Ingegneria del Software: Assegnamento finale

Eugenio Ancona VR359699
Giacomo Annaloro VR359866
Sara Drezza VR361237
Davide Mazzoni VR359125
Michele Scala VR359312
Matteo Zanoncello VR359107

1 Descrizione della struttura del progetto

Il secondo assegnamento riguarda la progettazione di un sistema di gestione del traffico in una porzione di strada, e in particolare del rispetto dei limiti di velocità. Il sistema è composto da una torre centrale che comunica via wireless con le automobili che percorrono la zona controllata. Le automobili sono di due tipi: automatiche, in grado di reagire autonomamente ai comandi della torre, e manuali, che si limitano a mostrare al guidatore le istruzioni su un display. La torre e le macchine comunicano attraverso due tipi di canali: un canale ideale utilizzato per i messaggi della torre e per quelli delle macchine nella fase di registrazione nel sistema, e 5 canali utilizzati dalle macchine per aggiornare la torre sulla propria velocità attuale. La capacità di questi ultimi è limitata, e quindi in caso si saturino impediranno la registrazione di nuove auto nel sistema.

Nota: Lo schema UML delle classi si trova in un file a parte

1.1 Tower

La classe `Tower` rappresenta la torre centrale; è dotata di una mappa in cui associa ad ogni id (che identifica univocamente una macchina) se la velocità della macchina corrispondente rispetti o meno i limiti, aggiornata in base ai messaggi che le macchine stesse inviano. Inoltre istanzia i diversi `CarChannel` che le automobili utilizzano per comunicare con la torre. Per comunicare con le macchine invece utilizza un `TowerChannel` ideale di capacità infinita. Utilizzando un timer, che rappresenta la velocità di invio dei pacchetti, la torre invia `JoinMessage` alle macchine che non sono ancora registrate, `TowerMessage` ad ogni macchina comunicandole se la velocità attuale è entro i limiti o se è necessario rallentare e `Registermessage` per comunicare alle macchine appena entrate nel sistema il canale attraverso cui comunicare. Il packet rate della torre nel nostro progetto è diverso da quello indicato nelle specifiche sia per permettere una visualizzazione della simulazione più agevole, sia perché da specifica il rapporto con il packet rate delle macchine era troppo basso: se la torre mandasse solo il doppio dei pacchetti delle macchine automatiche (o il quadruplo di quelle manuali), come da specifica, potrebbe appunto gestire efficientemente solo fino a due macchine automatiche o 4 manuali; un numero superiore di macchine

causerebbe ritardi nelle risposte e quindi non garantirebbe il rispetto dei limiti da parte di tutte le vetture.

1.2 CarChannel

La classe `CarChannel` rappresenta il mezzo di comunicazione delle auto verso la torre. Le istanze di questa classe, generate al bisogno dalla `Tower`, non fanno altro che trasmettere gli `SpeedMessage` delle diverse vetture alla torre. Ogni canale tiene anche conto di quanta parte della sua banda è occupata al momento.

1.3 TowerChannel

`TowerChannel` rappresenta il mezzo tramite il quale la torre comunica con le automobili nel sistema. A differenza del `CarChannel` non ha limiti di capacità. Viene usato eccezionalmente anche per la comunicazioni nel verso contrario solo durante la fase di registrazione di un nuovo veicolo al sistema, prima che gli venga assegnato un `CarChannel`.

1.4 Car

I due diversi tipi di macchine sono stati implementati dalle classi `ManualCar` e `AutomaticCar`; poiché i comportamenti delle due tipologie di automobili non differiscono molto, entrambe estendono la classe astratta `Car` che fattorizza tutte le caratteristiche comuni: tiene conto della velocità attuale e contiene le referenze agli oggetti `SendBehavior` e `receiveBehavior` e ai canali, oltre ad offrire tutti i metodi per gestire il display. In effetti la maggior parte delle differenze nei comportamenti di `ManualCar` e `AutomaticCar` sono gestite dai due oggetti behavior, ma abbiamo comunque deciso di modellare questa gerarchia per una maggiore mantenibilità, in caso in futuro si aggiungano altre differenze o altri tipi di vettura.

1.5 SendBehavior e ReceiveBehavior

La classe `SendBehavior` rappresenta l'applicazione dello Strategy Pattern per distinguere i comportamenti dei due tipi di macchine. Da specifica la differenza nel comportamento in invio dei messaggi è il packet rate, mentre per quanto riguarda il comportamento in ricezione la macchina manuale lascia che sia il guidatore a reagire alle direttive della torre, laddove la macchina automatica reagisce autonomamente frenando se necessario.

1.6 Messages

Abbiamo raggruppato sotto un interfaccia marker `Message` tutti i diversi tipi di messaggio che vengono spediti all'interno del sistema:

- `JoinMessage`: inviato dalla `Tower` alle nuove macchine che si avvicinano all'area controllata
- `OKMessage`: inviato dalle macchine in risposta al `JoinMessage`
- `RegisterMessage`: Inviato dalla `Tower` per registrare un'automobile al sistema, riservandole uno spazio su un `CarChannel`

- `SpeedMessage`: inviato dalle macchine con frequenza fissa, per aggiornare la torre sulla propria velocità attuale
- `TowerMessage`: inviato periodicamente dalla torre per dare direttive alle automobili controllate
- `ExitMessage` inviato da una macchina per informare la torre che sta per uscire dall'area di controllo

1.7 Adapter

Le specifiche richiedevano di implementare l'Adapter pattern per poter mascherare un `ManualCar` da `AutomaticCar`. A questo scopo abbiamo modellato le classi `SendAdapter` e `ReceiveAdapter` che estendono le classi dei behavior automatici ma istanziano degli oggetti di tipo `ManualSend` e `ManualReceive` ed eseguono in realtà i metodi di quest'ultimi. Il codice che inserisce nel sistema le macchine adattate è commentato nella classe `Simulator` in quanto il simulatore non gestisce il comportamento del guidatore delle `ManualCar` adattate, ritenendole `AutomaticCar` e la velocità della macchina non rispetta i limiti.

1.8 Simulator

La classe `Simulator` si occupa di simulare gli stimoli esterni per verificare il corretto funzionamento del sistema. In particolare si occupa di generare le macchine che cercano di entrare nella zona controllata e di simulare le variazioni di velocità che la torre deve monitorare. Abbiamo inoltre scelto di utilizzare questa classe come interfaccia Façade per la grafica, essendo quella che ha una maggiore visione di insieme.

1.9 MainWindow e InfoBean

`MainWindow` è la principale classe grafica che si occupa di visualizzare l'andamento della simulazione del sistema. Per aggiornare i dati visualizzati fa affidamento sul solo metodo `Simulator.getInfoBean()` che restituisce tutti i dati necessari alla visualizzazione, facendo uso della classe ausiliaria `InfoBean` che appunto incapsula tutte le informazioni necessarie.

2 Design pattern applicati

Strategy Lo Strategy Pattern è stato applicato alla classe astratta `Car`; infatti viene estesa da `ManualCar` e `AutomaticCar` che sono due tipi di macchine che differiscono sia per la velocità nell'inviare pacchetti, sia per il modo in cui reagiscono a un messaggio della torre che li avvisa dell'eccessiva velocità. Quindi sono state aggiunte le due classi astratte `SendBehavior` e `ReceiveBehavior`, estese dalle classi effettive che descrivono i comportamenti dei due diversi tipi di auto nel mandare e nel ricevere i messaggi.

Observer L'Observer Pattern viene usato nella comunicazione tra i nodi `Tower` e ognuna delle `Car` che sono presenti. Infatti, per esempio, la macchina manda un messaggio, cioè notifica l'update, alla torre ogni volta che cambia la sua velocità.

Facade Il Facade Pattern coinvolge le classi `Simulator`, `InfoBean` e `MainWindow`. Infatti il `Simulator` si preoccupa di fornire un'interfaccia semplice per la `MainWindow`, prendendo i dati che quest'ultima dovrà mostrare a video dalle varie classi del sistema e organizzandole in un `InfoBean`. In questo modo alla classe `MainWindow` non interessa quali classi ci sono e come funzionano, perché le vengono fornite tutte le informazioni necessarie dal `Simulator`.

Adapter Infine, è stato sviluppato anche l'Adapter Pattern come richiesto, per convertire una macchina automatica in una macchina manuale.