Università della Calabria – Facoltà di Ingegneria – Corso di laurea in Ingegneria Informatica

Ingegneria del Software

aa 2006/2007

Traffic Flow System

Relazione di Progetto

Studente

Falace Gabriele

matricola 74956

Indice

Progettazione concettuale

Comportamento del sistema

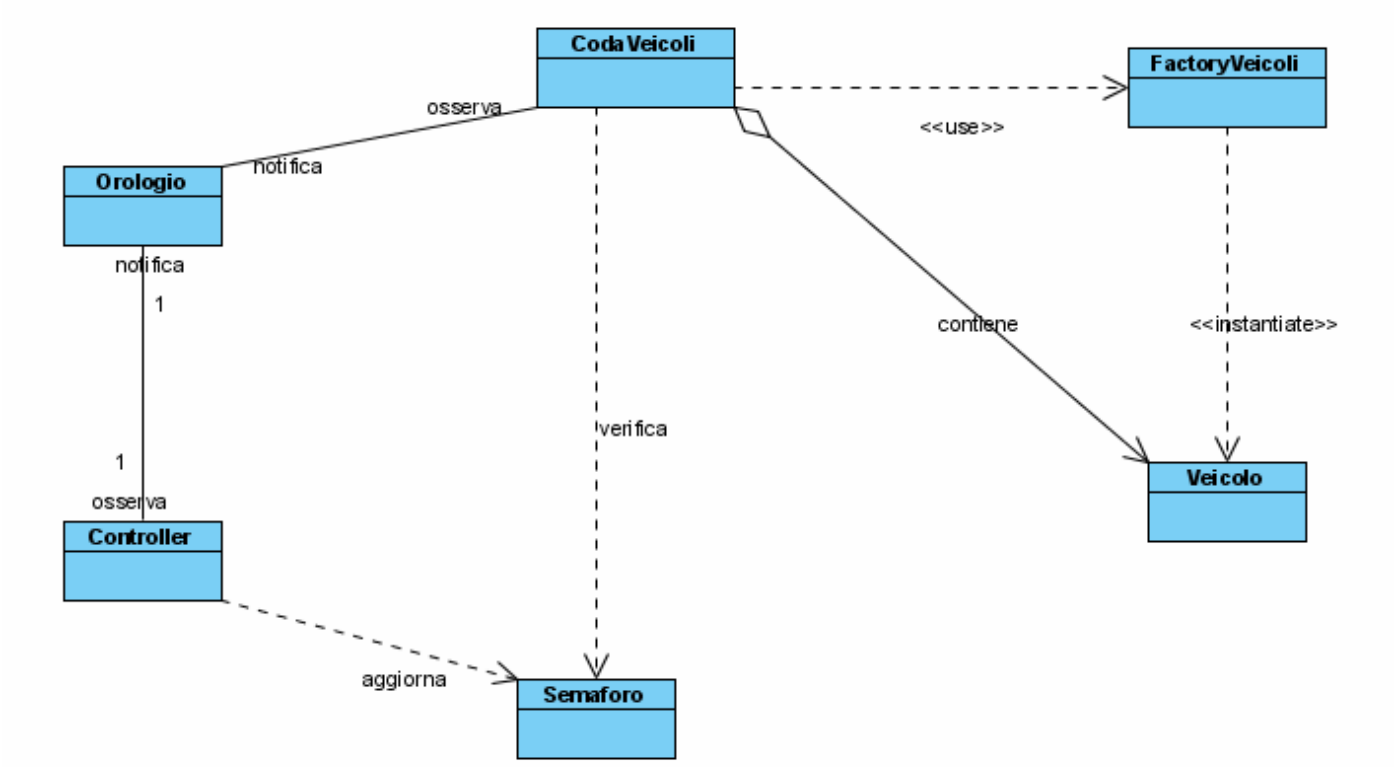
Class Diagrams raffinati

Principali casi di test

Progettazione Concettuale

In questa prima fase di progetto si identificano la principali entità coinvolte nel sistema software da realizzare.
Dal momento che la specifica include numerosi punti già specificati (nel senso che sono indicate più o meno esplicitamente le soluzioni da adottare) si propone il seguente diagramma UML delle classi , piuttosto di alto livello, per la modellazione concettuale.

Diagramma delle classi preliminare

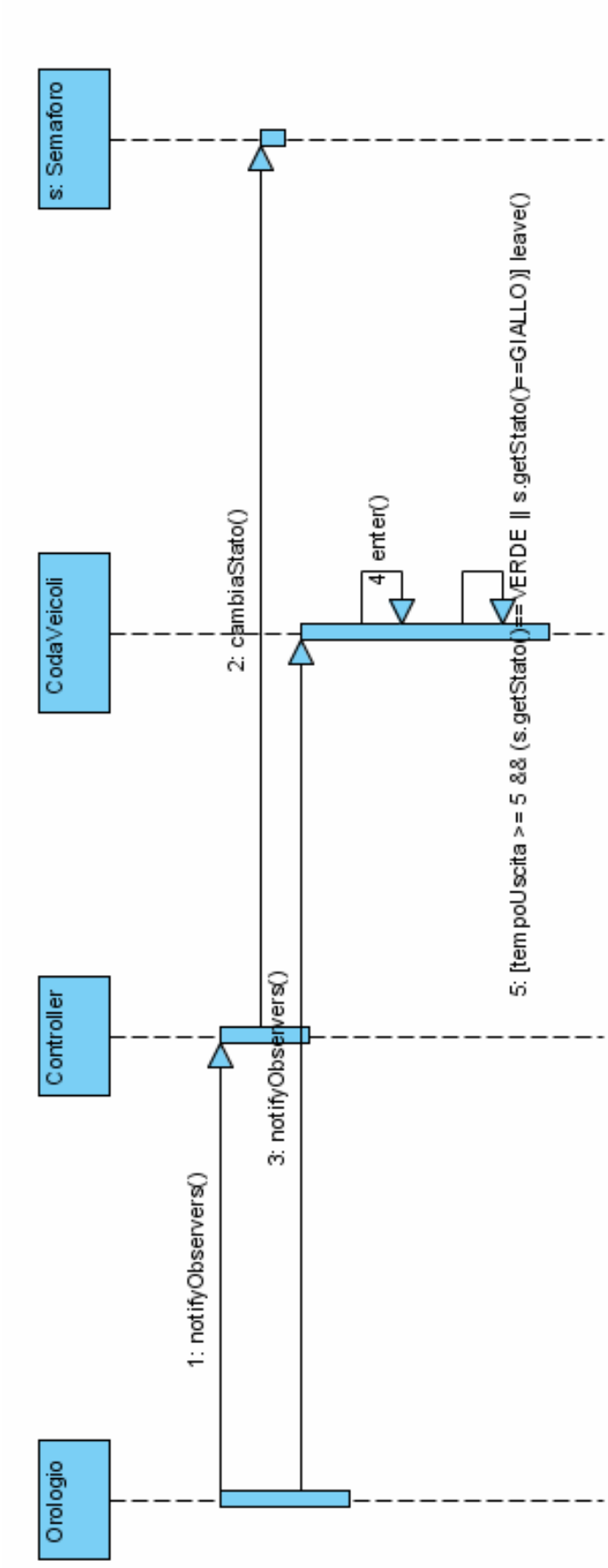


L'idea è quella di realizzare una classe Orologio che, utilizzando un timer, mantiene il tempo e, allo scoccare di ogni secondo, notifica appunto il passaggio del secondo al controller ed alle code.
Il controller aggiorna i semafori come da specifica. La coda veicoli (probabilmente) fa entrare un certo veicolo in coda e, se il semaforo lo permette, fa passare un veicolo ogni 5 secondi (come da specifica).
Per far entrare veicoli nella coda, questa si avvale dell' uso di apposito factory che, secondo probabilità, crea un veicolo e lo mette in coda.

Comportamento del sistema

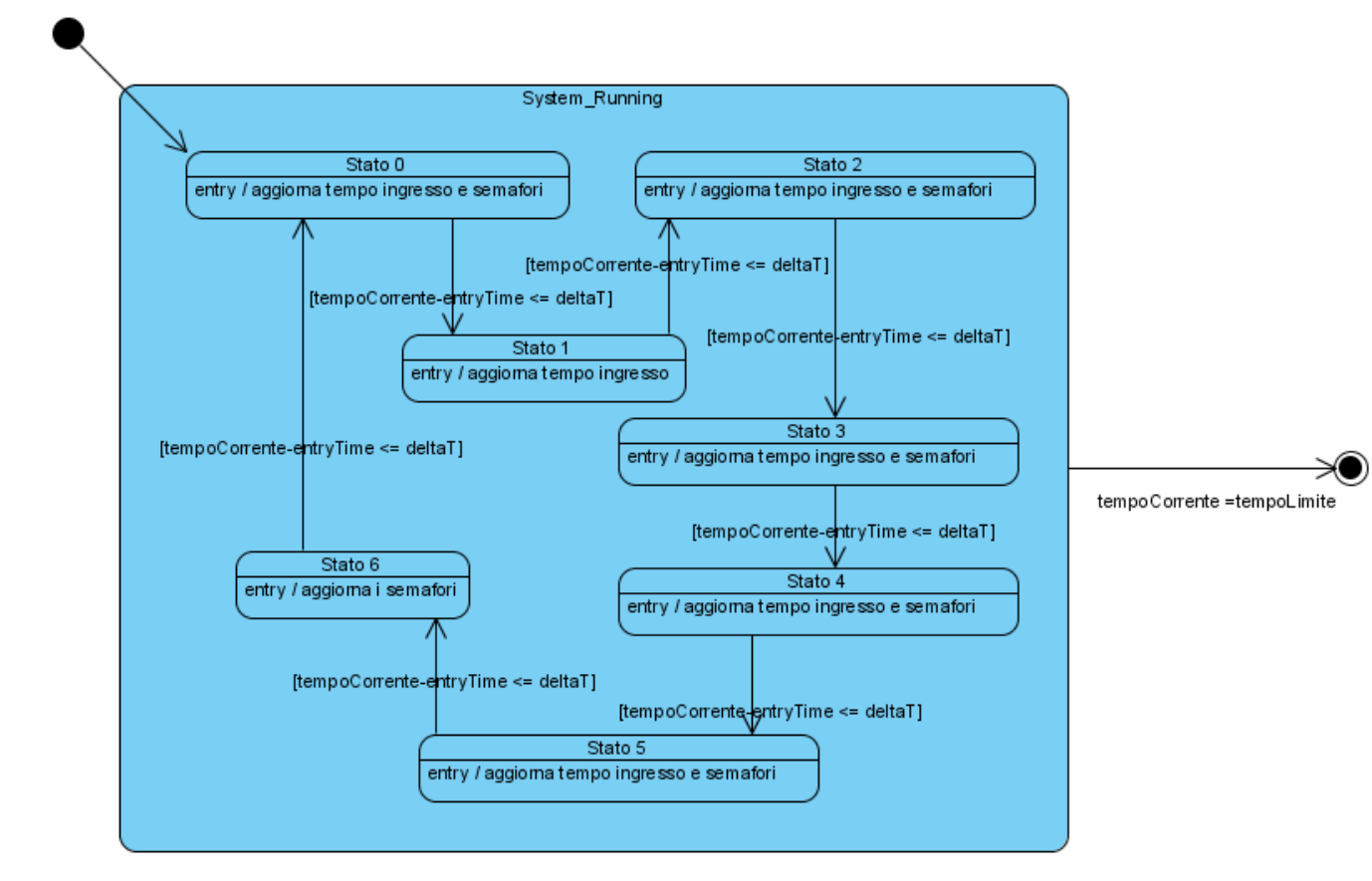
Occorre, ora, andare più in dettaglio sul comportamento del sistema e sulle operazioni associate a ciascuna delle entità delineate precedentemente.
In questo modo sarà possibile assegnare meglio le responsabilità agli oggetti che verranno progettati.

Diagramma di sequenza



Il diagramma di sequenza sopra mostra in modo più schematico e preciso la dinamica del sistema, come descritta nella fase preliminare - concettuale.
L' Orologio notifica sia le code sia il controller. Questo cambia lo stato dei semafori, in accordo con il protocollo dato da specifica (dettagliato nel seguito).
Quando le code ricevono la notifica che un secondo è trascorso, fanno entrare un veicolo nella loro coda chiamando l'apposito metodo enter() e cercano per quanto sia possibile di chiamare leave() per fare lasciare la coda ai veicoli in essa, naturalmente in ordine FIFO.
Precisamente, la coda verifica il fatto che il semaforo sia verde o giallo e qualora lo sia, fa passare un veicolo. Per accertarsi che il prossimo veicolo lasci la coda dopo 5 secondi da quello appena passato (come da specifica) viene mantenuta una variabile posta a 0 quando passa un veicolo e incrementata ad ogni colpo del clock: quando tale variabile raggiunge 5, allora il veicolo precedente è passato (in quanto sono trascorsi 5 secondi) e può essere nuovamente chiamato leave(), resettando la variabile.

Diagramma di stato del controller



il diagramma mostra un altro importante aspetto della dinamica del sistema: l'evoluzione degli stati del controllore.

In particolare ogni stato è caratterizzato dal fatto che, salvo richieste di passaggio, dopo un certo ΔT il controllore cambia autonomamente stato, compiendo una transizione spontanea.
L'idea è di utilizzare una variabile "entryTime" in cui viene memorizzato l'istante di tempo in cui il sistema entra nello stato e calcolando ad ogni istante il ΔT come

ΔT = tempoCorrente – entryTime

cosicché appena il sistema raggiunge il ΔT predefinito dello stato corrente, passa nello stato successivo.
Questa dinamica è subordinata al fatto che un veicolo richieda il passaggio, cosa che avviene al suo ingresso nella coda.

Seguono le specifiche in OCL dei ΔT relativi a ciascuno stato,

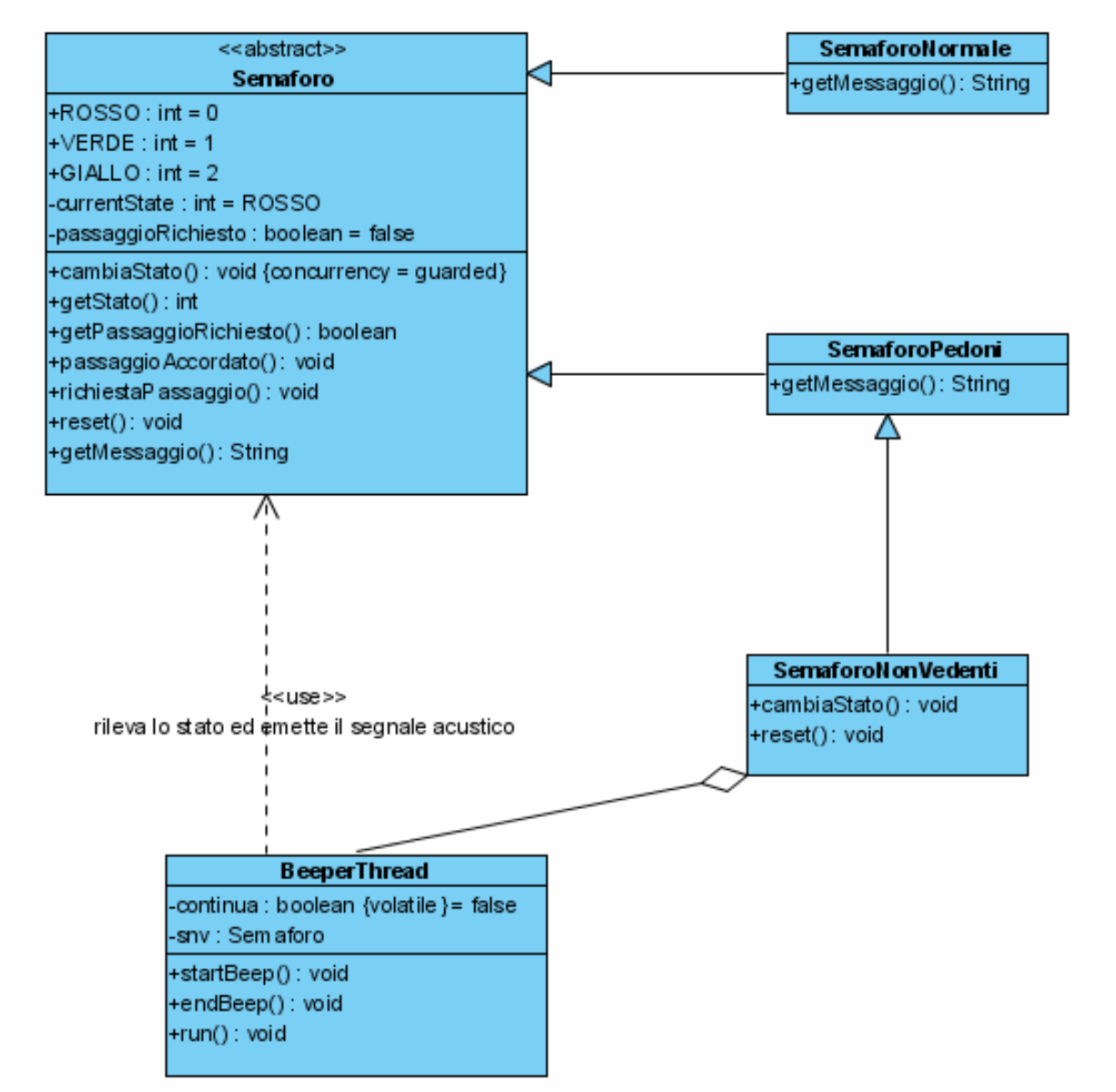
Specifiche in OCL

```
Context Stato0
        inv : self.deltaT = 120

Context Stato1
        inv: super.deltaT = Math.random()*60

Context Stato2
        inv: self.deltaT = 12

Context Stato3
        inv: self.deltaT = 5

Context Stato4
        inv: self.deltaT = 60

Context Stato5
        inv: self.deltaT = 20

Context Stato6
        inv: self.deltaT = 5
```

Class Diagrams raffinati

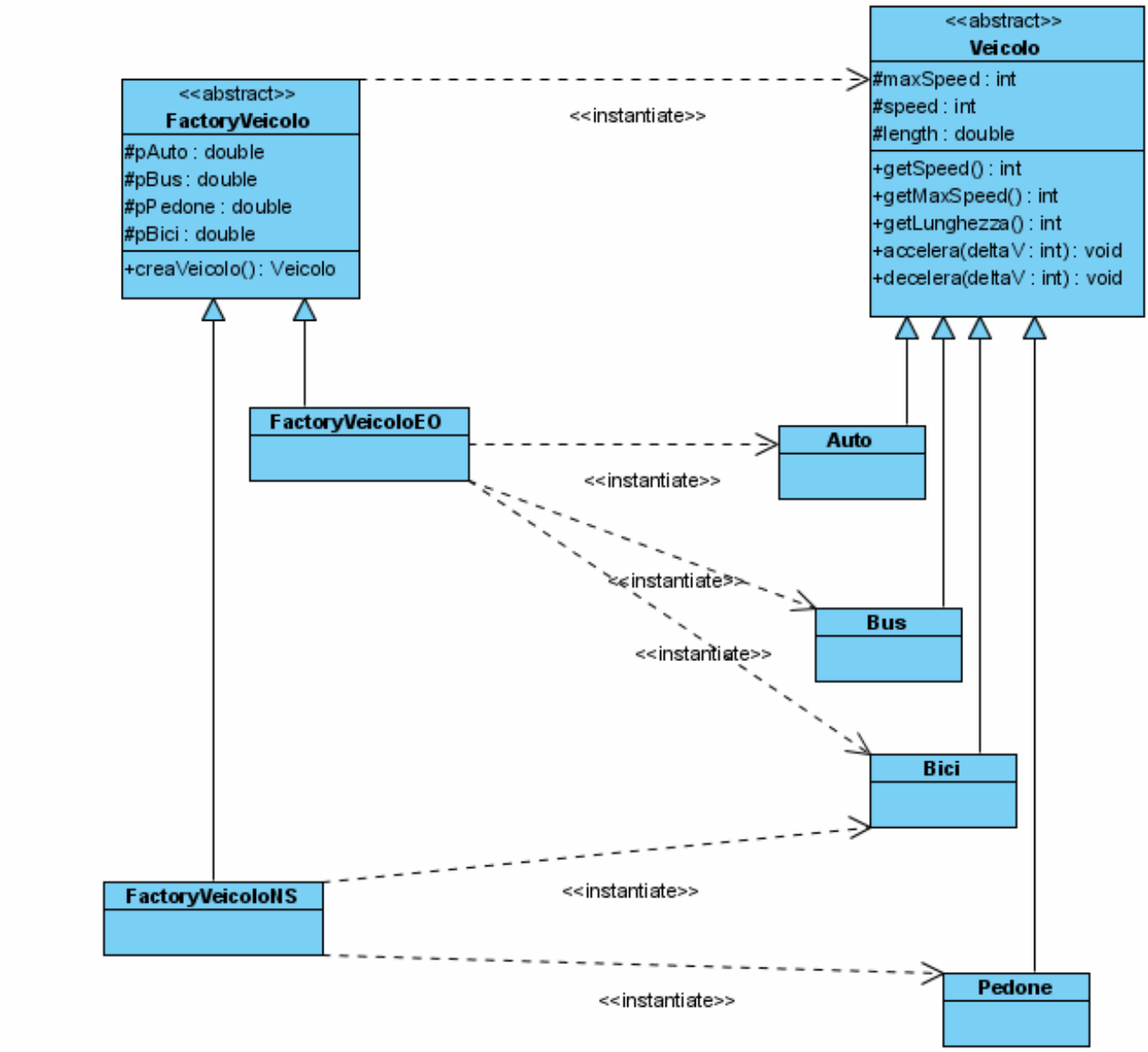
A questo punto si possono dettagliare i componenti del sistema attribuendo loro i ruoli esaminati in precedenza.
In particolare si aggiungono gli opportuni attributi e metodi nei diagrammi di classe concettuali progettati inizialmente, seguendo quanto esposto nell'esame del comportamento del sistema appena svolto.

I Semafori



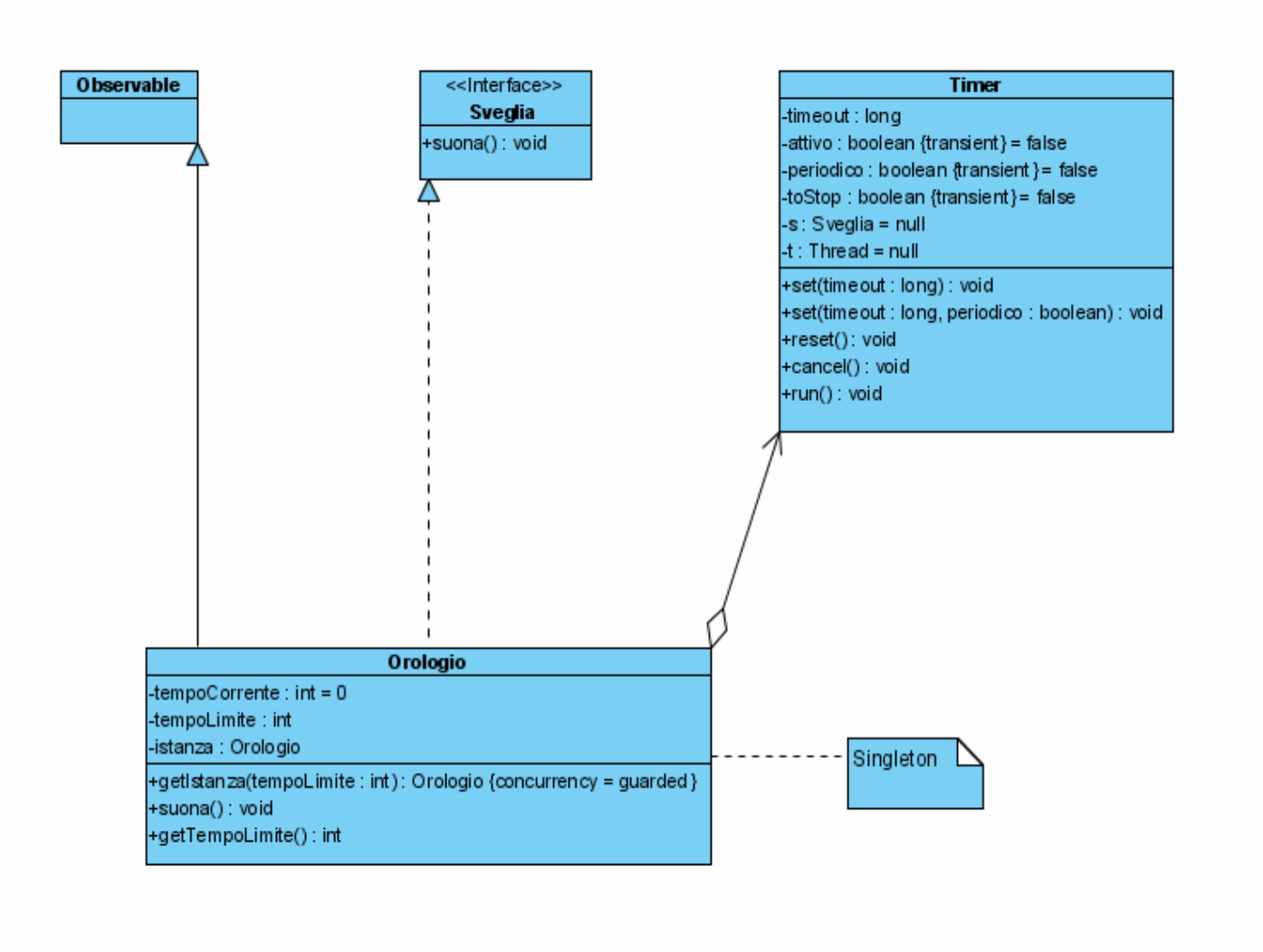
l’oggetto semaforo è fondamentale per il sistema: rappresenta i semafori dell’incrocio distinguendo, grazie ad alcune specializzazioni, i normali semafori dai semafori pedonali e da quelli per i non vedenti.
In generale un semaforo è dotato di uno stato interno che può essere ROSSO, VERDE o GIALLO. Quando viene invocato cambiaStato(), lo stato scatta (circolarmente) nello stato successivo secondo l’ordine ROSSO, VERDE,GIALLO. Una variabile passaggioRichiesto serve per memorizzare se presso quel semaforo è stato richiesto il passaggio. Inoltre i semafori per non vedenti sono muniti di un thread interno che emette un segnale acustico quando il semaforo è verde, segnale che aumenta di frequenza quando il semaforo diventa giallo e sparisce quando questo si fa rosso.

Il factory dei veicoli



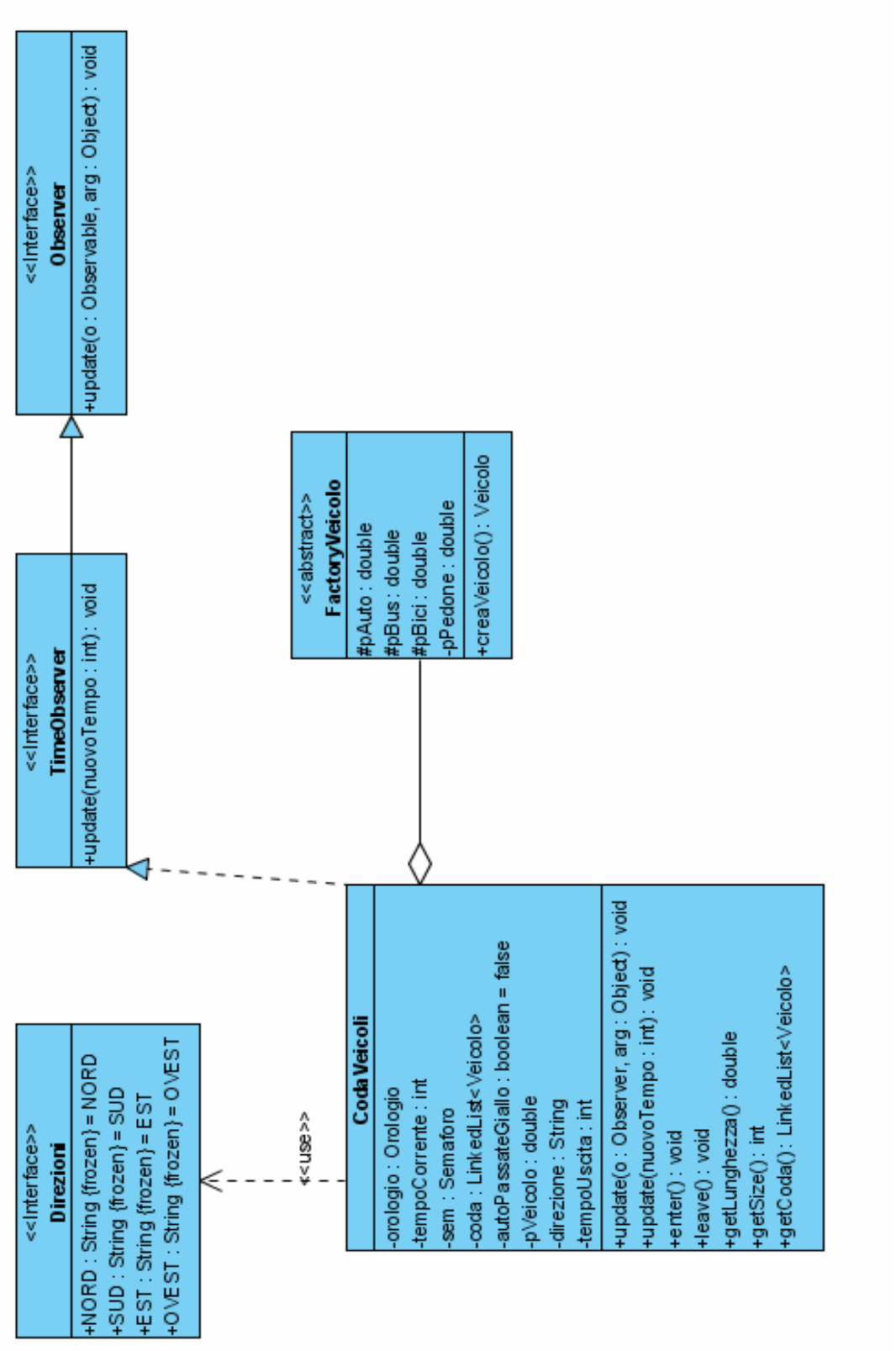
Il factory dei veicoli è utile perché, ad ogni istante, la coda dei veicoli prova a fare entrare un nuovo veicolo nella coda. Il punto è che la creazione di un particolare veicolo è legata alla probabilità che si presenti proprio quel tipo di veicolo per cui la coda non potrebbe semplicemente invocare il costruttore di un tipo particolare. L'utilizzo di un apposito FACTORY fa in modo che sia questo ad occuparsi di istanziare un veicolo secondo l'opportuna probabilità (metodo creaVeicolo()) e di ritornarlo al chiamante.

L’ orologio del sistema



L’Orologio del sistema è un oggetto che, utilizzando un Timer (package is) si occupa di registrare il tempo, implementando l’interfaccia Sveglia.
Lo scopo principale dell’oggetto è, poi, quello di rendere disponibile tale tempo a tutti gli oggetti che fossero interessati, mediante la notifica del passaggio dei secondi. A tal fine viene estesa la classe Observable, in modo da realizzare un OBSERVER in modalità push, nel senso che è l’oggetto osservato (l’Orologio) a passare il dato aggiornato (il nuovo valore di tempoCorrente) agli osservatori.
Come richiesto dalla specifica, viene fatto in modo che possa esistere solo una istanza di Orologio impiegando l pattern SINGLETON. Il costruttore privato e il metodo statico getIstanza() garantiscono, appunto, che l’oggetto già istanziato “internamente” dalla classe non possa essere istanziato nuovamente dall’ esterno.

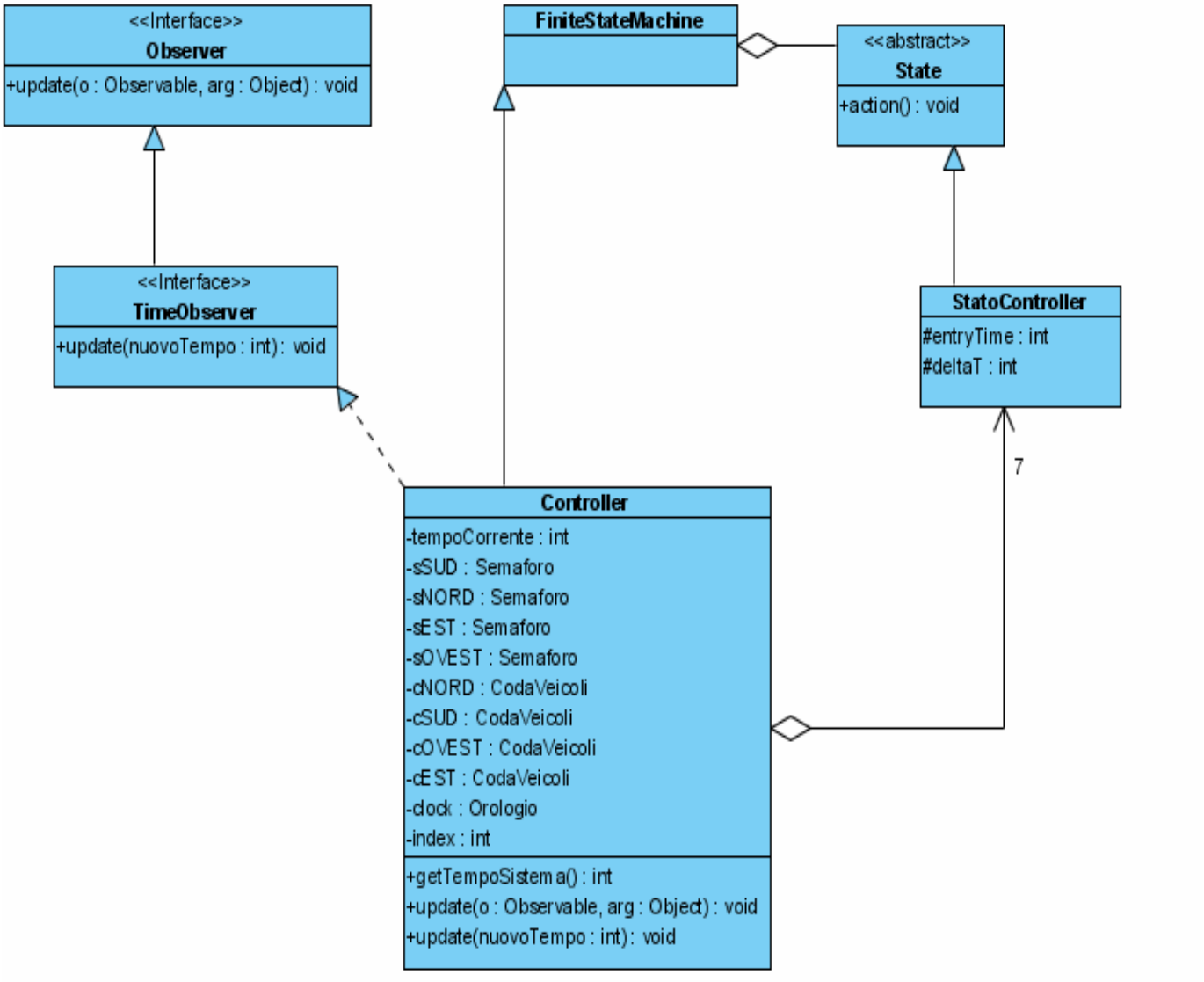
Le code dei veicoli



Il comportamento delle code dei veicoli è quello già esposto in precedenza: quando vengono notificate, chiamano enter() per far entrare, con una certa probabilità, un

nuovo veicolo in coda, dopodiché controllano lo stato del loro semaforo e qualora sia possibile chiamano leave() in modo da far abbandonare la coda al primo veicolo. Dal momento che ciascun veicolo impiega 5 secondi a passare, la coda misura (variabile "tempoUscita") il tempo tra il passaggio di due veicoli, così viene chiamata leave() quando il semaforo è verde o giallo e sono passati almeno 5 secondi dal passaggio dell'ultimo veicolo.

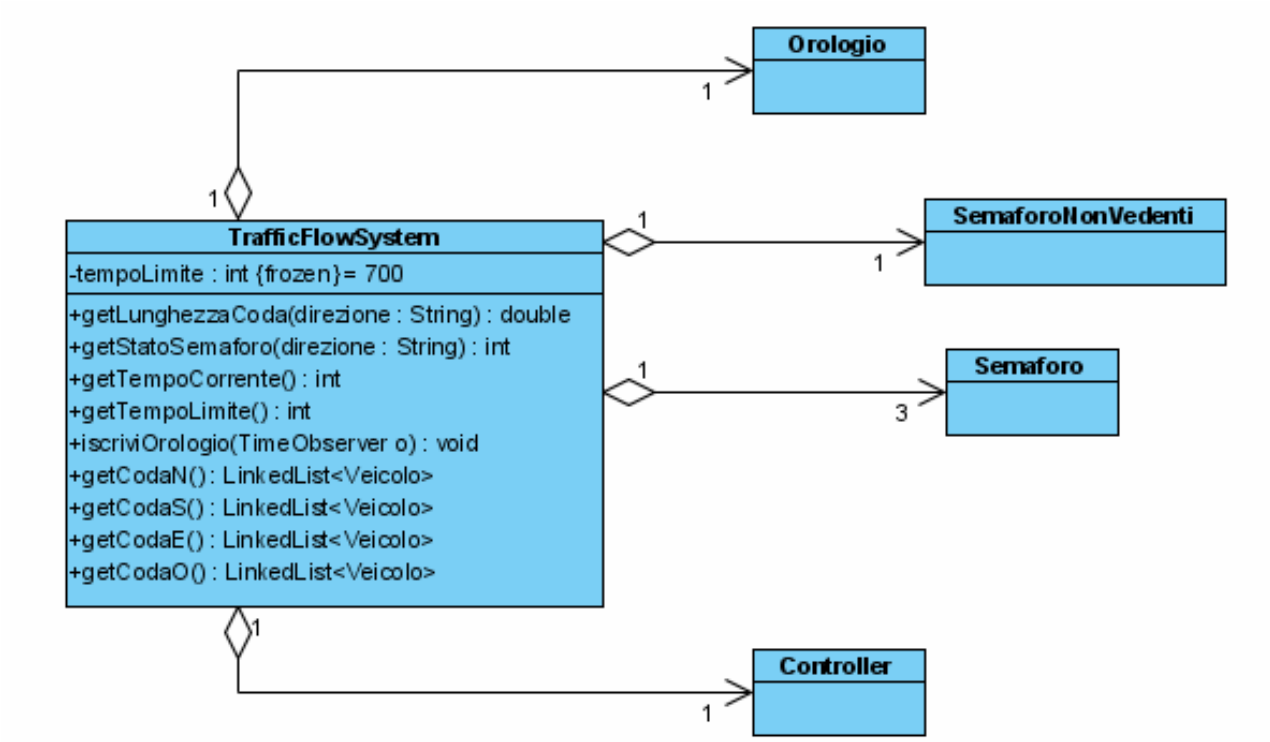
Il controller



Il Controller è l'oggetto che si fa carico del corretto funzionamento dei semafori. Il Controller è realizzato mediante il pattern STATE e la dinamica dell' evoluzione negli stati è realizzata mediante il meccanismo descritto in precedenza. In particolare

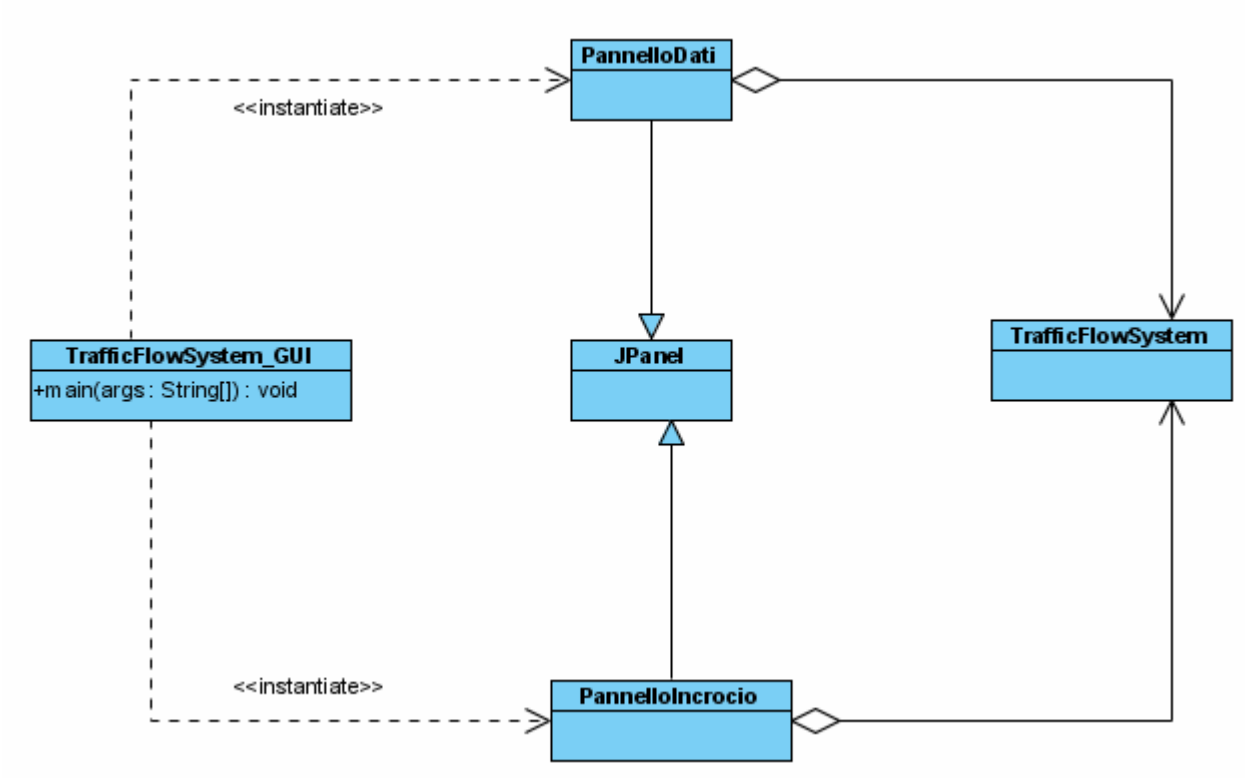
la macchina a stati finiti è il Controller stesso, che aggrega 7 stati ognuno dei quali contiene le variabili “deltaT” ed “entryTime”.
Essendo un TimeObserver, il controllore viene notificato ad ogni secondo e ad ogni secondo, quindi, controlla se deve portarsi in un nuovo stato ed in tal caso effettua la transizione modificando lo stato dei semafori e registrando il tempo di ingresso nel nuovo stato.

Il sistema



L’ oggetto TrafficFlowSystem fa un po’ da “aggregatore” del sistema, in quanto ha il compito di istanziare gli oggetti, passando a ciascuno gli opportuni riferimenti di cui ha bisogno per funzionare correttamente.

La GUI



La GUI è costituita da due pannelli, collocati in uno JSplitPane, ciascuno dei quali funge da osservatore del sistema; I due pannelli sono anch' essi dei TimeObserver e pertanto ricevono notifica del passare dei secondi dall' Orologio. Quando ricevono tale notifica, i pannelli si ridisegnano, catturando lo stato del sistema in quel dato istante. In particolare:
Il PannelloDati mostra numericamente il tempo di sistema e la lunghezza delle code dei veicoli, oppure il numero di veicoli in coda nelle code che ammettono la presenza di pedoni.
Il PannelloIncrocio mostra, invece, graficamente l'incrocio in modo da visualizzare i veicoli in coda e lo stato dei semafori. In questo pannello la grafica non è eccezionalmente bella, ma questo non è stato ritenuto di primaria importanza, visto che è comunque sufficiente modificare il solo PannelloIncrocio per migliorarla e che il sistema inoltre è pensato non avere interazione con l'utente.

Specifiche OCL degli aggiornamenti

/*
* *Transizione di stato del controller susseguente l' update*
*/

Context

Controller::update(nuovoTempo: int) : void

pre:

(s[index] == s[0] **and**
(sNORD->getPassaggioRichiesto() *or* SUD.getPassaggioRichiesto())**) or**
S[index].deltaT < tempoCorrente – s[index].entryTime

post:

index = (index@pre + 1)%7
tempoCorrente = nuovoTempo

/*
* *Chiamata di leave susseguente l' update*
*/

Context

CodaVeicoli::update(nuovoTempo: int) : void

pre:

self.coda->notEmpty **and** self.tempoUscita >= 5 **and (** self.sem->getStato() ==VERDE
or self.sem.getStato()==GIALLO **and not** autoPassateGiallo**)**

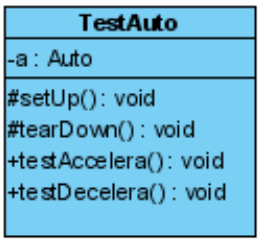
post:

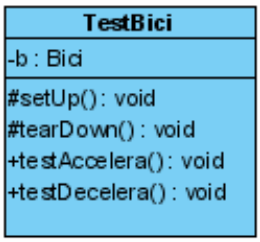
self.coda->isEmpty **or** self.coda->size = self.coda->size@pre -1
self.tempoUscita = (tempoUscita@pre + 1)%5
self.tempoCorrente = nuovoTempo

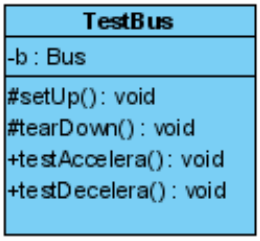
Principali casi di test

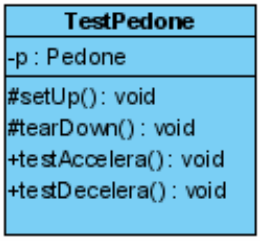
Sono infine riportati i diagrammi dei principali casi di test eseguiti sul sistema.

Test dei veicoli

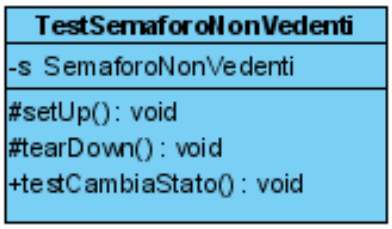


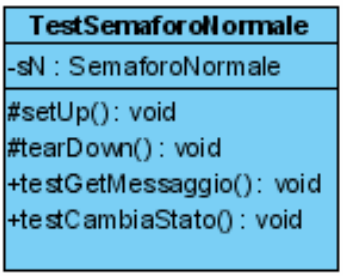




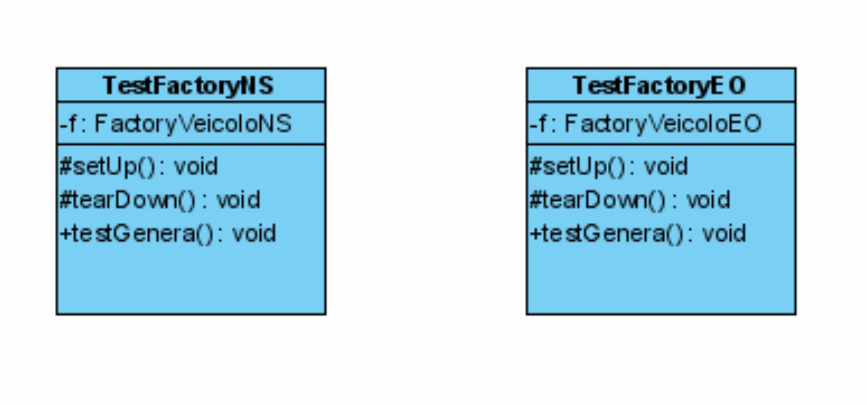


Test dei semafori

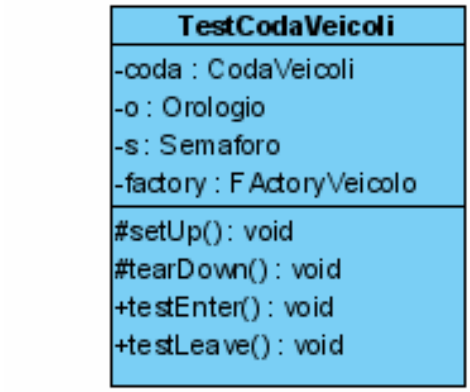




Test dei factory



Test delle code di veicoli



dal momento che questi rivestono un ruolo importante nel funzionamento del sistema, si analizza più in dettaglio la logica del metodo update() del controller e delle code dei veicoli