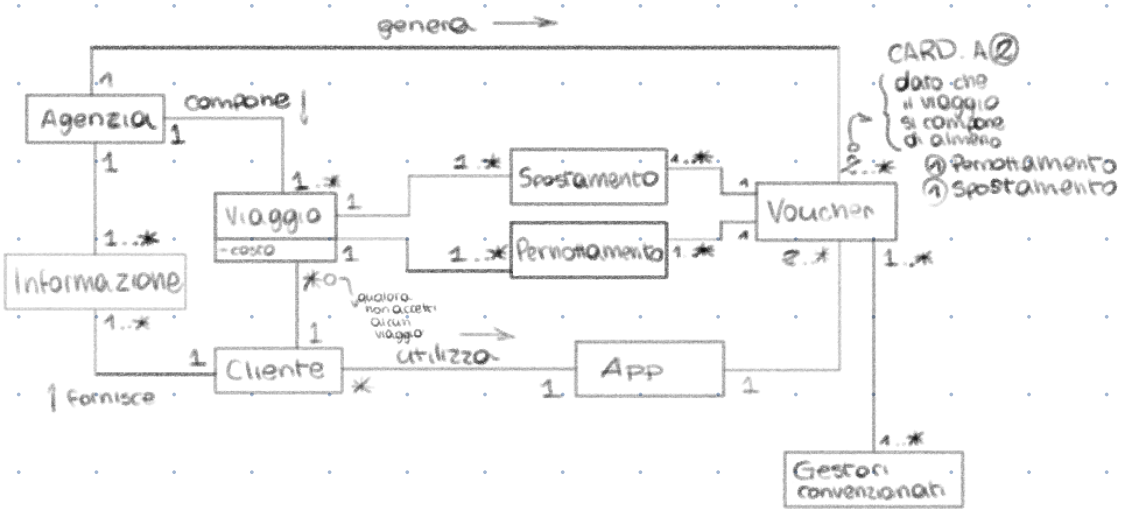
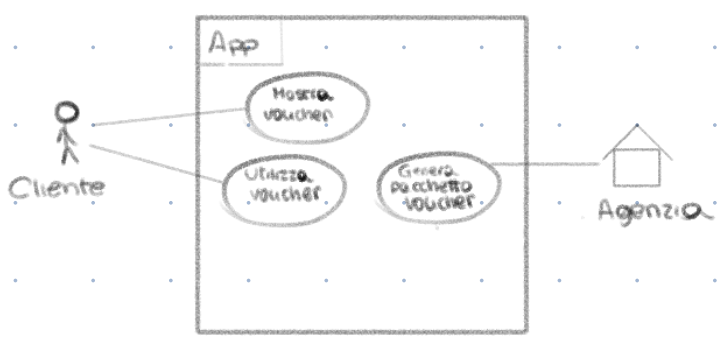
Domain model



Use case

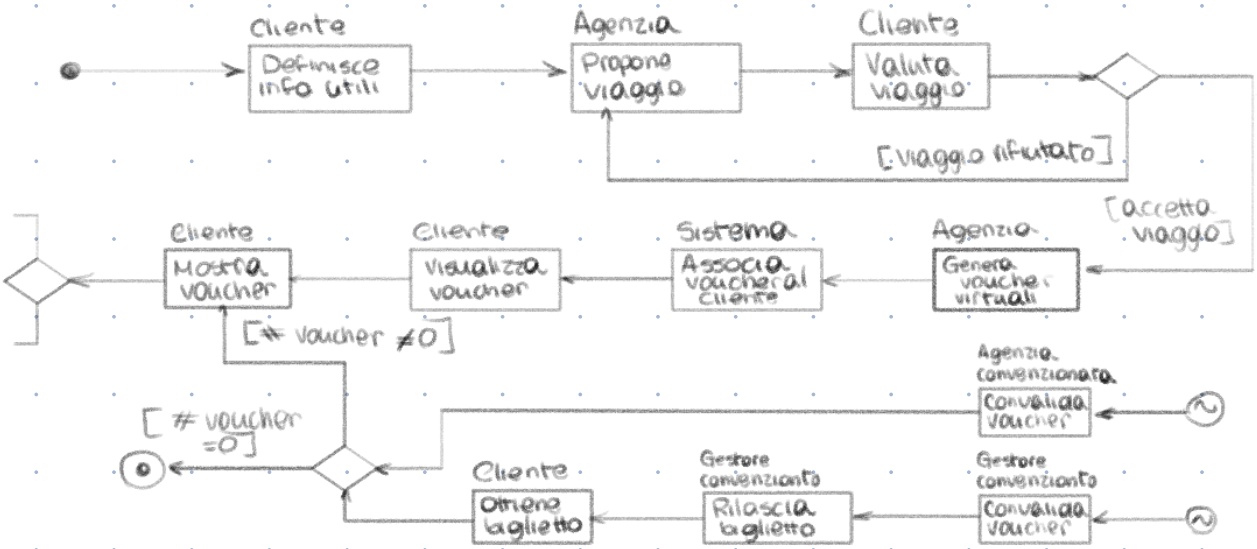


UC: Genera pacchetto voucher
Attore: Agenzia
Pre: Agenzia autenticata dal sistema
Cliente abbia accettato l'offerta del viaggio

1. Agenzia seleziona generazione voucher
2. Sistema richiede dati dei partecipanti
3. Agenzia immette dati
4. Sistema richiede dati dei pernottamenti e spostamenti
5. Agenzia immette dati
6. Agenzia conferma immissione
7. Sistema mostra maschera di conferma inserimento

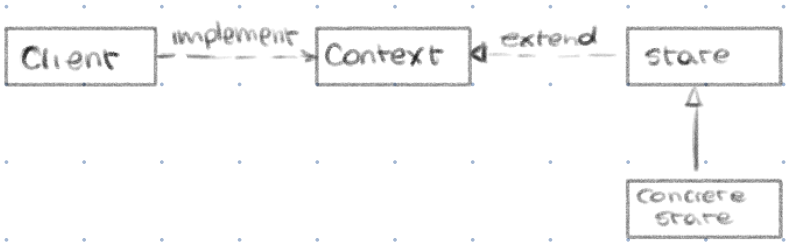
Post: Agenzia consegna pacchetto voucher al cliente

Activity diagram



② STATE

State è un pattern comportamentale del catalogo GOF, per cui appartiene all'insieme di pattern che tentano di favorire un aiuto concreto per implementazioni corrette di doing responsabilities.
Un approccio simile è strettamente correlato al meccanismo di delega o di composizione, attuato dal catalogo GoF e preferito rispetto ad inheritance, poiché favorisce l'intercambiabilità comportamentale durante runtime; si ricorda che la scelta di attuare "composition over inheritance" avviene affinché sia riportata la suddivisione tra domain layer e business logic layer, in maniera tale che le classi siano modellate per favorire un unico behavior, come trattato nell'intuizione implementativa del software pattern high cohesion.
State descrive una certa problematica in cui si richiede l'intercambialità comportamentale di classi software in base allo stato corrente dell'istanza corrente; la soluzione propone di adeguare tra entità esterne e logica algoritmica un'interfaccia che mantenga un riferimento allo stato corrente dell'oggetto osservato, in modo tale che possa adeguare flussi di controllo destinati a modificare il behavior rispetto all'implementazione adeguate nella gerarchia di sottotipi.
Rispetto a quanto detto è l'interfaccia a stabilire il contesto in cui debba avvenire il cambiamento comportamentale, grazie anche all'implementazione delle funzionalità di facciata sviluppate dalla gerarchia di sottotipi, in cui la superclasse contiene il metodo generale caratterizzante lo stato corrente, il quale sarà ereditato e specializzato da sottoclassi in base alle caratterizzazioni acquisibili dall'oggetto esterno.
Molto probabilmente un mancato uso non permetterebbe la intercambiabilità comportamentale durante run time e il sole impiego di principi legati al paradigma OO renderebbe la struttura interna complessa e non adattiva a modifiche oppure ad aggiunte di funzionalità, contrariamente a State in cui sarebbe sufficiente individuare un'ulteriore sottoclasse a cui delegare l'implementazione e lo sviluppo comportamentale, garantendo la totale soddisfazione di OCP e LSP
A livello di domain model promuove



③ OPEN - CLOSED

Open - closed é uno dei cinque principi cardine del modello SOLID, adoperato affinché sia garantita una software quality di spessore, da cui é associata una corretta gestione delle dipendenze, si ricorda che una dipendenza é potenzialmente un percorso che diffonde cambiamenti all'interno dell'intero sistema software, da cui derivano design smells, causa principale della scarsa reusability del codice sviluppato oltre ad un mancato comportamento adattivo dinnanzi a modifiche. Il principio richiede che classi software siano aperte ad estensioni di propne funzionalitá ma chiuse rispetto alle modifiche di metodi giá implementati, una necessitá simile é adeguata affinché non siano violate dipendenze esistenti, in modo tale che non sia innescato il meccanismo che possa richiedere modifiche a cascata all'interno della soluzione modellata, Perranto qualora siano richieste modifiche si attua inheritance, ossia si concretizzano sottoclassi che ereditino funzionalitá a cui sará applicato polimorfismo affinché siano specializzate in relazione alle richieste originane.