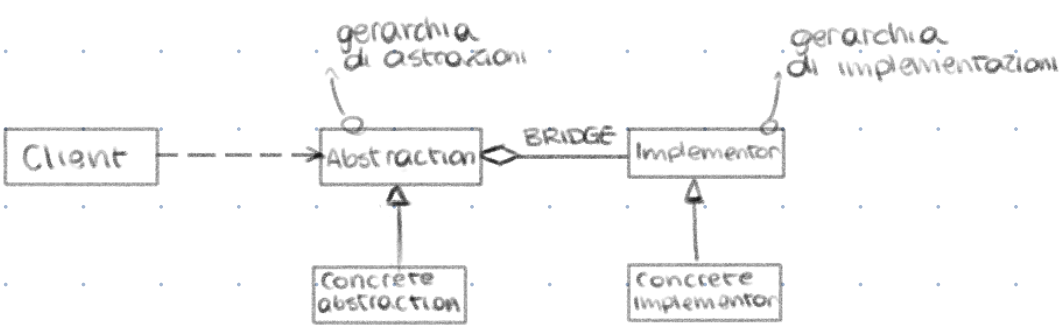


Activity diagram



② BRIDGE

Il problema relazionato al pattern del catalogo GoF stabilisce la volontà di separare l'implementazione comportamentale da condizioni e supposizioni del mondo esterno
Rispetto a quanto detto la soluzione di riferimento é adeguata qualora si tenta di combinare il layer del dominio rispetto al layer della business, dove se attuato potrebbe provocare differenti avversità soprattutto rispetto a contesti in cui modifiche siano difficili da adeguare all'interno del sistema, oltre a rendere la soluzione modellata difficile da riutilizzare
Per cui Bridge propone di adottare una duplice distinzione, contraddistinta in gerarchia di astrazioni e gerarchia di implementazioni in maniera che possano svilupparsi e attuare modifiche indipendemente. Tramite un approccio simile é eliminata la tirannia implementativa imposta da condizioni e supposizioni offerte dal mondo reale, in cui si desidera che soluzioni precedenti si adeguino al modello comportamentale, contrariamente a ciò che é definito all'interno del pattern Adapter, in cui si modella la business logic affinche sia adattiva rispetto alle richieste di high level classes
A livello di utilizzo il pattern strutturale definito promuove



③ OPEN-CLOSED | DIPENDENCY INVERSION

I principi legati al modello SOLID permettono di garantire una software quality di spessore se soddisfatti.
Ciò avviene poiché permettono di gestire correttamente le dipendenze all'interno del sistema software, in cui si ricorda che una dipendenza rappresenta potenzialmente un percorso che diffonde cambiamenti nell'intero modello, causa principale della concretizzazione di design smells, ossia abbassano la qualità del codice implementato
Alcuni pattern in grado di rispondere ad una software quality di spessore sono:

- Open-closed principle, ammette che classi software dovrebbero essere aperte ad estensioni comportamentali ma chiuse rispetto a modifiche di metodi già implementati.

  Una mancata conseguzione potrebbe provocare errori a cascata, poiché se modificate funzionalità di certe entità caratterizzate dalla presenza di dipendenze che usufruiscono dei metodi definiti, anche quest'ultime dovrebbero provvedere ad attuare variazioni comportamentali
  Riassumendo, se aggiunte funzionalità non avviene alcun framredimenta, mentre se modificati metodi già implementati potrebbe concretizzarsi l'avversità posta dalla definizione di dipendenza

- Dipendeny Inversion principle, ammette che classi di alto livello, ossia dedite all'utilizzo di utenti finali dovrebbero dipendere da astrazioni e non da classi di basso livello, dette anche concrete, le quali simboleggiano entità software che sviluppano la logica algoritmica.
  Tipicamente pur di soddisfare un principio simile, é sovrapposto tra le due entità un layer astratto; tipicamente un'interfaccia, tra high level classes e low level classes: ciò avviene per favorire un concreto livello di schermatura affinché modifiche non abbiano ripercussioni su una delle due entità, se non entrambe, prese in gioco.
  Concludendo, se una classe affine al domain layer avesse la necessità di utilizzare funzionalità della business logic, provvederà a creare un'istanza dell'interfaccia, la quale contiene tutti i metodi di riferimento implementati da classi concrete, avviando alla realizzazione di una dipendenza diretta con la business logic.