OBJECT-ORIENTED SOFTWARE DESIGN
CORSO DI LAUREA IN INFORMATICA
UNIVAQ
A.A. 2018/2019

Pedaggio autostradale

Realizzato da:

Giulia	**Scoccia**	**249503**
Erika	**Biondi**	**247042**
Miriana	**Pompilio**	**248492**

a. Requisiti del sistema

Il sistema fornirà un software in grado di calcolare il pedaggio autostradale , in base ai caselli di entrata e uscita. Sul calcolo influiranno diversi fattori , ovvero : la distanza tra un casello e l'altro (km percorsi * tariffa al km dell'autostrada), la normativa vigente, e la classe del veicolo.

L'utente potrà calcolare il pedaggio inserendo la targa del veicolo e i caselli di entrata ed uscita.

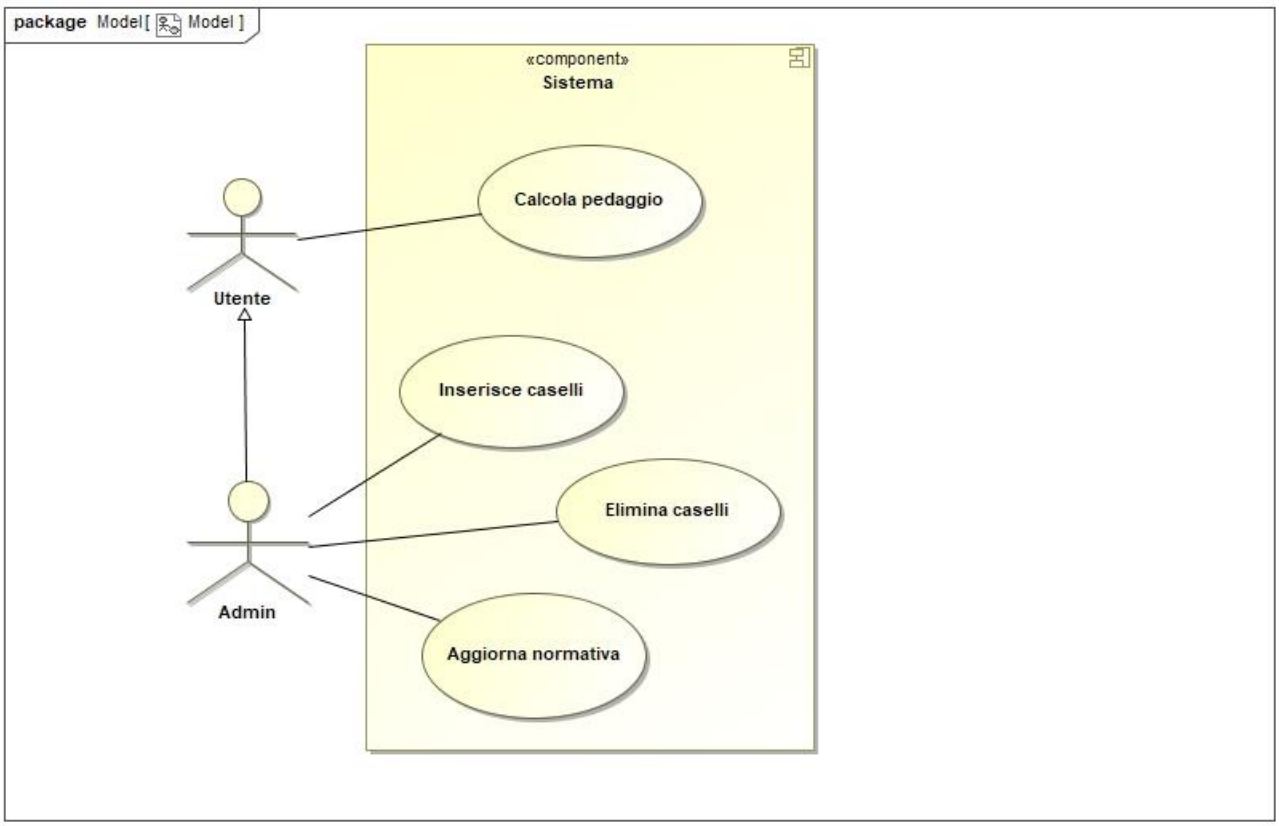
Il sistema fornirà due tipi di interfacce , una per l'utente attraverso la quale si potrà soltanto calcolare il pedaggio e un'altra per l'amministratore che potrà svolgere operazioni di inserimento, cancellazione dei caselli , oppure potrà aggiornare la normativa corrente. Ovviamente l'admin avrà, a sua volta, anche la possibilità di calcolare il pedaggio.

- Requisiti funzionali

Utente: calcola il pedaggio

Amministratore: inserisce e/o elimina i caselli, aggiorna la normativa, calcola il pedaggio.

• Use Cases



- ### Requisiti non funzionali

Il sistema deve essere di facile utilizzo.

L'interfaccia deve essere user friendly.

Il sistema deve essere espandibile.

Il sistema deve essere scalabile.

b. Architettura del sistema

- Obiettivi di Design

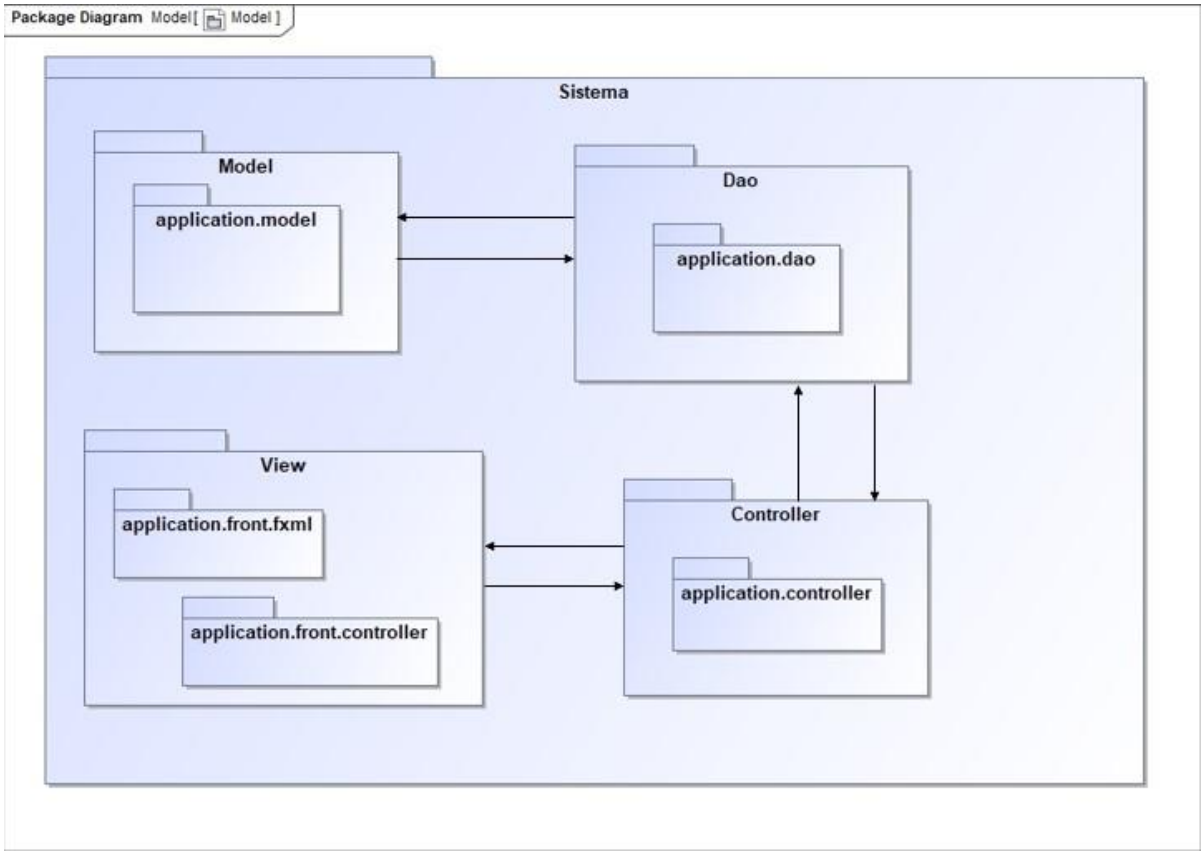
Il più importante obiettivo di Design è la scalabilità del sistema, in quanto bisogna tener conto che il sistema è in continuo aggiornamento, ovvero può subire dei cambiamenti, ad esempio per quanto riguarda le normative che si aggiornano quasi annualmente e quindi comportano la modifica delle classi veicolari. Per questo è opportuno che il sistema preveda la possibilità di effettuare determinate modifiche , senza impattare sulla funzionalità e senza dover stravolgere l'intero sistema.

- Strategie per la costruzione del sistema

Per quanto riguarda la realizzazione delle interfacce grafiche è stata utilizzata la tecnologia Java Fx attraverso il tool SceneBuilder. Tutte le view sono raccolte all'interno del package application.front.fxml. L'architettura software del sistema è basata sul pattern MVC (Model View Controller) in modo da riuscire a separare la parte statica da quella dinamica. La persistenza dei dati è garantita da un database MySql. Il lavoro è stato interamente implementato all'interno dell'ambiente di sviluppo Ecplise.

- Descrizione dell’architettura

Diagramma con Packages



L’architettura del sistema è stata suddivisa in 3 livelli:

1. Nel package application , al primo livello, si trova la classe principale del sistema, il main.

2. Al secondo livello , troviamo i package application.controller (controller relativi ad ogni classe) , application.dao (dichiarazione e implementazione delle query per la gestione dei dati) , application.front (terzo livello) e application.model (dichiarazioni delle entità principali del sistema).

3. Al terzo livello troviamo application.front.controller (controller sulla gestione delle action sulle view) e application.front.fxml (file.fxml che rappresentano le view).

- Design Pattern e Pattern Architetturali

I design pattern e i pattern architetturali utilizzati al fine della realizzazione del sistema sono :

1. MVC , utilizzato per distinguere la logica di presentazione dei dati dalla logica di business. E' basato su tre ruoli principali:

- Model, che fornisce i metodi per accedere ai dati dell'applicazione.

- View, che visualizza i dati contenuti nel model e si occupa dell'interazione con gli utenti.

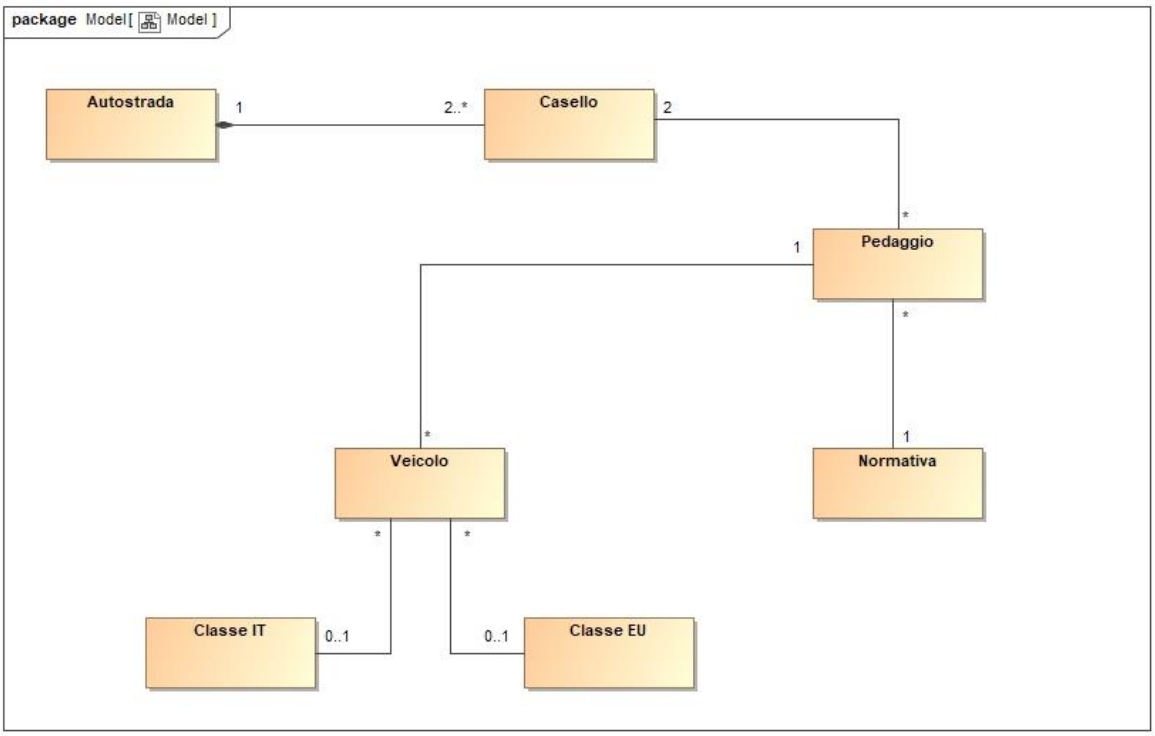
- Controller, che riceve comandi dall'utente attraverso le view e li attua modificando lo stato degli altri due componenti

2. Dao Factory, il pattern Data Access Object è un modello strutturale che consente di isolare il livello applicazione dal livello di persistenza dati. La funzionalità è di nascondere all'applicazione tutte le complessità coinvolte nell'esecuzione delle operazione sui dati nel DB. Ciò consente ad entrambi i livelli sopra citati di evolversi indipendentemente. Il vantaggio relativo all'uso del DAO è il mantenimento di una separazione tra le componenti Model e Controller di un'applicazione basata sul paradigma MVC.

3. Singleton , la funzione del singleton è quella di garantire un'unica istanza di una determinata classe. Il costruttore del singleton è stato definito come private e , in tal modo , l'unico punto di accesso alla classe viene fornito attraverso il metodo statico getInstance() , che si occupa di restituire l'unica istanza della classe.

c. Progettazione di classi e interfacce

- Class Diagram



- Descrizione delle classi

Le entità che compongono il nostro sistema sono le seguenti:

Autostrada , identificata da un suo id univoco che ha come attributi :

la sua lunghezza totale (km) , la tariffa al km , il nome , l'inizio e la fine.

Ogni Autostrada è formata da almeno due *Caselli* , anche essi identificati da un id, ed inoltre hanno un attributo che serve a localizzare a che altezza si trovano lungo l'autostrada (altezza_km) , e il nome del casello.

L'entità *Pedaggio* effettua il calcolo per quanto riguarda il pagamento una volta usciti dall'autostrada, in base al casello di entrata e di uscita e alla targa del veicolo.

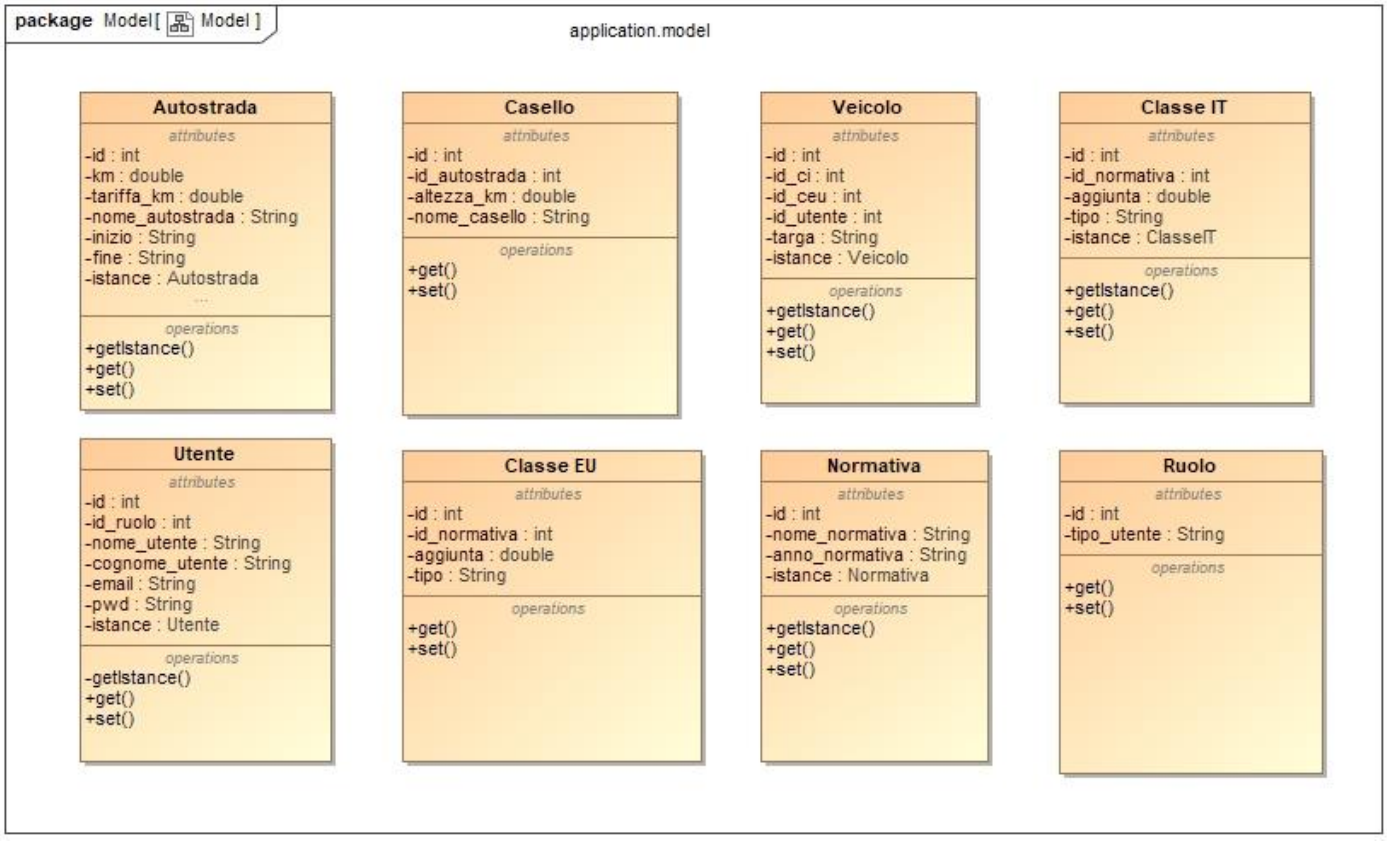
Inoltre, il calcolo del pedaggio si effettua in base a una *Normativa*, che ha un nome (può essere o Italiana o Europea) e l'anno in cui entra in vigore.

Infine c'è l'entità *Veicolo* , che è descritto da un numero di targa, un numero di assi , e l'altezza ed è collegato all'entità della classe di appartenenza.

Le entità *ClasseIT* , e *ClasseEU* , servono per determinare l'aggiunta da applicare nel calcolo del pedaggio.

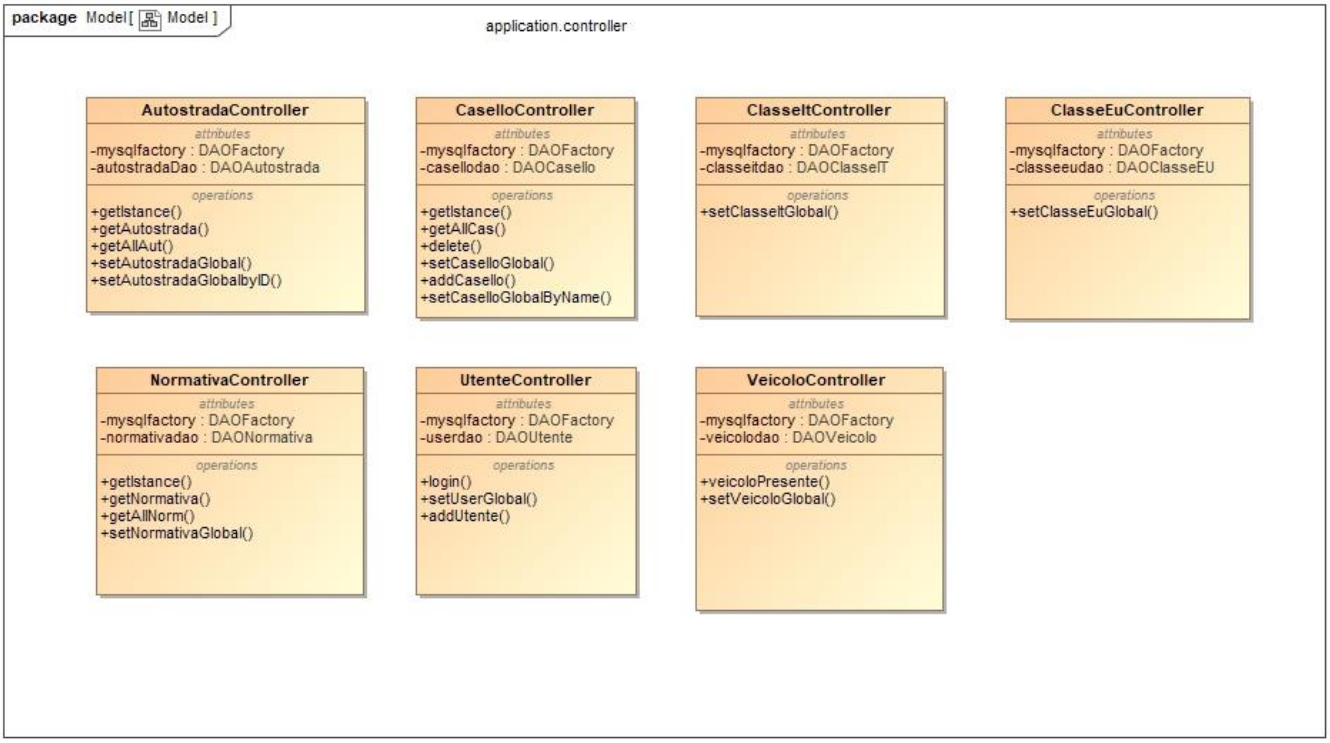
- Descrizione dei dettagli di design scelti

- **MODEL**



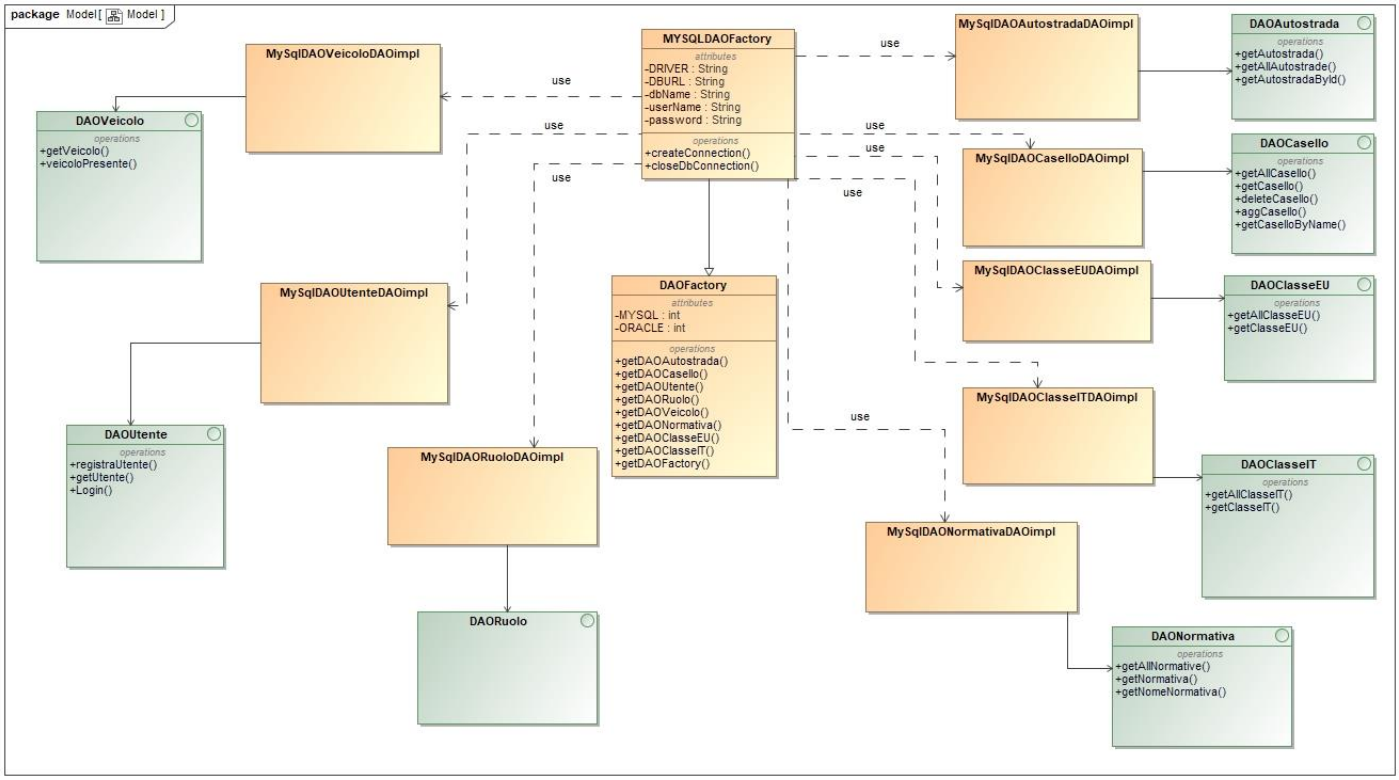
Tutto ruota attorno al model, esso interpreta il comportamento dell'applicazione in maniera del tutto indipendente dall'interfaccia utente. Il modello gestisce i dati, la logica e le regole dell'applicazione.

- **Controller**



Questo componente ha la responsabilità di trasformare le interazioni dell'utente della View in azioni eseguite dal Model. Ma il Controller non rappresenta un semplice "ponte" tra View e Model esso implementa la logica di controllo dell'applicazione.

- **Dao**



Per quanto riguarda il DAO, si tratta fondamentalmente di una classe con relativi metodi che rappresenta un'entità tabellare, usata per stratificare e isolare l'accesso ad una tabella tramite query (poste all'interno dei metodi della classe) creando un maggiore livello di astrazione ed una più facile manutenibilità. I metodi del DAO con le rispettive query dentro verranno così richiamati dalle classi della business logic.