

POLITECNICO DI MILANO
Ingegneria Industriale e dell'Informazione



DOCUMENTO FINALE
Progetto: Traffic Monitor

Progetto di laurea di:
Francesco Fiori Matr. 872755
Luigi Pederzani Matr. 867521

Anno accademico 2018/2019

INDICE

REQUISITI

TESTO DEL PROGETTO	5
STUDIO FATTIBILITÀ	6
MOTIVAZIONI E OBIETTIVI	6
DEFINIZIONE DEI REQUISITI	6
ANALISI DEL RISCHIO	6
SDM	7
SINTESI DEI GOAL	8
SRM - UTENTE.....	8
SRM - APPLICAZIONE MOBILE	9
SRM - SISTEMA CENTRALE	10
SRM - CENTRALINA STRADALE	11
SRM - CENTRALINA AUTOMOBILISTICA.....	12
DATA DICTIONARY	13

DESIGN

USE CASE DIAGRAM	15
CLASS DIAGRAM	16
SISTEMA CENTRALE.....	16
CENTRALINA	18
APPLICAZIONE MOBILE.....	19
MODIFICHE CLASS DIAGRAM POST JAVA	21
OBJECT DIAGRAM	22
COMPONENT DIAGRAM	23
DEPLOYMENT DIAGRAM	24
SEQUENCE DIAGRAMS	25
COLLABORATION DIAGRAM	28
STATE DIAGRAMS	29
ACTIVITY DIAGRAMS	30

JAVA

CONFIGURAZIONE	33
SISTEMA CENTRALE	35
GESTIONE DATI IN INGRESSO - A&S.....	35
ELABORAZIONE STATO - A&S	35
GESTIONE STATO - R	37
GESTIONE NOTIFICHE - A&S	39

GESTIONE NOTIFICHE - R	39
GESTIONE UTENTI - A&S.....	41
GESTIONE UTENTI - R.....	41
RMI - SETTAGGIO SERVER.....	42
GRAFICA	43
CENTRALINA STRADALE.....	45
STRUTTURA	45
CALCOLO STATO - A&S.....	45
CALCOLO STATO - R	46
FREQUENZA DI AGGIORNAMENTO	47
GRAFICA	48
RMI - SETTAGGIO LETTURA	48
APPLICAZIONE MOBILE	49
AUTENTICAZIONE.....	49
PAGINA PRINCIPALE.....	51
SEGNALAZIONE	52
CAMBIO POSIZIONE	53
NOTIFICA	54
NOTIFICA PUSH.....	54
LISTA NOTIFICHE.....	55
DATABASE	56
TESTING CON JUNIT.....	57
JUNIT TEST APPLICAZIONE	57
JUNIT TEST CENTRALINA STRADALE	58
JUNIT TEST SISTEMA CENTRALE	59

REQUISITI

TESTO DEL PROGETTO

Realizzare un sistema per il monitoraggio e il controllo integrato del traffico cittadino, composto dai seguenti sotto-sistemi che operano in modo distribuito:

- **Sistema centrale:** incaricato di memorizzare tutte le informazioni di stato, inviare notifiche a sistemi esterni in caso di specifici eventi, mostrare lo stato dell'intero sistema e sottosistemi. Il sistema quindi include una interfaccia utente che consente di esplorare le varie informazioni attuali.

Opzionale: è possibile decidere di mostrare i dati anche in un qualche tipo di forma grafica (diagrammi, mappe. ecc.).

- **Centraline stradali:** incaricate di monitorare il flusso di traffico del segmento stradale in cui collocate e inviarlo al sistema centrale con periodicità proporzionale all'ammontare di traffico.

- **Centraline automobilistiche:** incaricate di inviare con periodicità fissa il dato di velocità (e posizione) del veicolo su cui sono installate.

- **Applicazioni mobili:** installate su telefono cellulare e incaricate di inviare al sistema centrale esplicite segnalazioni di traffico (coda, con posizione GPS) da parte degli utenti / guidatori. Le applicazioni inoltre ricevono notifiche dal sistema centrale per qualsiasi evento di traffico (coda, velocità lenta, traffico elevato) in un raggio fisso dalla posizione (ultima registrata) del telefono.

Specificare, progettare e implementare il sistema distribuito necessario, coprendo: sistema centrale, applicazione mobile, e una a scelta tra centralina stradale e centralina automobilistica.

Definire esplicitamente tutti i formati dei dati scambiati e le modalità di scambio (protocollo).

È possibile raffinare i requisiti ed aggiungere ipotesi e assunzioni sul contesto, sensate e in linea con quanto indicato nei requisiti. Tali estensioni devono essere esplicitamente riportate nella documentazione di progetto (sezione specifica requisiti).

STUDIO FATTIBILITÀ

Motivazioni e Obiettivi

Il progetto Traffic Monitor ha come finalità quella di raccogliere dati sul traffico in tempo reale per consentirne il monitoraggio e l'analisi, mostrando e comunicando agli utenti le informazioni riguardo lo stato del traffico.

L'obiettivo per l'azienda è lo sviluppo di un sistema per il monitoraggio del traffico che sia scalabile e adattabile ad ogni città a cui sarà venduto.

Ad ogni città acquirente verrà fornito l'intero sistema centrale unitamente alle centraline per la raccolta dei dati che saranno installate e configurate, in aggiunta ad un contratto di assistenza in caso di guasti hardware e software. Traffic Monitor individua come propri clienti le città che vogliono monitorare il traffico nelle proprie strade, a differenza di altre soluzioni simili già presenti sul mercato come Waze o Google Maps che basano il proprio business model sugli utenti.

Definizione dei Requisiti

Il sistema centrale, progettato per l'elaborazione di una grande mole di dati raccolti, analizzerà le informazioni sullo stato del traffico fornendole all'utente. La raccolta dei dati avverrà tramite molteplici tipologie di dispositivi, differenziando le modalità di acquisizione al fine di poter raccogliere dati coerenti. È necessario che utilizzi un metodo di correzione o prevenzione degli errori affinché questi influiscano poco sul risultato finale.

Per la comunicazione tra sistema e centraline l'utente potrà interrogare il sistema per visualizzare i dati tramite l'interfaccia grafica del sistema centrale e riceverà notifiche sullo stato del traffico in un'area circoscritta alla sua posizione qualora abbia installata l'applicazione mobile sul proprio smartphone.

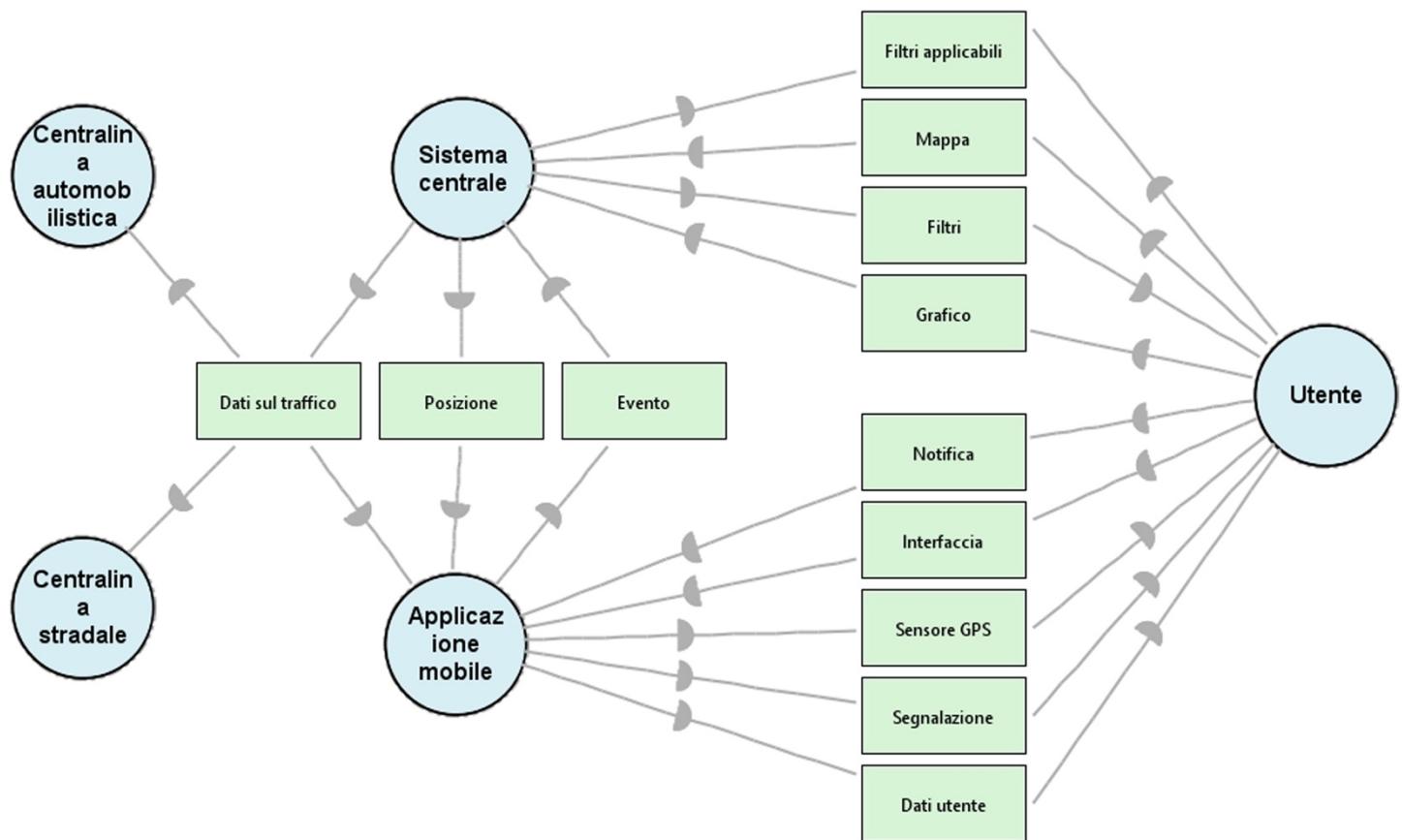
Analisi del rischio

Traffic Monitor sarà sviluppato interfacciandosi con centraline simulate via software, il che non comporterà al momento alcuna spesa per la realizzazione hardware, al fine di abbattere costi e complessità tecnologica circoscrivendola solamente all'apprendimento del linguaggio di programmazione Java.

Come indicato nelle specifiche, il progetto deve essere terminato entro il giorno 11 dicembre 2018. Un allungamento dei tempi di sviluppo per qualunque fattore porterebbe la mancata conclusione dell'elaborazione entro il giorno previsto, il che inficerebbe sulla buona riuscita dell'intero progetto.

SDM

Un insieme di nodi e link dove ogni nodo rappresenta un attore e ogni link fra due attori sta ad indicare che un attore dipende dall'altro per ottenere risorse che serviranno al primo a raggiungere il suo obiettivo.



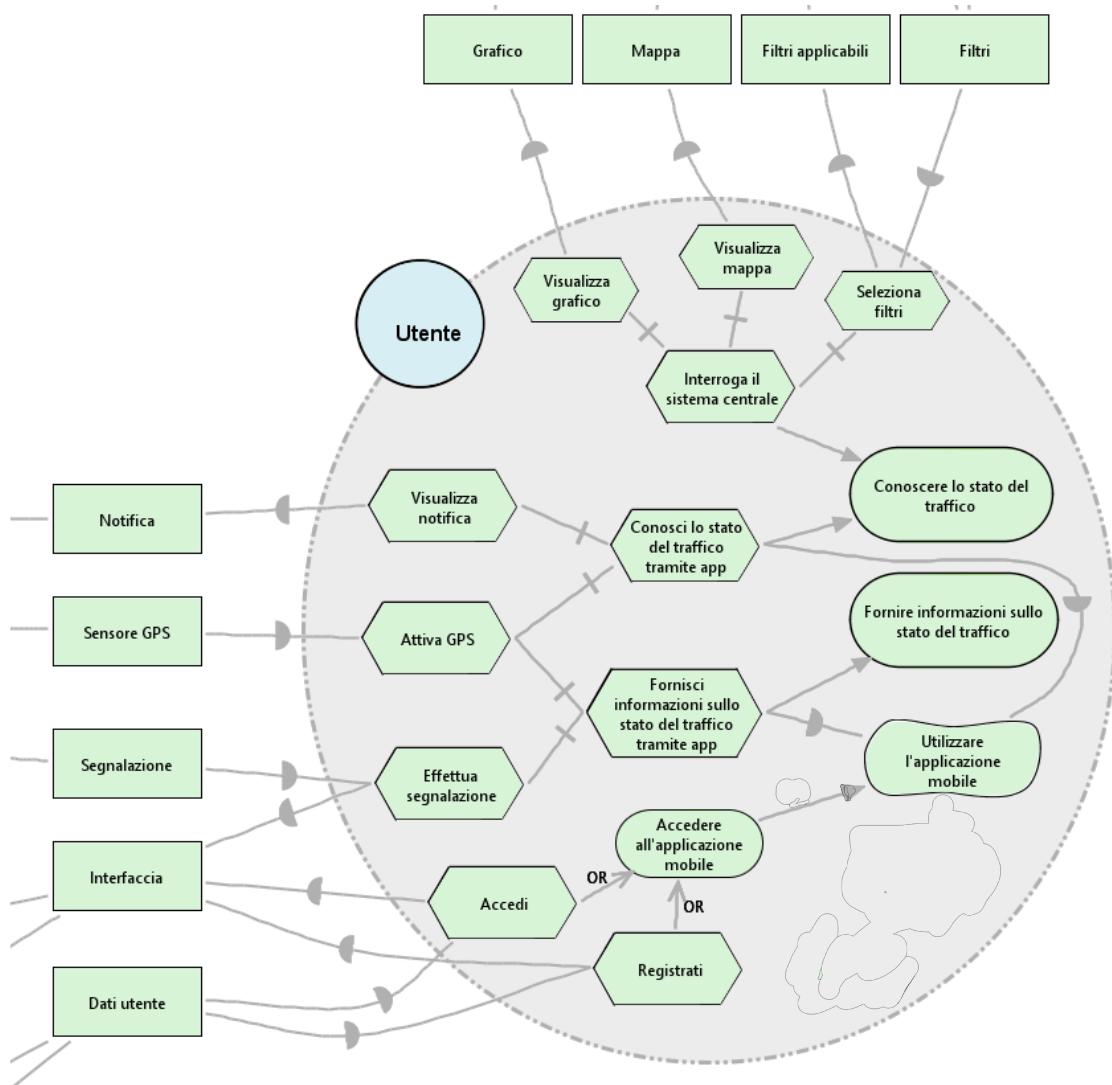
SINTESI DEI GOAL

SRM – Utente

L'utente presenta due principali goal: vuole conoscere lo stato del traffico e vuole contribuire nel fornire informazioni a riguardo.

Per il primo obiettivo, l'utente potrà interfacciarsi con il sistema centrale, interrogarlo per filtrare i dati che vedrà anche graficamente mediante mappe e/o grafici, e ricevere notifiche riguardo gli eventi dall'applicazione mobile con la quale potrà effettuare segnalazioni contribuendo quindi al secondo obiettivo.

L'utente, quindi, vuole poter utilizzare l'applicazione, per farlo è necessario che la scarichi sul proprio dispositivo, si registri e che vi acceda.



SRM – Applicazione Mobile

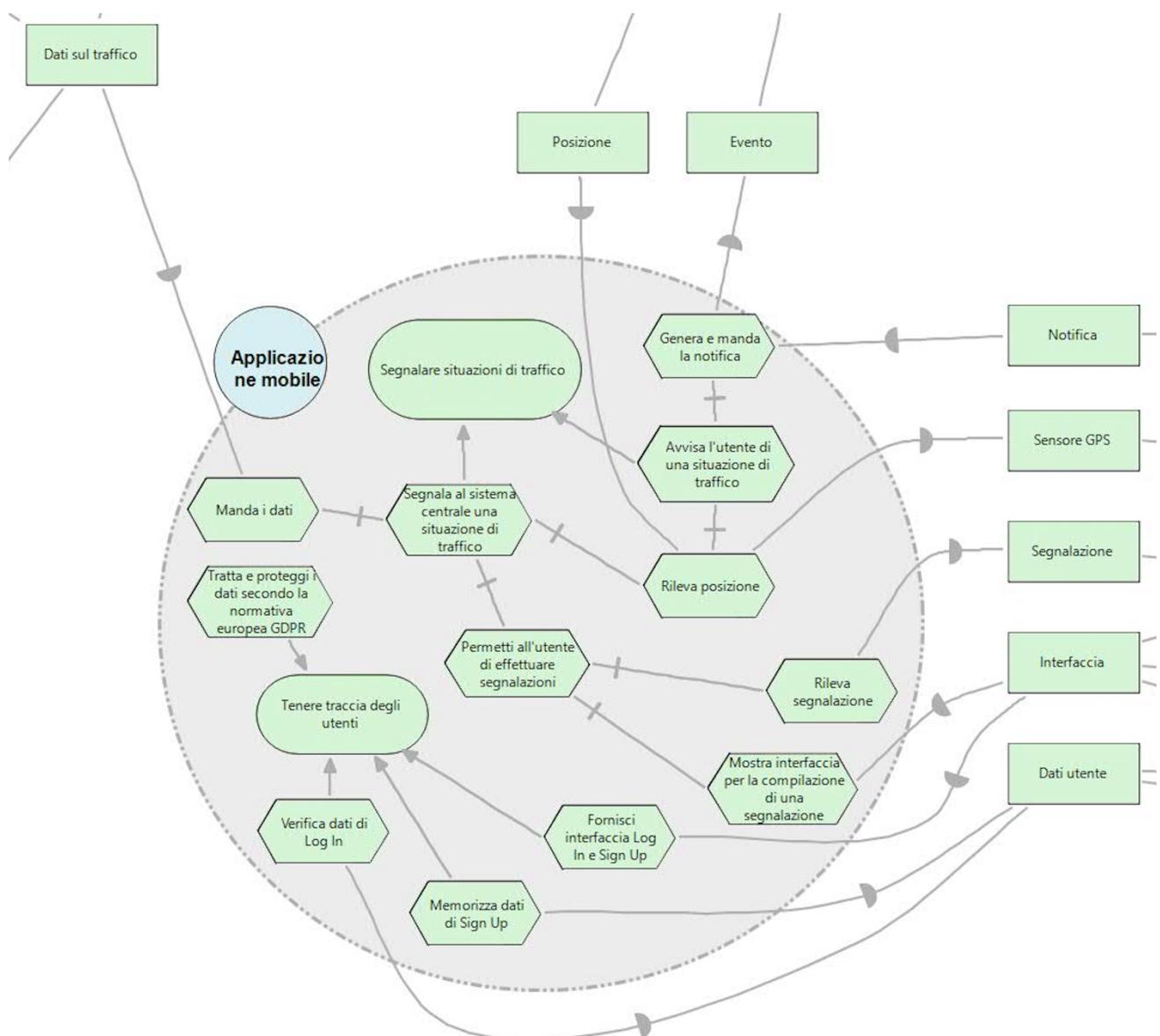
L'applicazione mobile ha come obiettivo principale la segnalazione dello stato del traffico, da parte sia dell'utente e sia del sistema centrale.

Le segnalazioni effettuate dall'utente saranno indirizzate al sistema centrale, compilando l'apposito form dall'interfaccia dell'applicazione mobile.

Le notifiche indirizzate all'utente, invece, risulteranno essere gli eventi rilevati dal sistema centrale che l'applicazione gli notificherà.

È fondamentale che venga rilevata tramite GPS la posizione del dispositivo utilizzato, al fine di mandare notifiche rilevanti e/o associare segnalazione con la via a cui si riferisce.

L'applicazione inoltre dovrà tenere traccia degli utenti fornendo un'interfaccia per il Log In ed il Sign Up, salvare i dati utente ad ogni nuova registrazione e verificare la correttezza delle credenziali di accesso. Il trattamento e la protezione dei dati avverrà rispettando la normativa europea GDPR.



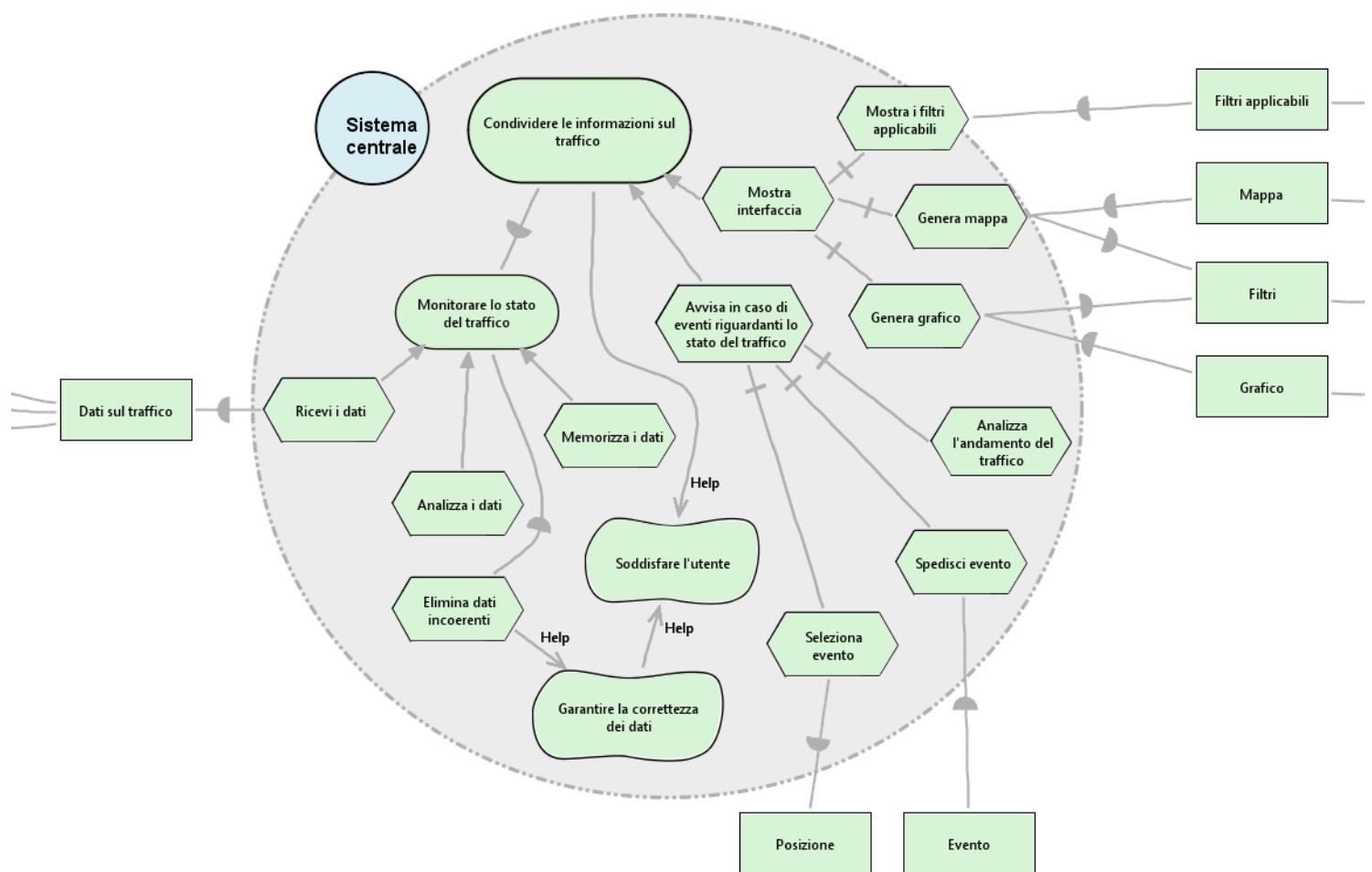
SRM – Sistema Centrale

L'obiettivo principale del sistema centrale è quello di fornire le informazioni riguardanti lo stato del traffico.

Verrà fornita all'utente un'interfaccia grafica per la visualizzazione dei dati, che consentirà di selezionare dei filtri ed eventualmente la mappa o un grafico. Il sistema centrale dovrà analizzare e confrontare i dati riguardanti il traffico raccolti da tutte le centraline ed eventuali segnalazioni, potendo segnalare all'utente lo stato del traffico e possibili variazioni o eventi (es. lavori in corso, incidenti).

Un ulteriore goal del sistema centrale sarà il monitoraggio dello stato del traffico: memorizzando, analizzando e confrontando i dati raccolti da tutte le centraline ed eventuali segnalazioni da parte degli utenti. Risulta evidente, quindi, che, per condividere le informazioni, il sistema dovrà prima gestirle lui stesso, motivo per il quale l'obiettivo principale dipende da questo appena definito.

Infine, il soft goal di garantire la correttezza dei dati, eliminando i dati incoerenti, cosicché gli utenti possano fare affidamento sulle informazioni ricevute e dunque siano soddisfatti del servizio offerto da Traffic Monitor.

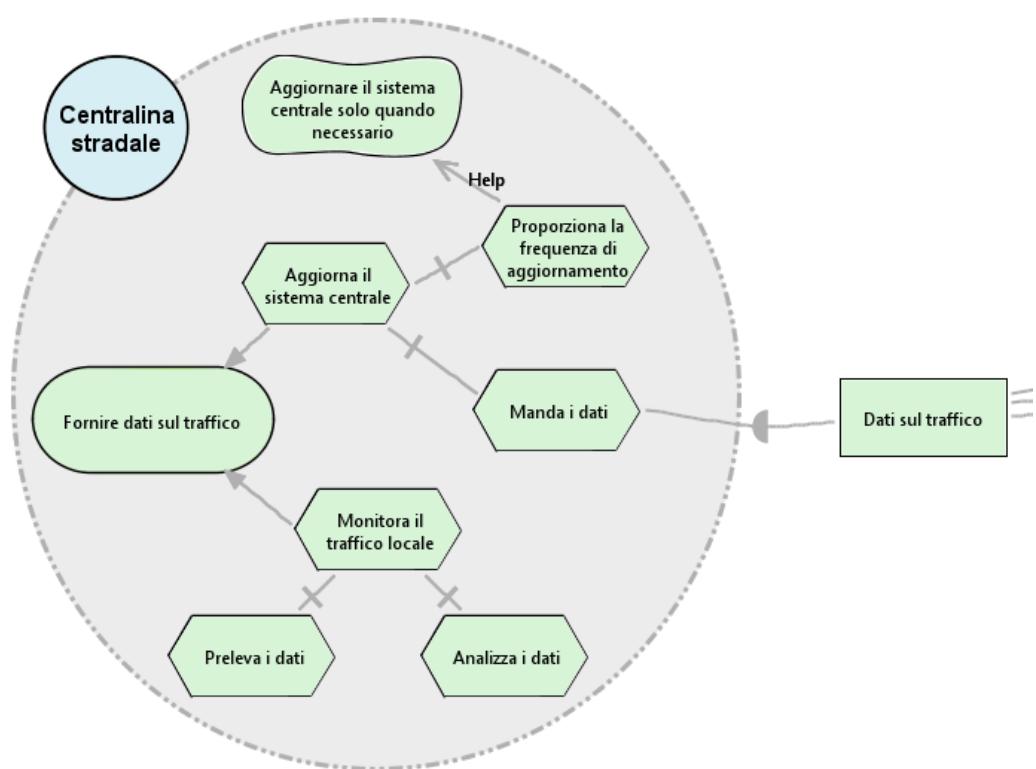


SRM – Centralina Stradale

La centralina stradale deve fornire al sistema centrale i dati del traffico che preleva.

Il suo goal principale è il monitoraggio del traffico nel tratto stradale in cui è posizionata e l'aggiornamento del sistema centrale con periodicità proporzionale all'ammontare del traffico.

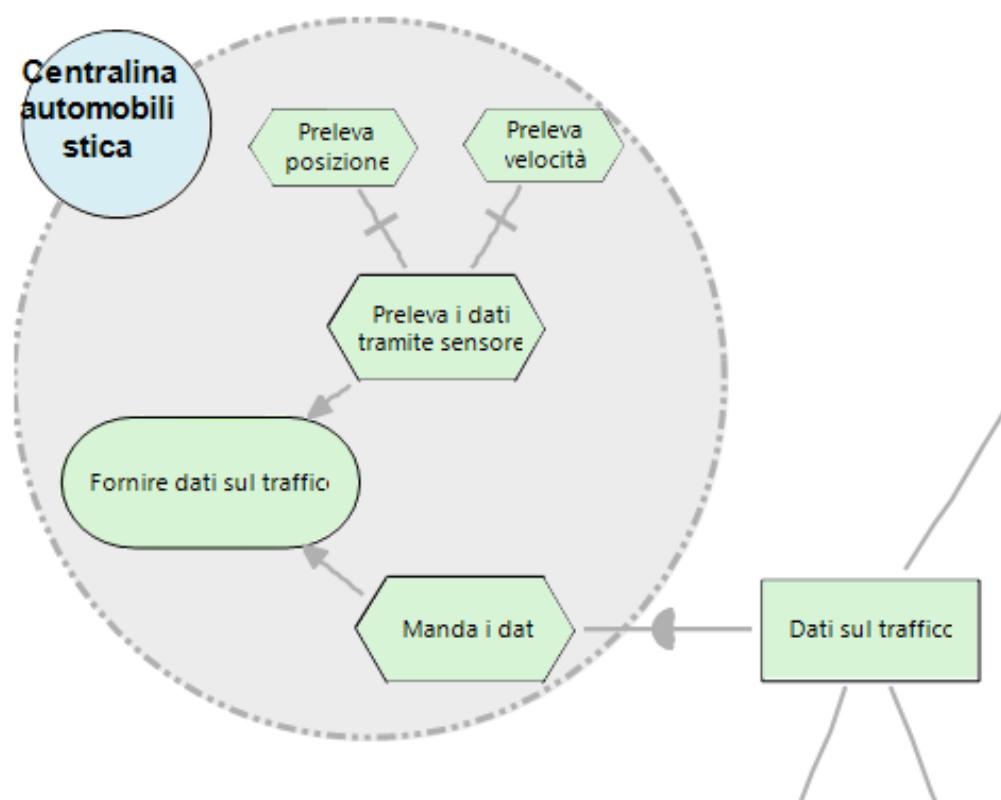
Per soddisfare l'obiettivo, la centralina stradale dovrà prima di tutto prelevare i dati dai sensori, per poi analizzarli ricavandone lo stato del traffico e spedirli. È necessario, inoltre, che la frequenza di aggiornamento con la quale vengono inviati i dati al sistema centrale venga proporzionale in funzione del traffico affinché questi non venga aggiornato inutilmente a fronte di situazioni normali.



SRM – Centralina Automobilistica

Così come la centralina stradale, la centralina automobilistica deve fornire i dati sul traffico basandosi sulla velocità media nel periodo di rilevazione e sulla posizione della vettura sulla quale è installato.

L'obiettivo viene raggiunto prelevando i dati specificati dalla centralina automobilistica quando la vettura è in moto, ed aggiornando a intervalli di tempo costante il sistema centrale.



DATA DICTIONARY

Il data dictionary contiene la definizione e le informazioni riguardo i termini principali utilizzati nel progetto.

NOME	DESCRIZIONE	SINONIMI	ESEMPI	SOTTOTIPI	SUPERTIPI	ATTRIBUTI	COMPONENTI	RELAZIONI
Utente	Persona fisica che consulta le informazioni raccolte dal sistema centrale ed interagisce con l'applicazione mobile dalla quale manda segnalazioni e riceve notifiche.	Persona	<ul style="list-style-type: none"> - Keanu - keanu.reeves@mail.com - ***** - Via Valleggio 			<ul style="list-style-type: none"> - Username - Email - Password - Posizione (via) 		Notifica, Segnalazione, Sistema Centrale, Applicazione mobile
Notifica	Comunicazione mandata dal sistema centrale all'utente, che contiene informazioni sullo stato del traffico nella posizione in cui si trova l'utente.	Avviso, Messaggio				<ul style="list-style-type: none"> - Via - Stato del traffico 	Evento	Applicazione mobile, Utente
Sistema Centrale	Software che raccoglie, gestisce e mostra le informazioni riguardo lo stato del traffico in tutti i tratti di strada (vie) di cui ha ricevuto i dati.	Software	Traffic Monitor.exe			<ul style="list-style-type: none"> - Vie - Limiti di velocità - Velocità raccolte - Stati del traffico - timestamp 		Utente, Applicazione Mobile, Centralina Stradale, Centralina Automobilistica, Segnalazione
Centralina Stradale	Dispositivo adibito alla raccolta delle informazioni sulla velocità e densità del traffico nel tratto di strada (via) in cui è posizionato.	Sensore Stradale	Centralina Via Valleggio 			<ul style="list-style-type: none"> - Via - Limite velocità - Velocità raccolta - Stato del traffico 	Sensori	Sistema Centrale
Centralina Automobilistica	Dispositivo adibito alla raccolta di informazioni sulla velocità e sulla posizione della vettura sulla quale è collocato.	Sensore Mobile	Centralina Automobile 1 			<ul style="list-style-type: none"> - Posizione (via) - Velocità 		Sistema Centrale
Stato del traffico	Livello di scorrevolezza di una via (in percentuale) rispetto al limite di velocità stabilito.	Livello di rallentamento	Traffico Bloccato					Via, Sistema Centrale, Notifica, Segnalazione, Applicazione Mobile, Centralina stradale, Centralina automobilistica
Mappa	Reticolo stradale contenente le vie sottoposte ad analisi. Idealisticamente dovrebbe trattarsi di una città.	Reticolo stradale					Via	
Strada	Infrastruttura di trasporto destinata a veicoli terrestri sulla quale vengono effettuate le rilevazioni.	Infrastruttura						
Segmento	Sezione di una via che si estende da un incrocio (o dal punto iniziale della via) all'incrocio successivo (o punto finale della via). Identifica la posizione delle applicazioni mobili.	Posizione	01 Via Valleggio		Via	<ul style="list-style-type: none"> - Segmenti adiacenti - Lunghezza 		Via, Applicazione Mobile, Sistema Centrale
Via	Sezione di strada identificata da un inizio, una fine, ed un limite di velocità	Sezione di strada	Via Valleggio	Segmento		<ul style="list-style-type: none"> - Nome - Limite di velocità - Tipologia 		Segmento, Sistema Centrale, Notifica, Segnalazione, Applicazione Mobile, Centralina stradale, Centralina automobilistica
Applicazione mobile	Applicazione destinata alla notifica degli eventi ed alla compilazione ad parte degli utenti di segnalazioni riguardanti il traffico.	Software, Programma	Traffic Monitor Mobile.apk			<ul style="list-style-type: none"> - Utenti - Posizione 		Utente, Sistema Centrale, Notifica, Segnalazione
Segnalazione	Aggiornamento riguardante lo stato del traffico effettuato dall'utente per il sistema centrale tramite l'applicazione mobile.	Trasmissione	<ul style="list-style-type: none"> - Via Valleggio - Coda scorrevole 			<ul style="list-style-type: none"> - Posizione (via) - Stato del traffico 		Applicazione Mobile, Utente

DESIGN

USE CASE DIAGRAM

Lo USE CASE DIAGRAM descrive le funzioni e i servizi offerti dal sistema e dall'applicazione all'Utente.

L'utente, tramite il sistema, vuole visualizzare i dati del traffico in vari formati come: mappa, diagramma e testuale; di cui può selezionare filtri per ottenere i dati desiderati.

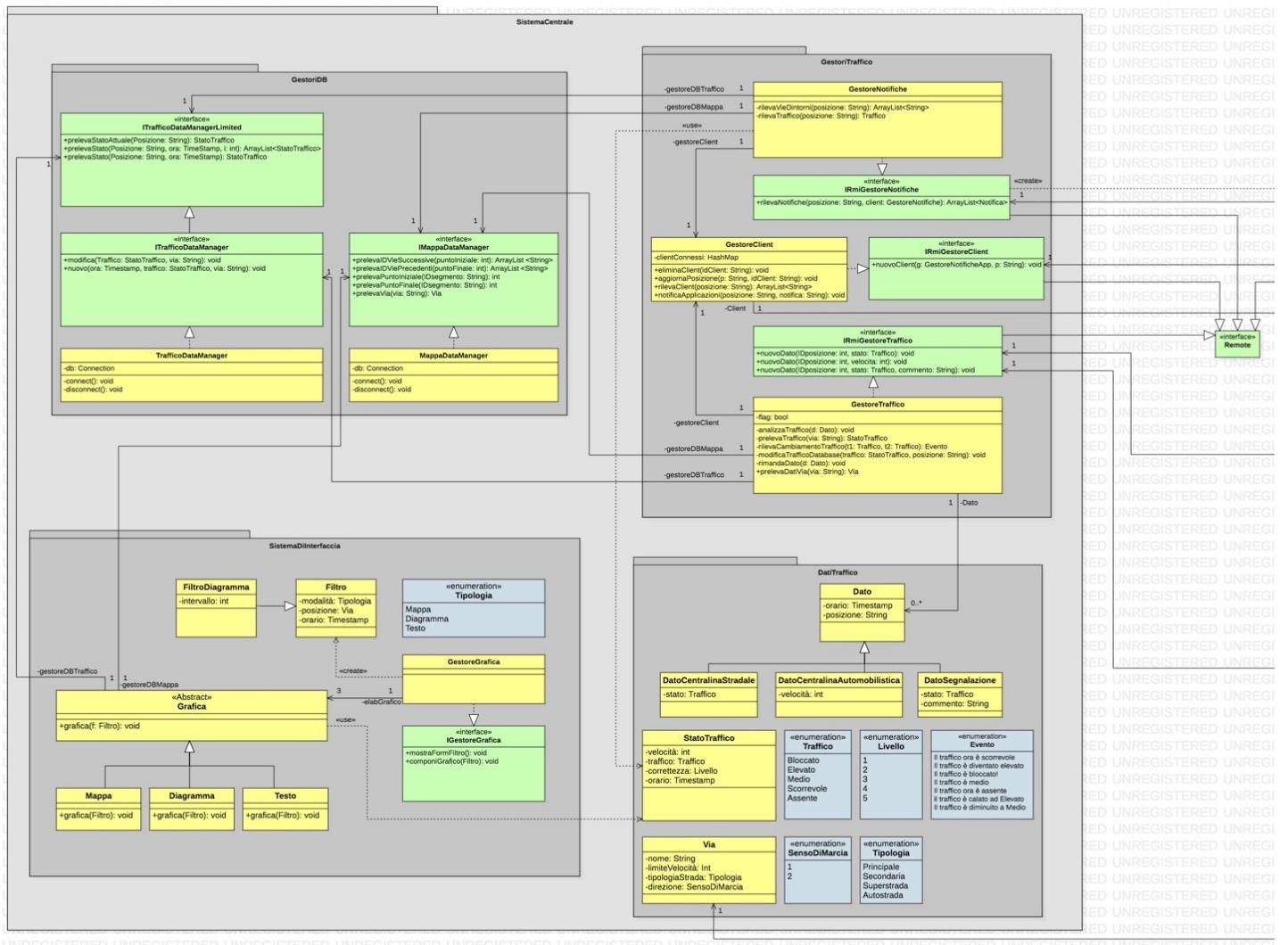
Invece, tramite l'applicazione, può ricevere le notifiche per conoscere lo stato del traffico in tempo reale, e mandare al sistema segnalazioni sullo stato del traffico, contribuendo a sua volta al monitoraggio.



CLASS DIAGRAM

Il CLASS DIAGRAM è stato scomposto in tre principali pacchetti; SistemaCentrale, ApplicazioneMobile e Centralina. Questa suddivisione consente una migliore visione d'insieme del diagramma e ne facilita la comprensione della funzionalità delle classi.

SISTEMA CENTRALE



TrafficDataManager: questa classe è adibita alla comunicazione con il database per la scrittura e la lettura dei dati riguardanti lo stato del traffico. L'interfaccia che realizza risulta essere la generalizzazione di una seconda, in quanto, delle tre classi utilizzatrici dei suoi metodi, solo GestoreTraffic necessiterà della possibilità di modificare i dati del database (l'interfaccia Limited presenta solo metodi per la lettura dei dati).

MappaDataManager: anche questa classe è adibita alla gestione del database, ma su una tabella differente rispetto a TrafficoDataManager. Il compito di MappaDataManager, difatti, è la sola lettura dei dati statici riguardanti le vie, come la loro struttura (nome, limite di velocità, tipologia di strada e senso di marcia) e/o le loro intersezioni.

GestoreNotifiche: questa classe è il principale mezzo di comunicazione tra sistema centrale ed applicazione mobile. Quando rileva un cambiamento di via (o segmento di via), chiama, tramite RMI, il metodo rilevaNotifiche, il quale preleverà eventuali situazioni di traffico nelle vie adiacenti dal database tramite TrafficoDataManager e MappaDataManager. Prima di comunicare all'applicazione (o meglio al GestoreNotifiche) le notifiche rilevate, tramite GestoreClient l'ultima posizione rilevata riferita a quel client.

GestoreClient: questa classe gestore monitora le applicazioni “viventi” associandole ad una posizione (HashMap). Nel momento in cui l'applicazione si “sveglia” (quando l'utente ha effettuato l'autenticazione), contatta l'istanza di GestoreClient tramite interfaccia RMI per permettergli di salvare un nuovo client.

GestoreTraffico: è la classe principale del progetto. GestoreTraffico preleva i dati spediti dalle centraline e/o dalle applicazioni per elaborarli in informazioni riguardanti lo stato del traffico. L'attributo flag è stato pensato per evitare che, tramite processi paralleli, più dati riguardanti la stessa via vengano elaborati contemporaneamente. Quando un dato troverà il flag attivo, verrà accodato ad una lista. GestoreTraffico potrà anche comunicare con l'applicazione tramite GestoreClient (chiamando il metodo notificaApplicazioni) per informarla dei cambiamenti nello stato del traffico (situazione ipotetica in cui l'applicazione si trova in una via ed ha già eseguito richiesta al sistema centrale per lo stato del traffico nei dintorni; un eventuale cambiamento improvviso le rimarrebbe sconosciuto se questo procedimento non fosse stato ideato).

Filtro(FiltroDiagramma): è la classe contenente i dati riguardanti le informazioni su quali dati l'utente vuole conoscere e come li vuole visualizzare (tipologia di grafico).

GestoreGrafica: è il gestore dell'interfaccia per l'utente. Si occupa di costruire il filtro e di chiamare il metodo di costruzione grafica richiesto.

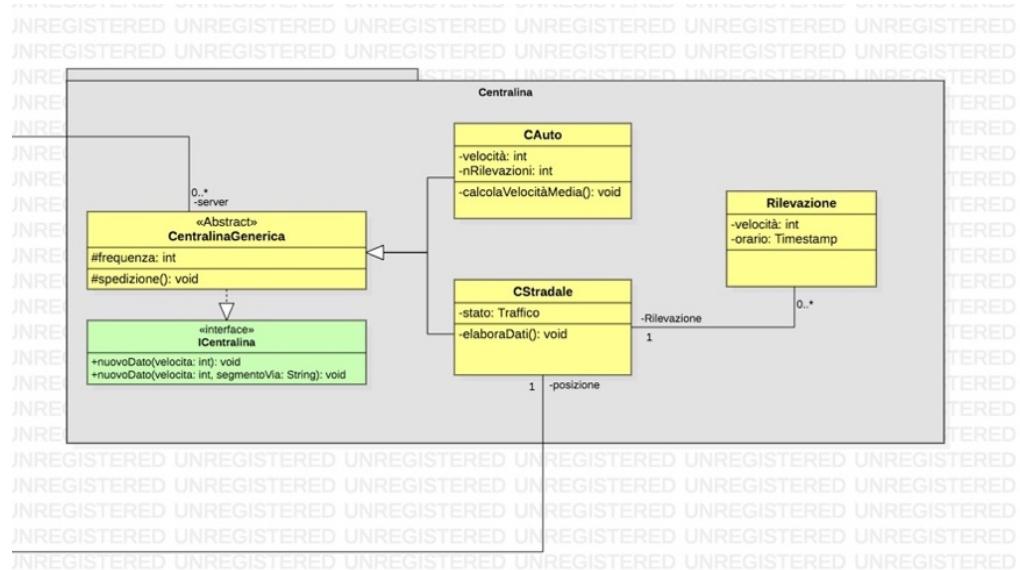
Grafica(Mappa, Diagramma, Testo): è la classe adibita alla costruzione del grafico richiesto. Per farlo potrà accedere ai metodi di lettura (Limited) di TrafficoDataManager ed a MappaDataManager. È strutturata come uno Strategy Pattern.

Dato(DatoCentralinaAutomobilistica, DatoCentralinaStradale, DatoSegnalazione): è la classe contenente le informazioni riguardanti le segnalazioni e le rilevazioni comunicate al sistema centrale.

StatoTraffic: questa classe rappresenta lo stato del traffico associato ad una via in un determinato momento della giornata.

Via: contiene le informazioni precedentemente specificate riguardo una via.

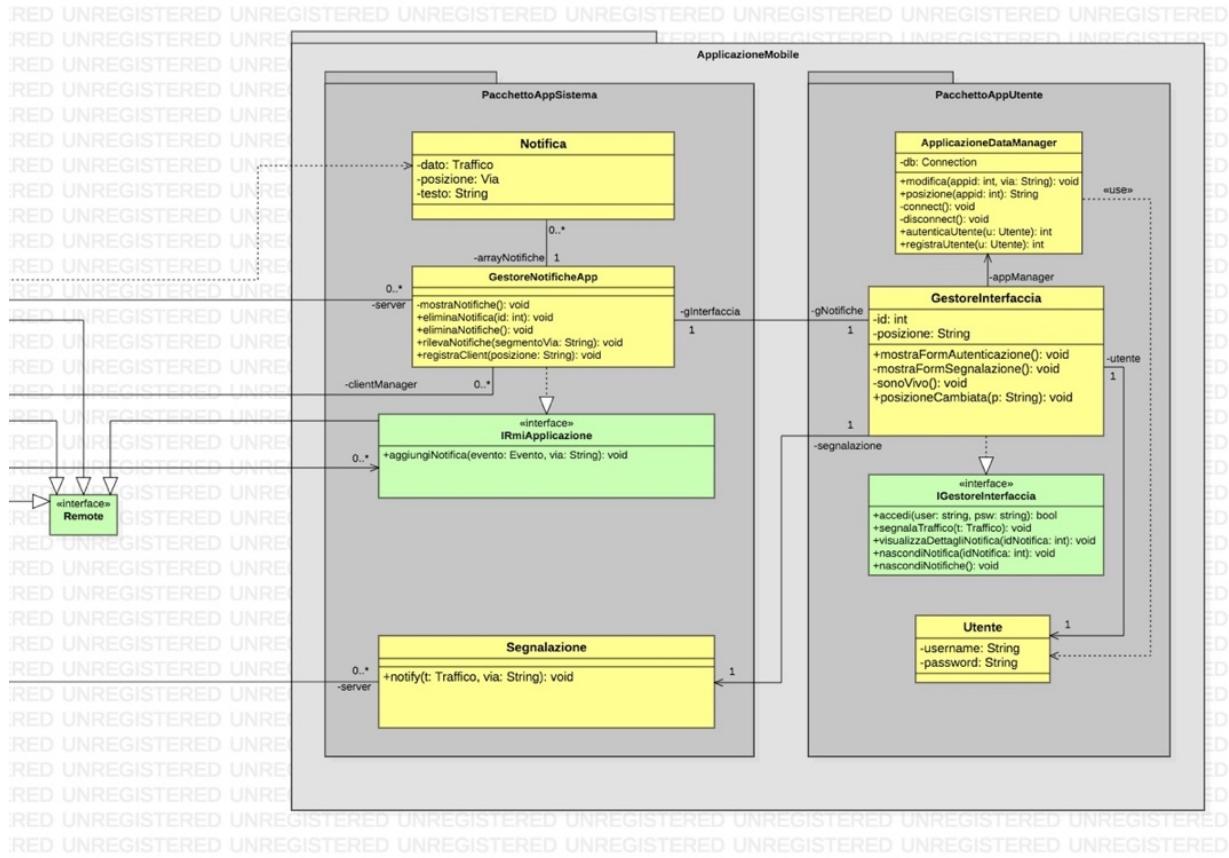
CENTRALINA



Centralina(CAuto, CStradale): è la classe principale per le centraline (per le centraline automobilistiche è anche l'unica). Ha il compito di svolgere calcoli e di comunicarne i risultati al sistema centrale (il metodo spedizione, tramite RMI, chiama nuovoDato di GestoreTraffic).

Rilevazione: contiene una rilevazione effettuata dalla centralina stradale.

APPLICAZIONE MOBILE



GestoreInterfaccia: questa classe consente l’interfaccia tra utente ed applicazione. Quando l’utente ha effettuato l’interimento dei dati per l’autenticazione, GestoreInterfaccia li preleva e chiama ApplicazioneDataManager per la verifica. Presenta la stessa funzionalità per l’elaborazione di una segnalazione e la visualizzazione delle notifiche. Inoltre monitora la posizione dell’applicazione al fine di “avvisare” GestoreNotifiche di un eventuale cambio di via (segmento) e memorizzare tale dato nel database (tramite ApplicazioneDataManager).

ApplicazioneDataManager: è la classe adibita all’utilizzo del database e, di conseguenza, di verificare o registrare gli utenti e prelevare e/o modificare la posizione associata all’utente.

Utente: è la classe contenente i dati relativi all’utente

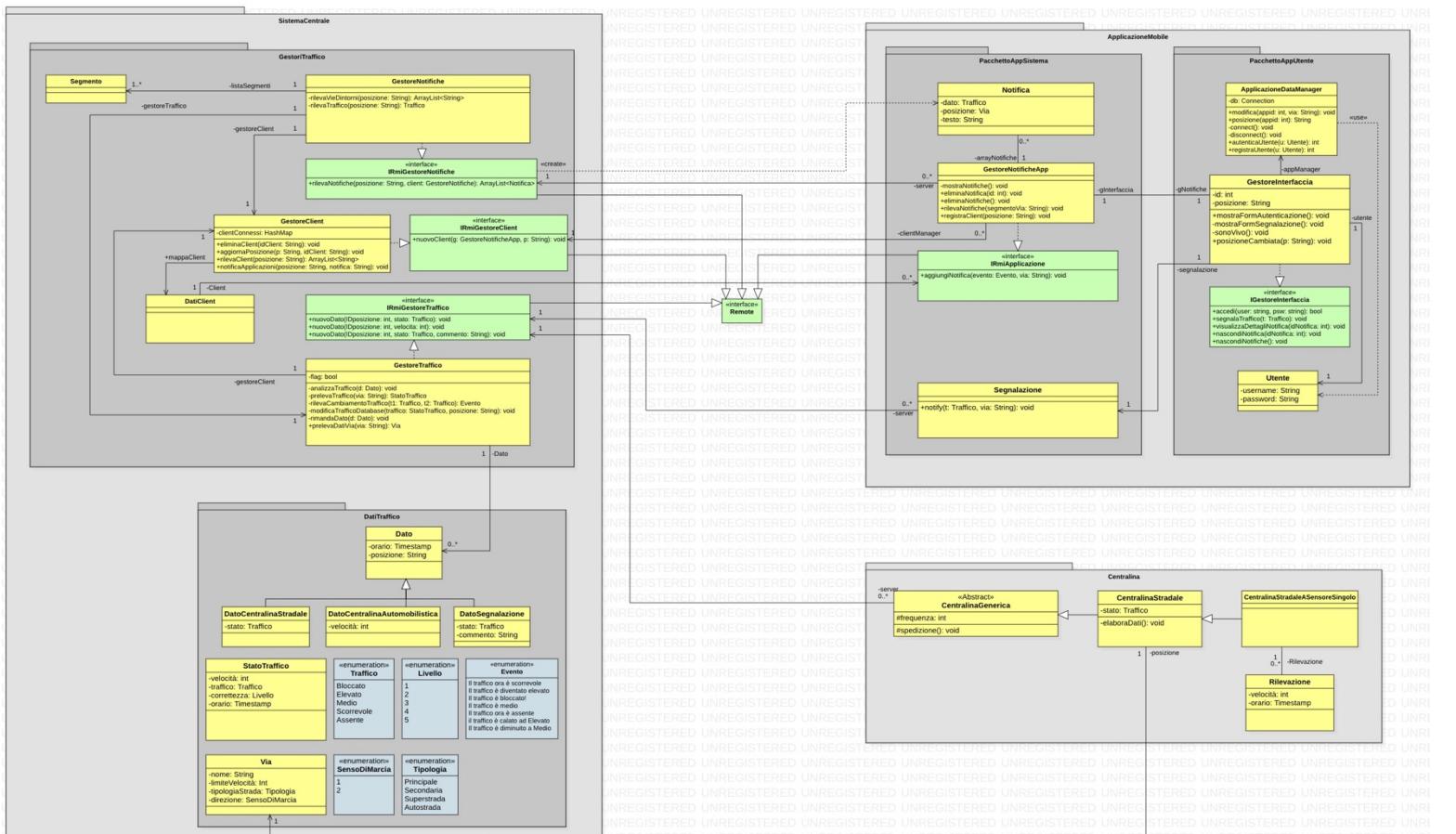
Segnalazione: questa classe è adibita alla comunicazione con il sistema centrale (tramite RMI) per la spedizione dei dati posizione (rilevata dall'applicazione) e traffico (generato dall'utente).

Notifica: è la classe contenitore per le informazioni riguardo lo stato del traffico particolare nelle vie circostanti.

GestoreNotificheApp: è la classe destinata alla richiesta ed alla ricevuta delle notifiche. Ha anche il compito, ad autenticazione effettuata, di comunicare a GestoreClient il suo riferimento e la sua posizione, affinché questi possano essere memorizzati nella HashMap.

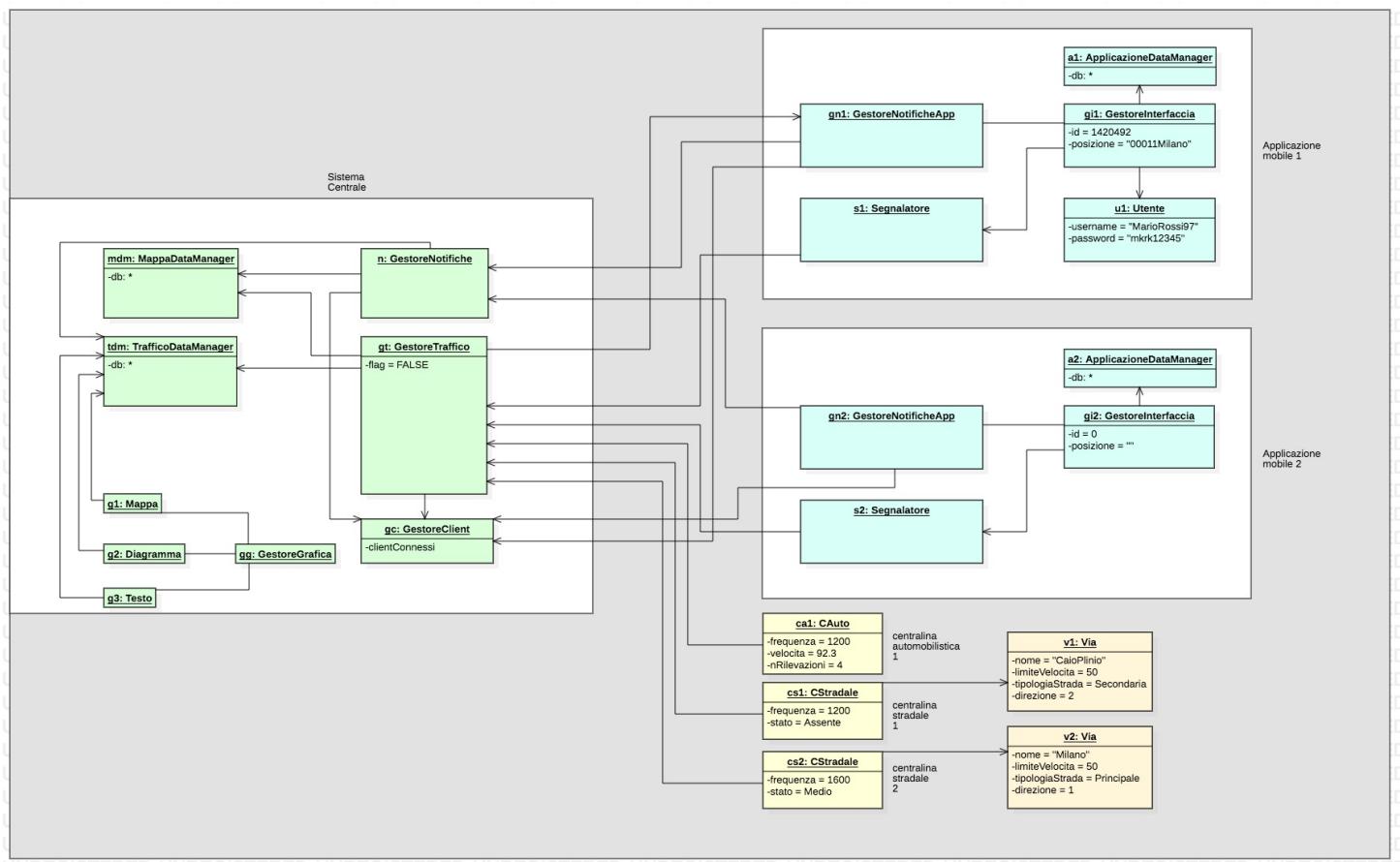
MODIFICHE CLASS DIAGRAM POST JAVA

Il class diagram ha subito marginali rielaborazioni in fase di sviluppo. Ciò è stato causato dal fatto che la precedente stesura fu sviluppata senza una conoscenza approfondita del linguaggio Java e, di conseguenza, alcune relazioni e classi non sono state elaborate in maniera ottimale e funzionale. Come è possibile osservare, sono state rimosse completamente i package GestoriDB e GestoriInterfaccia. Questa decisione è stata intrapresa in quanto la grafica (l'interfaccia per l'utente) non necessitava di una diretta classe di interfaccia ai dati, in quanto già stata implementata per accedere alle rilevazioni dell'istanza di GestoreTraffico. Le classi per la gestione del database, invece, sono state rimosse in quanto il database stesso non è stato sviluppato. Questa decisione è stata presa dal momento che, dopo un'attenta analisi, è stato notato il basso livello di utilità di una sua eventuale implementazione; le informazioni delle vie vengono memorizzate in un file che verrà letto una sola volta all'accensione del sistema centrale (inutile quindi strutturare un database per accedervi ipoteticamente solo una volta), mentre, avendo deciso di memorizzare solo le informazioni attuali riguardanti lo stato del traffico, il database per il contenimento delle rilevazioni perde la sua utilità tenendo già l'istanza di GestoreTraffico traccia degli stati del traffico attuali.



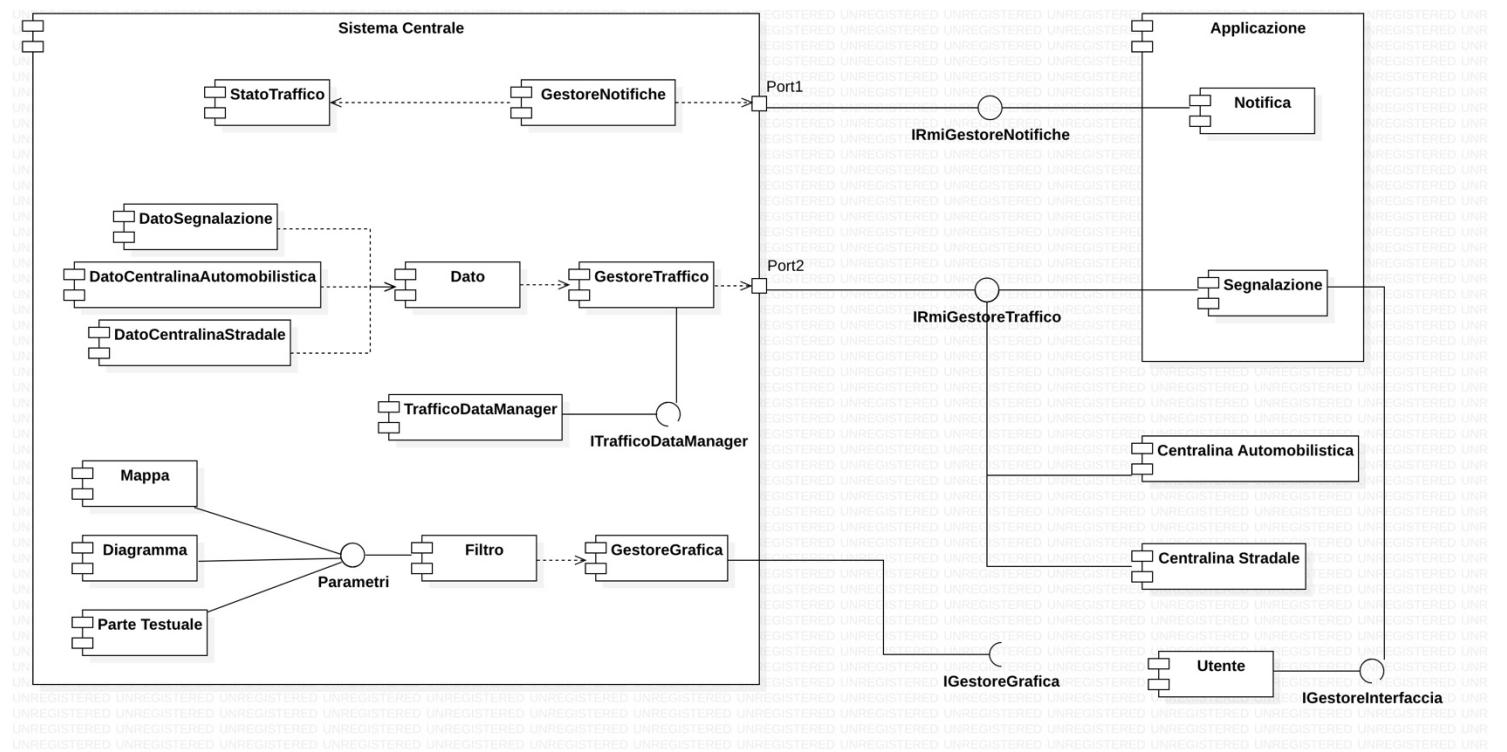
OBJECT DIAGRAM

L'OBJECT DIAGRAM riportato rappresenta la situazione statica delle istanze in un momento prossimo all'avvio del sistema, con le centraline che ancora non hanno rilevato alcun dato e le applicazioni sulle quali nella prima è già stata eseguita l'autenticazione dell'utente e sulla seconda l'utente ancora deve inserire le sue credenziali (quindi l'applicazione non ha ancora comunicato al sistema centrale la sua attività, motivo per il quale sono presenti solo le connessioni applicazione/sistema e non sistema/applicazione).



COMPONENT DIAGRAM

Il COMPONENT DIAGRAM rappresenta la struttura interna del software del Sistema Centrale e la sua comunicazione con l'applicazione e le centraline. Sono mostrati i suoi componenti principali e le relazioni tra di loro:



DEPLOYMENT DIAGRAM

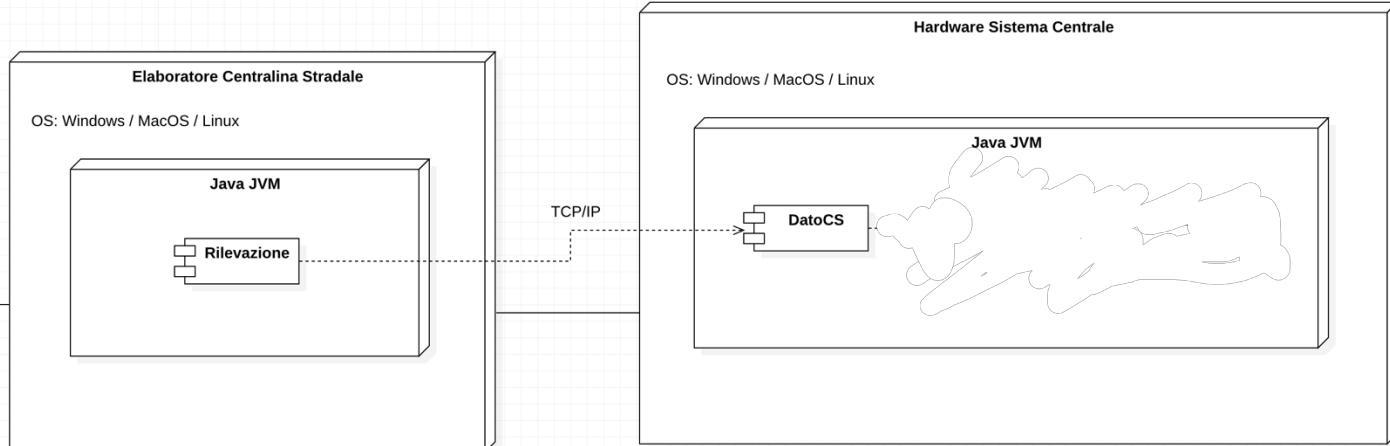
Il DEPLOYMENT DIAGRAM descrive la struttura della Centralina Stradale in termini di risorse hardware.

I nodi presenti nel diagramma sono: il sensore che monitora un tratto di strada, raccogliendo dati sulla velocità media di ogni singola macchina e l'hardware della centralina che si occupa di analizzare lo stato del traffico per adattare i tempi di invio dei dati, secondo l'intensità di traffico.

I dati infine vengono mandati al sistema centrale anch'esso rappresentato con un nodo.

Il software viene eseguito sulla Java Virtual Machine (JVM) rappresentata da un nodo interno all'hardware e per completezza nel diagramma sono presenti i componenti software più rilevanti dato che spesso il component diagram e il deployment diagram sono rappresentati contestualmente.

COMUNICAZIONE TRA
CENTRALINA STRADALE
E SISTEMA CENTRALE

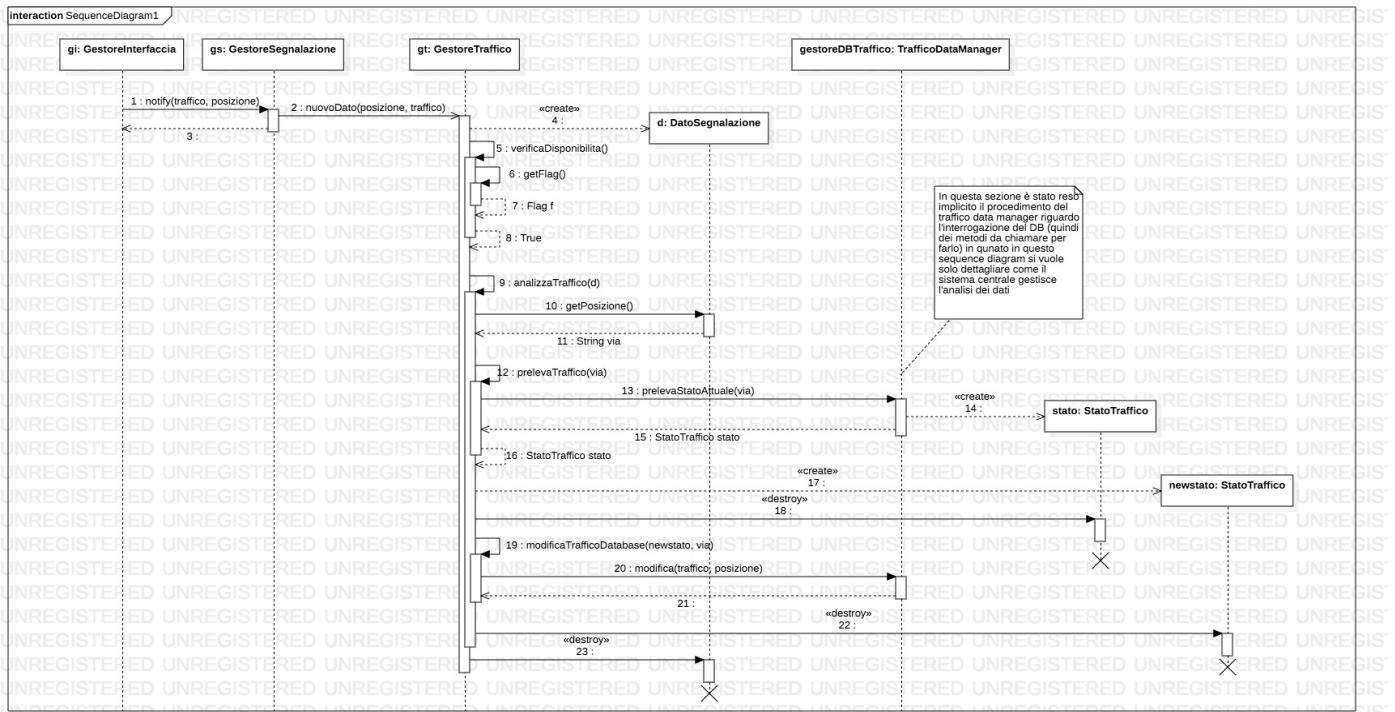


SEQUENCE DIAGRAMS

Il SEQUENCE DIAGRAM descrive uno scenario, quest'ultimo è composto da una successione ordinata di azioni definite da scelte già effettuate.

Come da specifiche sono stati descritti due scenari;

- Il sistema centrale riceve una segnalazione e la elabora:



1,2) gi, tramite il gs, comunica a gt (istanziata nel sistema centrale) la segnalazione dell'utente.

3) la segnalazione dell'utente viene memorizzata in un'istanza della classe DatoSegnalazione.

4,5,6,7) gt verifica, tramite l'analisi dello stato di flag (semaforo per segnalare se il sistema sta già elaborando un dato), seleziona il procedimento da svolgere.

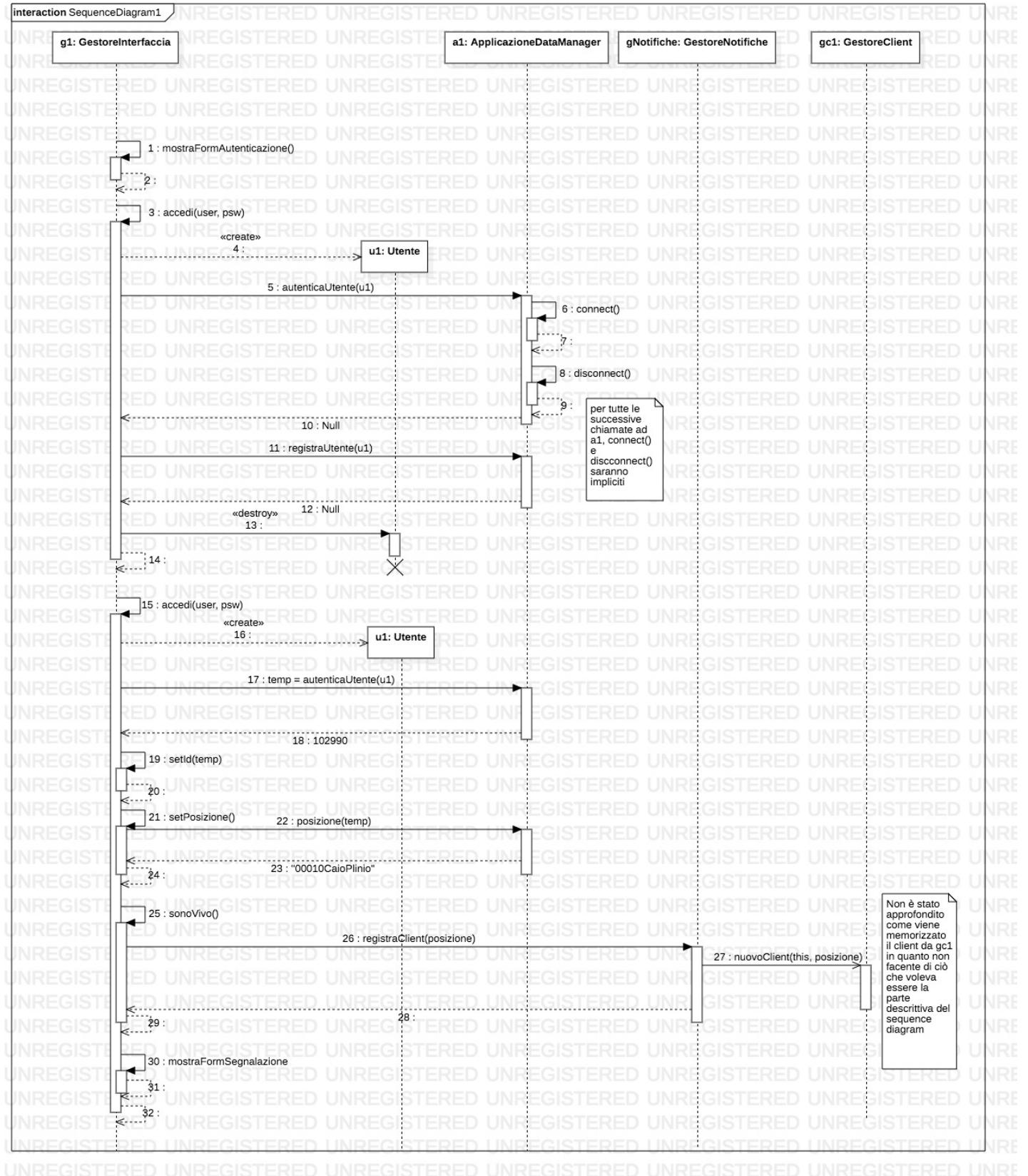
8) essendo il flag libero (true), gt inizia l'analisi del traffico.

9,10,11,12,13,14,15) gt, tramite gestoreDBTraffic, dopo aver estratto la posizione dall'istanza di DatoSegnalazione, preleva dal database lo stato del traffico precedentemente elaborato e ne crea un'istanza.

16) gt elabora un nuovo stato del traffico confrontando il precedente con il dato ricevuto.

17,18,19,20) gt distrugge l'istanza del precedente stato del traffico, accede al database tramite gestoreDBTraffic, memorizza il nuovo stato del traffico per poi distruggere anche la sua istanza.

- L'utente effettua l'autenticazione, sbaglia, ed effettua nuovamente l'autenticazione:

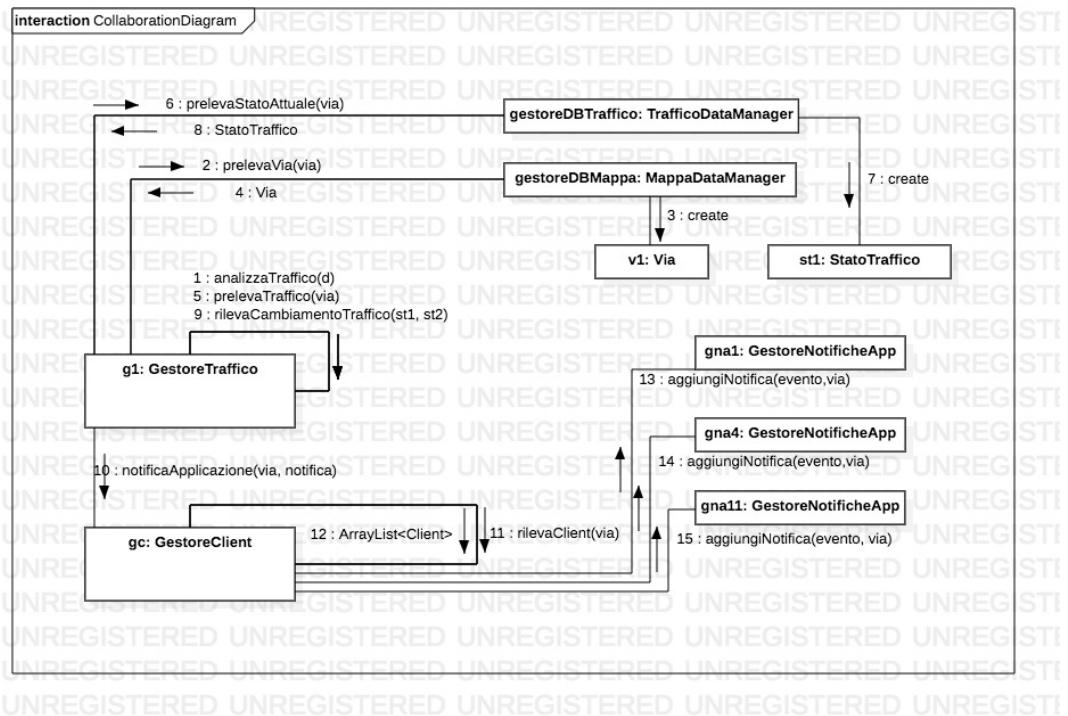


- 1,2) g1 chiama il metodo per l'elaborazione dell'interfaccia utente e questi,
- 3) l'utente preme il bottone per eseguire l'autenticazione e viene chiamato il metodo accedi
- 4,5,6,7,8,9,10) g1 istanzia u1 come Utente e, tramite a1, verifica la presenza di tale utente su database. autenticaUtente ritorna un valore nullo a causa del fatto che l'utente aveva sbagliato la password o non era registrato.
- 11,12) essendo che l'utente poteva non era registrato, g1 ne richiede la registrazione. Il metodo registraUtente ritorna un valore nullo, ovvero l'utente ha sbagliato la password.
- 13,14) l'istanza di utente viene distrutta e termina il metodo accedi, riportando g1 in stato di attesa.
- 15,16,17,18) l'utente tenta di nuovo l'accesso e questa volta va a buon fine.
- 20,21,22,23,24) vengono settati gli attributi di g1 (la posizione viene settata in base all'ultima posizione memorizzata sul database per il rispettivo utente).
- 25,26,27,28,29) g1, tramite l'istanza di GestoreNotificheApp, aggiunge un suo riferimento al sistema centrale (nello specifico all'istanza di GestoreClient nel sistema centrale)
- 30,31,32) viene generato il form per la compilazione della segnalazione e l'applicazione torna in stato di attesa.

COLLABORATION DIAGRAM

Il COLLABORATION DIAGRAM descrive come interagiscono gli oggetti collegati tramite associazioni, mostrando i messaggi che si inviano per interagire tra di loro.

Il diagramma mostra la notifica alle applicazioni, mandata dal sistema centrale, quando ha rilevato un cambiamento dello stato del traffico una volta analizzati i dati.



STATE DIAGRAMS

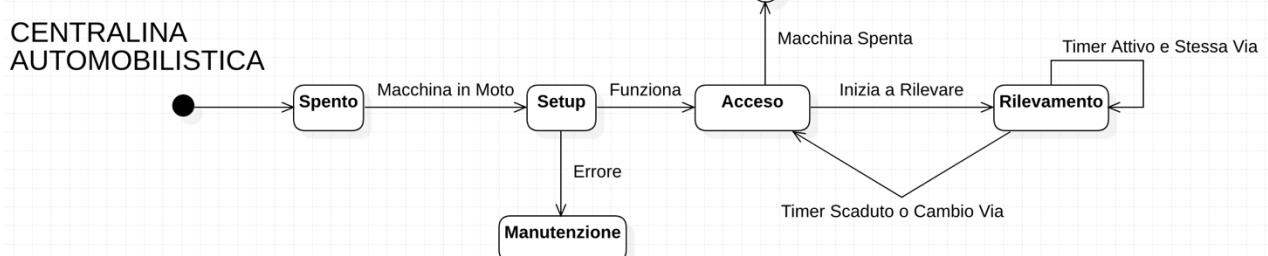
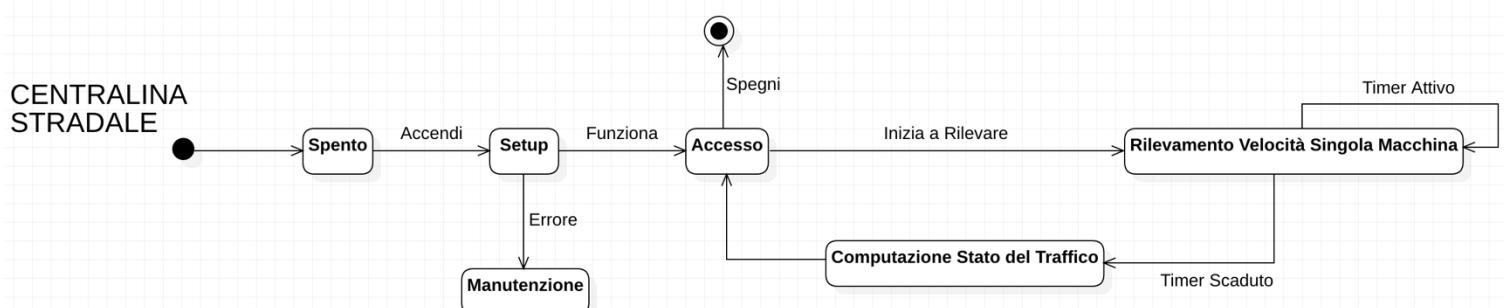
Lo STATE DIAGRAM descrive gli stati assunti dalla classe in relazione a determinati eventi, partendo dallo stato iniziale fino ad uno stato finale.

Come da specifiche sono stati descritti due diagrammi:

Condividono invece la parte iniziale in cui si descrivono gli stati che passano da spento a setup, da questo stato se vi è qualche errore la centralina entra nello stato di manutenzione in attesa di essere sistemata.

Invece se funziona regolarmente passa allo stato di acceso da cui si differenziano i due diagrammi.

- 1) **La Centralina Stradale** passa allo stato di rilevamento della velocità di ogni singola macchina continuando in quello stato fino a che il periodo di rilevamento gestito dal timer non scade, dopodiché passa allo stato di computazione dei rilevamenti al fine di modificare la periodicità dell'invio dei dati al sistema centrale che avverrà nello stato di acceso.
- 2) **La Centralina Automobilistica** è in stato di rilevamento della velocità della macchina su cui è installata finché il timer è attivo e sta percorrendo la stessa via, poi va nello stato di acceso in cui invia i dati raccolti al sistema centrale.



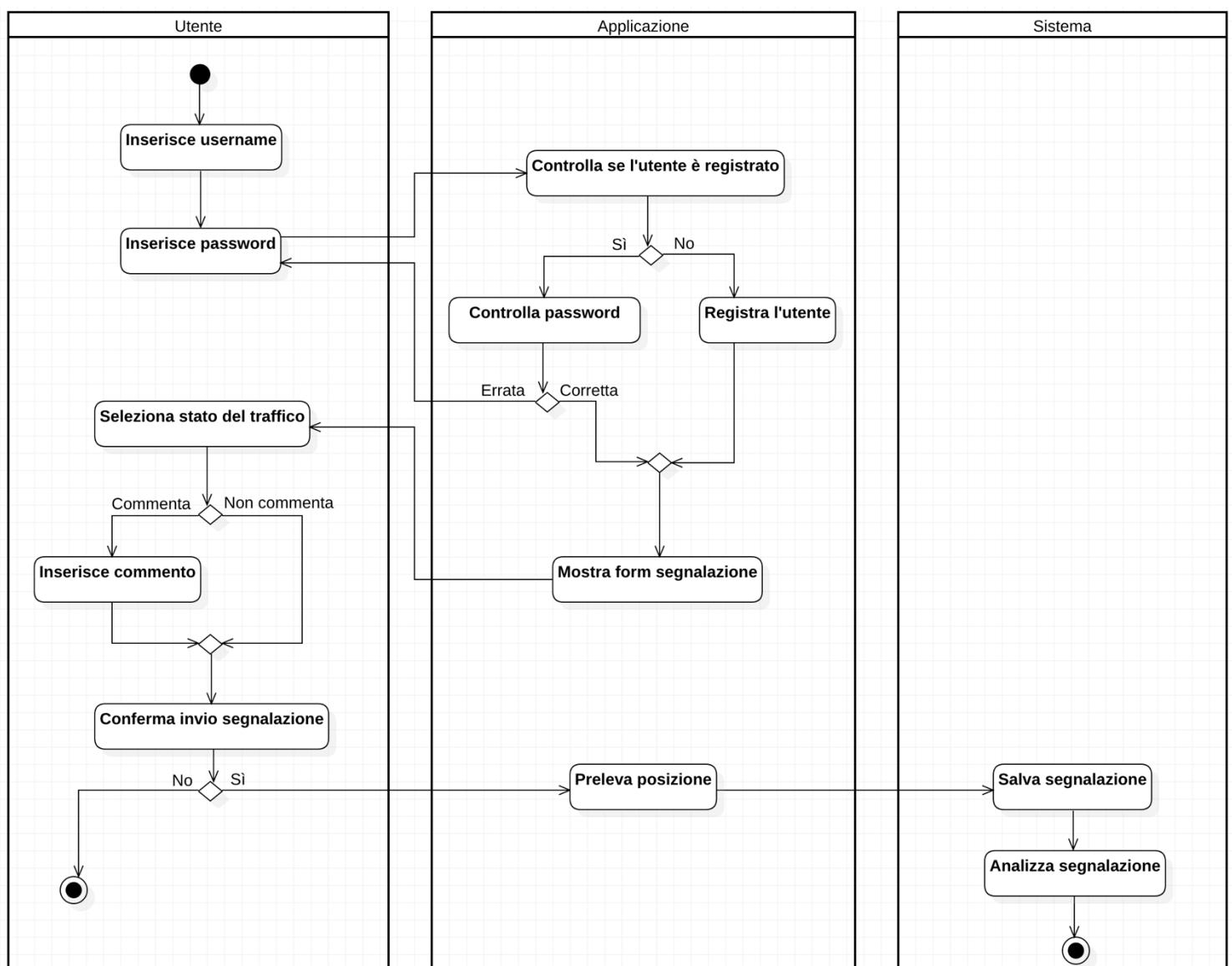
ACTIVITY DIAGRAMS

L'ACTIVITY DIAGRAM mostra una sequenza di passi chiamati attività, punti decisionali e rami che descrivono il flusso del programma.

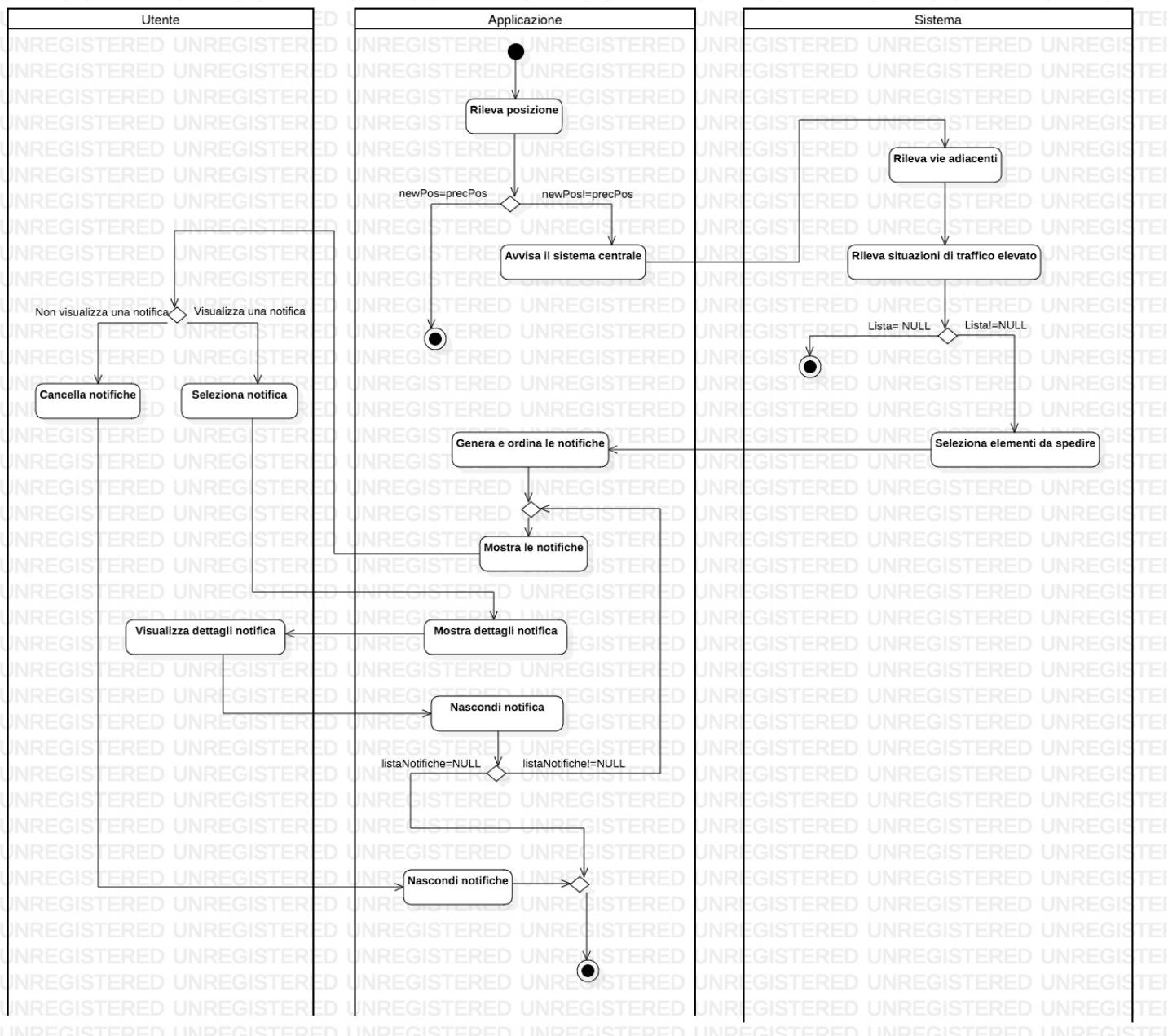
1) Il primo diagramma descrive i passi che esegue l'utente per mandare una **segnalazione** al sistema centrale.

Per poterlo fare l'utente deve accedere all'applicazione che controllerà se è già registrato, se lo è viene controllata se la password è corretta, se invece non lo è viene registrato con la password che ha inserito.

Una volta autenticato viene mostrato il form per l'invio della segnalazione in cui l'utente inserisce lo stato del traffico da una select con opzioni preimpostate e un commento facoltativo che descriva il motivo della situazione di traffico. Quando l'utente conferma l'invio della segnalazione, l'applicazione preleva la posizione dal GPS dello smartphone e manda il tutto al sistema centrale che la salva e la analizza con gli altri dati in suo possesso.



2) Il secondo diagramma descrive invece i passaggi eseguiti per la **notifica** e la visualizzazione di eventi sul traffico quando la posizione cambia. Quando l'applicazione rileva un cambiamento nella posizione, il sistema centrale cerca, tra le vie adiacenti alla nuova posizione rilevata, stati del traffico aventi traffico elevato. Una volta rilevati, gli eventi vengono segnalati all'applicazione che li elabora e li mostra all'utente. Ora inizia il ciclo di comunicazione tra utente ed applicazione che consiste nella richiesta di visualizzazione dei dettagli delle notifiche (e la loro successiva eliminazione). Il ciclo termina a notifiche esaurite o qualora l'utente dovesse scegliere di eliminarle senza visualizzarne i dettagli.



JAVA

CONFIGURAZIONE

Prima del primo avvio del progetto leggere il file README.

CONNESSIONE RMI

La connessione tramite RMI è stata realizzata in locale (localhost) che permette la comunicazione tra: centraline – sistema centrale – applicazioni.

FUNZIONAMENTO RMI

Il funzionamento del RMI è garantito con la connessione ad internet spenta. Dato che alcune configurazioni di sicurezza del firewall dei router possono bloccare la comunicazione del RMI.

CONNESSIONE DATABASE

Il database è stato realizzato in locale (localhost), che viene avviato tramite il software XAMPP.

Download XAMPP: <https://www.apachefriends.org/it/download.html>

CREAZIONE DATABASE E TABELLE

Tramite il pannello phpmyadmin copiare e incollare le seguenti query:

```
GRANT ALL PRIVILEGES ON *.* TO 'gruppo09'@'localhost' IDENTIFIED  
BY 'gruppo09';  
  
CREATE DATABASE gruppo09 DEFAULT CHARACTER SET utf8 COLLATE  
utf8_unicode_ci;  
USE gruppo09;  
  
CREATE TABLE utenti ( id smallint unsigned not null  
auto_increment, username varchar(20) not null unique, password  
varchar(32) not null, via varchar(64) not null, constraint  
chiave_primaria primary key (id) );  
  
CREATE TABLE vie ( via varchar(64) not null unique, limite tinyint  
unsigned not null, constraint chiave_primaria primary key (via) );  
  
CREATE TABLE statitraffico (via varchar(64) not null unique, stato  
smallint unsigned not null, correttezza tinyint unsigned not null,  
orario datetime, constraint chiave_primaria primary key (via) );
```

POPOLAMENTO DATABASE

Tramite il pannello phpmyadmin copiare e incollare le seguenti query:

SISTEMA CENTRALE

GESTIONE DATI IN INGRESSO – ANALISI E STRATEGIA

Il sistema centrale (in dettaglio l’istanza della classe GestoreTraffic) ha il compito di prelevare ed elaborare i dati segnalati dalle applicazioni e/o dalle centraline stradali. Essendo l’elaborazione sviluppata tramite processi concorrenti, in un’eventuale situazione di traffico o di ridotta velocità di calcolo del sistema, il rischio di perdita dei dati in ingresso aumenta. Ciò è causato dal fatto che l’elaborazione di un dato richiede la conoscenza del risultato della precedente elaborazione, ovvero, se un dato A inizia l’elaborazione prima del termine dell’elaborazione di un dato B, A verrà sviluppato utilizzando lo stesso risultato C su cui si è basato B, causando, al termine del processo di elaborazione di A, una sovrascrittura del risultato di B. Ciò che si vuole, quindi, è che A attenda il termine di B per l’inizio dell’elaborazione, così da poter ricavare un risultato sulla base di C+B. Per la soluzione di questo problema, ogni dato in ingresso verrà accodato ad una lista di politica FIFO. Al termine dell’accodamento, se la lista presenta solo il dato appena inserito (ovvero non è in esecuzione alcun calcolo sullo stato del traffico), il sistema centrale inizierà l’elaborazione. A risultato ottenuto, il dato verrà eliminato dalla lista e, se questa non dovesse risultare vuota (ovvero durante l’elaborazione sono stati rilevati ed accodati alla lista altri dati), si inizierà una nuova elaborazione prelevando dalla testa.

È possibile allora individuare due principali situazioni; un dato A viene ricevuto dal sistema centrale che, osservando che la lista d’attesa non presenta dati accodati, lo inserisce e ne inizia l’elaborazione, ed un dato B viene ricevuto dal sistema centrale che, osservando che la lista di attesa non è vuota, vi accoda il dato letto e termina l’esecuzione del processo.

È importante, inoltre, riconoscere che l’elaborazione di un dato richiede la conoscenza dei soli risultati riguardanti la sua stessa posizione, motivo per il quale vi saranno tante liste di attesa quante saranno le vie analizzate.

ELABORAZIONE STATO – ANALISI E STRATEGIA

Anzitutto è stato deciso di discretizzare lo stato del traffico e di utilizzare solo 5 identificatori; BLOCCATO (stato in cui il traffico appare bloccato e l’avanzare delle auto risulta minimo), ELEVATO (l’avanzamento delle auto non risulta bloccato, ma comunque molto limitato e basso), MEDIO (le auto non appaiono bloccate ed il loro flusso appare solamente rallentato), SCORREVOLE (l’avanzamento delle auto risulta scorrevole e fluido) ed ASSENTE (la strada appare completamente libera).

Sono state rilevate importanti problematiche riguardanti l'elaborazione dello stato del traffico; dovrà basarsi su dati di dubbia correttezza e provenienti da differenti fonti, più o meno attendibili. È necessario e fondamentale, quindi, che qualunque sia il metodo di calcolo sviluppato per l'elaborazione dello stato del traffico, venga tenuta in considerazione la possibilità che alcuni dati in ingresso potrebbero essere effettivamente errati e/o corrotti, e si provveda alla realizzazione di un metodo per limitarne i danni. La nostra soluzione consiste nell'assegnare delle priorità (o gradi di affidabilità) ed implementare la modellizzazione di un livello di correttezza.

Alle segnalazioni degli utenti è stato assegnato il valore di affidabilità 1 (l'utente potrebbe effettuare segnalazioni erronee a causa della sua natura burlona, e/o potrebbe non essere in grado di riconoscere correttamente lo stato del traffico) e 4 per le centraline stradali (avendo a che fare con numerose rilevazioni ed utilizzando un sistema di identificazione del traffico appositamente sviluppato, è giusto ipotizzarne una elevata coerenza nei dati segnalati).

Il livello di correttezza, invece, è assegnato allo stato del traffico generale ricavato dal sistema centrale e consiste in una scala di valori interi da 0 a 5. Più il valore della scala è elevato, più è auspicabile ipotizzare che lo stato del traffico rilevato sia effettivamente corretto.

Il sistema centrale (nel dettaglio l'istanza di GestoreTraffico) utilizzerà quindi correttezza ed affidabilità per elaborare uno stato del traffico il più possibile vicino alla realtà. L'elaborazione consisterà nel confrontare lo stato del traffico generale del sistema centrale con quello segnalato dal dato in elaborazione. Si procede riducendo o aumentando la correttezza del valore dell'affidabilità in base al loro confronto. Quando la differenza assume un valore negativo, il livello di traffico generale del sistema viene modificato in quello rilevato dal dato in elaborazione, ed il livello di correttezza viene impostato al modulo del valore negativo della sottrazione effettuata.

ESEMPIO: il sistema centrale ha, come risultato delle analisi precedenti, uno stato del traffico con livello di traffico ELEVATO e correttezza 2. Il dato in ingresso è stato effettuato invece da una centralina stradale (quindi affidabilità 4) ed ha segnalato un livello di traffico SCORREVOLE. Essendo i due livelli di traffico differenti, si esegue la sottrazione tra correttezza ed affidabilità; 2-4. Il risultato è negativo, motivo per il quale il livello di traffico dello stato del traffico del sistema cambia a SCORREVOLE con correttezza pari al modulo del risultato della sottrazione $|-2|=2$. Stato del traffico finale, quindi, risulta essere SCORREVOLE con correttezza 2.

Per risolvere la questione che, dopo un certo periodo, a mancanza di dati in ingresso, lo stato del traffico rilevato potrebbe essere cambiato, si è deciso di

ridurre la correttezza di 1 ogni 10 minuti (solo se maggiore di 0) cosicché, con la successiva segnalazione, si riconfermerà o si cambierà lo stato del traffico.

GESTIONE STATO – REALIZZAZIONE

GestoreTraffic

- nuovoDato(): è il metodo principale del sistema centrale. Ha il compito di prelevare i dati, elaborarli, gestirli e richiederne l'analisi. Nel codice è mostrato come viene gestita la coda d'ingresso per una stessa via

```
100     // ELABORA SOLO SE LISTA VUOTA
101     if (lista.isEmpty() == true)
102     {
103         System.out.println("dato in calcolo");
104         lista.add( nuovoDato );
105         // ELABORA FINO A QUANDO LA LISTA NON è VUOTA
106         while(lista.isEmpty() == false)
107         {
108             analizzaTraffico( nuovoDato );
109             nuovoDato=lista.get(0);
110             lista.remove(0);
111             nuovoDato.kill();
112         }
113     }
114     // SE LISTA!=VUOTA ACCODA IL DATO
115     else
116     {
117         System.out.println("dato accodato");
118         lista.add(nuovoDato);
119     }
120 }
```

- analizzaTraffico(): è il metodo di elaborazione dello stato del traffico. Gestisce inizialmente il valore del dato in ingresso comparandolo con lo stato del traffico precedentemente rilevato (in caso di assenza di quest'ultimo, questi viene inizializzato e impostato al valore del dato). Successivamente si procede alla modifica dello stato del traffico. Nel codice riportato è possibile osservare come è stata gestita la questione della riduzione della correttezza ogni 10 minuti di inattività.

```

142     // la correttezza viene ridotta di 1 ogni 10 minuti
143     trafficoNew.setOrario(d.getOrario());
144     Calendar cOld=Calendar.getInstance();
145     Calendar cNew=Calendar.getInstance();
146     cOld=trafficoNew.getPrimoOrario();
147     cNew.setTime(trafficoNew.getOrario().getTime());
148     cNew.set(Calendar.MINUTE, cNew.get(Calendar.MINUTE)-10);
149     while(cOld.compareTo(cNew)<0 && trafficoNew.getCorrettezza()>0)
150     {
151         //System.out.println(trafficoNew.getCorrettezza());
152         trafficoNew.diminisciCorrettezza();
153         cNew.set(Calendar.MINUTE, cNew.get(Calendar.MINUTE)-10);
154     }

```

La sezione successiva, invece, è il procedimento mediante il quale si stabilisce se modificare il valore generale dello stato del traffico e se notificare gli utenti (e con quale evento). Nel caso di modifica dello stato generale del traffico, vengono impostati nuovi valori agli orari di utilizzo ed allo stato del traffico di gestione.

```

164     // si verifica se notificare solo se la correttezza è >= 2
165     if (trafficoNew.getCorrettezza()>=1 || nuovoData==true)
166     {
167         StatoTraffic trafficoOld= new StatoTraffic();
168         if (mappaTrafficElaborato.containsKey(via)==true)
169         {
170             trafficoOld=mappaTrafficElaborato.get(via);
171         }
172         else
173         {
174             trafficoOld=null;
175         }
176         trafficoNew.modulazioneOra();
177         // se non sono stati registrati dati elaborati sul traffico
178         if (trafficoOld==null)
179         {
180             if (trafficoNew.getTraffic()==Traffic.ELEVATO)
181             {
182                 gestoreClient.notificaClient(Evento.AUMENTATOELEVATO, via);
183             }
184             else if (trafficoNew.getTraffic()==Traffic.BLOCCATO)
185             {
186                 gestoreClient.notificaClient(Evento.AUMENTATOBLOCCATO, via);
187             }
188             else if (trafficoNew.getTraffic()==Traffic.MEDIO)
189             {
190                 gestoreClient.notificaClient(Evento.AUMENTATOMEDIO, via);
191             }
192         }
193         // ricerca l'esistenza di un evento comparando old con new
194         else
195         {
196             Evento evento=rilevaCambiamentoTraffic(trafficoOld, trafficoNew);
197             //notifica l'evento
198             if (evento!=null)
199             {
200                 gestoreClient.notificaClient(evento, via);
201             }
202         }
203         // memorizza il dato su mappaTrafficElaborato sostituendo orario con primoOrario
204         mappaTrafficElaborato.remove(via);
205         trafficoOld=new StatoTraffic();
206         trafficoOld.setCorrettezza(trafficoNew.getCorrettezza());
207         trafficoOld.setOrarioCalendar(trafficoNew.getPrimoOrario());
208         trafficoOld.setTraffic(trafficoNew.getTraffic());
209         mappaTrafficElaborato.put(via, trafficoOld);
210         this.grafica.ridisegna();
211     }

```

GESTIONE NOTIFICHE – ANALISI E STRATEGIA

È richiesto (da specifiche) che all’utente venga notificato lo stato del traffico delle vie a lui prossime tramite un raggio d’azione. Abbiamo deciso, però, di procedere modificando lo strumento per la notifica da raggio d’azione a distanza d’azione; all’utente non verranno notificate le vie all’interno della circonferenza a lui circoscritta, ma solamente le vie raggiungibili percorrendo una distanza TOT dal successivo incrocio. Ciò è stato scelto onde evitare situazioni in cui l’utente potrebbe ricevere notifiche sicuramente non rilevanti. Inoltre, le vie (solamente per la questione delle notifiche) sono state spezzate in segmento. Ogni segmento si estende da un incrocio (o punto di partenza della via) all’incrocio successivo (o termine della via). Grazie a questo modello risulta ora possibile evitare ulteriormente la notifica di stati del traffico non rilevanti per l’utente (se l’utente ha già superato un incrocio, non avrebbe senso notificargli lo stato delle vie raggiungibili da quell’incrocio dal momento che ormai è passato).

È importante anche che il sistema centrale tenga traccia delle vie notificabili (vie notificate e vie non notificate ma adiacenti alla posizione attuale dell’utente) di ogni utente così da evitare una inutile ripetizione di segnalazioni già effettuate e di permettere l’elaborazione di un sistema in grado di notificare l’utente qualora questi non cambi la sua posizione. Esempio: si ipotizzi un utente generico A, fisso in una posizione B. Nel momento in cui A si era spostato in B, gli era stato notificato lo stato del traffico di C (via adiacente a B). Rimanendo fermo, però, non richiede più al sistema centrale alcun aggiornamento. Sarà questi, allora, che, nel momento in cui verrà rilevato un evento particolare, dovrà interrogare i suoi dati per trovare gli utenti con una via notificabile (o posizione attuale) concordante con quella dell’evento ed avvisarlo.

GESTIONE NOTIFICHE – REALIZZAZIONE

GestoreNotifiche

- `RegistraClient()`: nel codice è possibile riconoscere (nella sezione successiva alla connessione RMI) come l’utente venga identificato tramite la porta RMI utilizzata per la connessione. Ciò consente anche un automatico aggiornamento dei client “morti”, in quanto quando uno di essi dovesse diventare in attivo, la successiva richiesta di registrazione verrà effettuata sulla sua porta RMI.

```

71  /* METODO PER LA REGISTRAZIONE DI UN CLIENT */
72  public void registraClient(int tmp) throws RemoteException
73  {
74      try {
75          Registry registry = LocateRegistry.getRegistry("127.0.0.1", 12346+tmp);
76          IRmiApplicazione serverApp = (IRmiApplicazione) registry.lookup("Applicazione");
77          DatiClient dc= new DatiClient();
78          dc.setApplicazione(serverApp);
79          if (gestoreClient.getMappaClient().containsKey(tmp))
80          {
81              // si elimina il client se ne viene rilevato uno nuovo nella sua porta
82              gestoreClient.getMappaClient().get(tmp).kill();
83              gestoreClient.getMappaClient().remove(tmp);
84              gestoreClient.getMappaClient().put(tmp, dc);
85          }
86          else
87          {
88              // si registra il client
89              gestoreClient.getMappaClient().put(tmp, dc);
90              gestoreClient.getListeClient().add(tmp);
91          }
92          serverApp.ciao();
93      } catch (Exception e) {
94          e.printStackTrace();
95      }
96  }
97
98

```

- rilevaVieDintorni(): consiste nel metodo per il calcolo dei segmenti percorribili con una distanza pari al parametro distanza. È un metodo ricorsivo, ed i segmenti adiacenti vengono ritornati nella lista di parametro.

```

188  /* METODO PER LA RICERCA DELLE VIE ADIACENTI */
189  public void rilevaVieDintorni(String segmento, int distanza, ArrayList<String> vecchie)
190  {
191      // l'array di ritorno è l'array parametro
192      ArrayList<String> adiac=segmentiAdiacenti.get(segmento);
193      for (int i=0; i<adiac.size(); i++)
194      {
195          if (vecchie.contains(adiac.get(i))==false && adiac.get(i)!= segmento) {
196              vecchie.add(adiac.get(i));
197              if (infoSegmento.get(adiac.get(i)).getLunghezza()<distanza)
198              {
199                  rilevaVieDintorni(adiac.get(i), distanza-infoSegmento.get(adiac.get(i)).getLunghezza(), vecchie);
200              }
201          }
202      }
203  }
204
205
206

```

- rilevaNotifiche(): consiste nel metodo per il l'elaborazione delle notifiche riguardanti una posizione. Per fare ciò calcolerà le vie da notificare tramite il metodo rilevaVieDintorni(), per poi filtrarle e selezionare solo quelle avendo uno stato del traffico non basso. A queste si applicherà un ulteriore taglio andando a considerare solo le vie non notificate all'utente (come mostrato nel codice).

```

154     ArrayList<String> oldListaVie;
155     if (client == null)
156     {
157         oldListaVie= new ArrayList<String>();
158     }
159     else
160     {
161         oldListaVie=client.getVie();
162     }
163     for(int i=0; i<listaVie.size(); i++)
164     {
165         if (oldListaVie.contains(listaVie.get(i))==false)
166         {
167             traffico=this.gestoreTraffico.getTraffico(listaVie.get(i));
168             // se è stato effettuata una rilevazione del traffico sulla via si procede
169             if (traffico != null)
170             {
171                 if (
172                     traffico.getTraffico()==Traffico.BLOCCATO ||
173                     traffico.getTraffico()==Traffico.ELEVATO ||
174                     traffico.getTraffico()==Traffico.MEDIO)
175                 {
176                     //analizza ed elabora una notifica
177                     notifiche.add(analizzaNotifica(traffico,orario) + "/" + listaVie.get(i));
178                 }
179             }
180         }
181     }
182     if (client!=null) client.clearVie();
183     if (client!=null) client.setVie(listaVie);
184     return notifiche;
185 }

```

GESTIONE UTENTI – ANALISI E STRATEGIA

Come definito nella sezione Analisi e Strategia di Gestione Notifiche, il sistema centrale monitorerà gli spostamenti degli utenti. Ad autenticazione effettuata, quindi, l’utente comunicherà al sistema centrale la sua attività, e ad ogni richiesta di notifiche, l’istanza di GestoreClient, tramite GestoreNotifiche, memorizzerà i dati rilevanti prelevati dalla comunicazione. La gestione degli utenti inattivi verrà automaticamente applicata tramite la memorizzazione di un nuovo client; quando un’applicazione si spegne, il suo identificatore torna libero. In questo modo la successiva applicazione che richiederà la registrazione al sistema centrale utilizzerà l’identificatore di quella appena spenta. Il sistema centrale sostituirà così i dati “morti” con quelli del nuovo utente attivo.

GESTIONE UTENTI - REALIZZAZIONE

GestoreClient

- notificaClient(): tramite questo metodo, viene elaborata una stringa testo inerente all’evento rilevato per essere notificata a tutti gli utenti aventi la posizione passata per parametro al metodo come via notificata.

```

29e    public void notificaClient(Evento evento, String via) {
30        String commento=null;
31        for (int i=0; i< listaClient.size(); i++)
32        {
33            // se un'applicazione si trova nella via o le è stata notificata la via
34            if (mappaClient.get(listaClient.get(i)).getPosizione()==via ||
35                mappaClient.get(listaClient.get(i)).getVie().contains(via))
36            {
37                // generazione testo
38                if (evento==Evento.AUMENTATOBLOCCATO) commento="BLOCCATO! ";
39                else if (evento==Evento.AUMENTATOELEVATO) commento="ELEVATO! ";
40                else if (evento==Evento.AUMENTATOMEDIO) commento="medio ";
41                else if (evento==Evento.CALATOMEDIO) commento="calato a medio! ";
42                else if (evento==Evento.CALATOELEVATO) commento="calato ad elevato ";
43                else if (evento==Evento.CALATOSCORREVOLE) commento="calato a scorrevole ";
44                else if (evento==Evento.CALATOASSENTE) commento="calato ad assente ";
45                else if (evento==Evento.RIMASTOELEVATO) commento="COSTANTEMENTE ELEVATO ";
46                else if (evento==Evento.RIMASTOBLOCCATO) commento="COSTANTEMENTE BLOCCATO";
47                String notifica= commento + "/" + via;
48                try {
49                    mappaClient.get(listaClient.get(i)).getApplicazione().addNotifica(notifica);
50                } catch (RemoteException e) {
51                    // TODO Auto-generated catch block
52                    e.printStackTrace();
53                }
54            }
55        }
56    }
57 }

```

- memorizzazione client: i dati del client vengono memorizzati tramite una Hash map avente per chiave l'identificatore della porta RMI utilizzata per la connessione.

```

15
16     private Map<Integer,DatiClient> mappaClient;
17     private ArrayList<Integer> listaClient;
18
19
20     /* COSTRUTTORE */
21e    public GestoreClient()
22    {
23        this.mappaClient=new HashMap<Integer, DatiClient>();
24        listaClient= new ArrayList<Integer>();
25    }
26

```

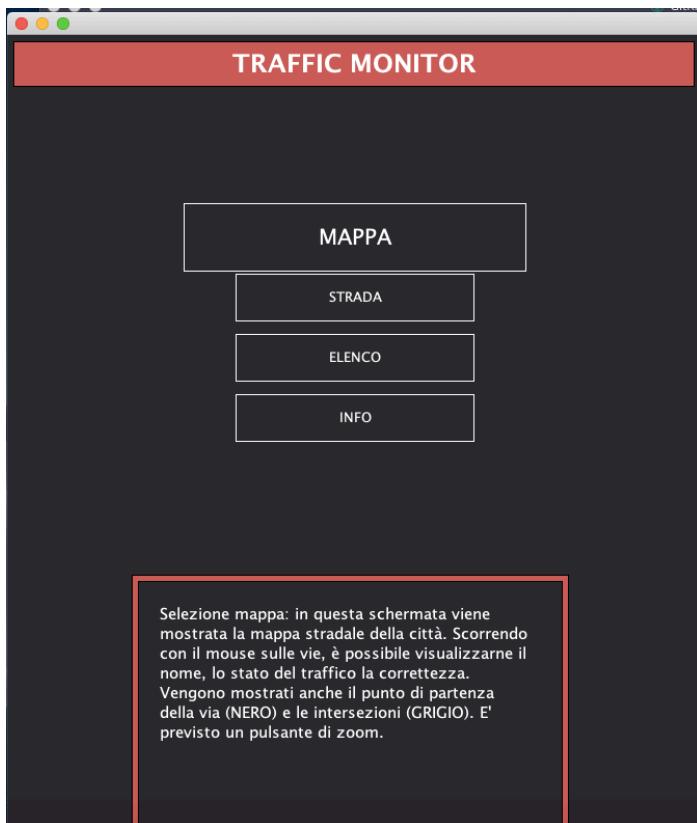
RMI – SETTAGGIO SERVER

```

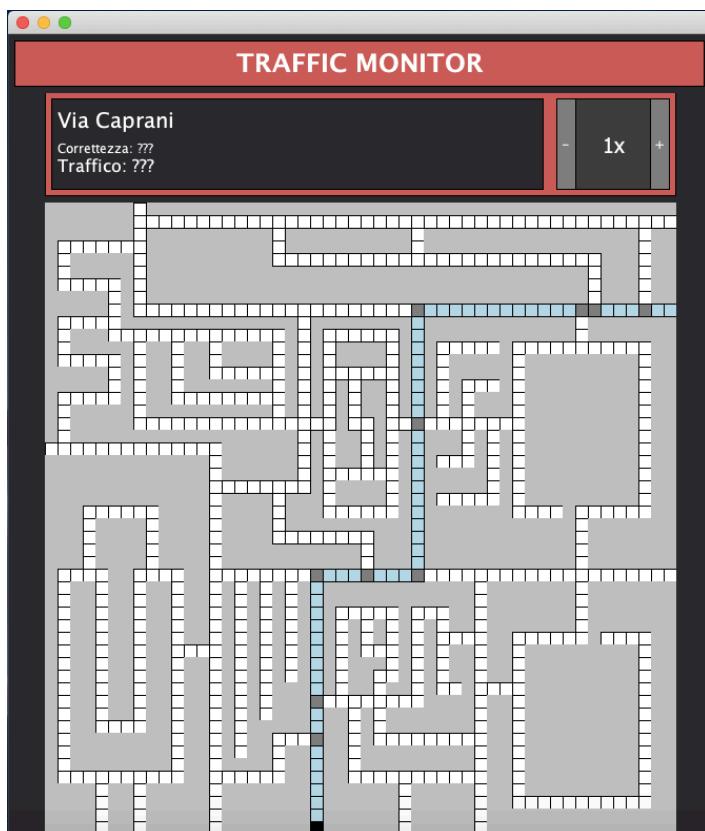
166     // RMI
167     Registry registry = null;
168     try {
169         registry = LocateRegistry.createRegistry(12345);
170         registry.rebind("GestoreTrafico", gt);
171         registry = LocateRegistry.createRegistry(12346);
172         registry.rebind("GestoreNotifiche", gnt);
173         System.out.println("Server online");
174     } catch (Exception e) {
175         e.printStackTrace();
176     }
177 }

```

GRAFICA



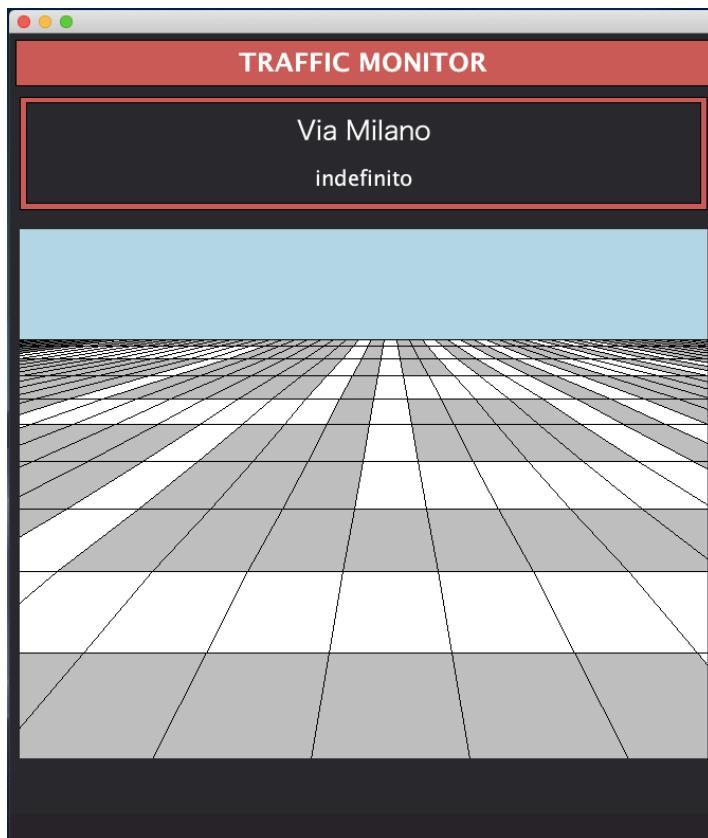
MENU: è la schermata principale tramite la quale l'utente può selezionare la modalità di visualizzazione dei dati; mappa, strada ed elenco (ancora in fase di sviluppo). Nel pannello sottostante alle opzioni viene velocemente descritta velocemente la funzionalità della modalità di visualizzazione puntata. È presente anche un ulteriore menù rapido visualizzabile portando il cursore sul logo che consentirà la navigazione rapida tra schermate.



MAPPA: consiste nella visualizzazione grafica 2D della mappa stradale della città. Per la realizzazione si è scelto l'utilizzo di una mappa visualizzabile come una matrice 50x50. È importante precisare che le dimensioni delle strade sono state rese irrealistiche al fine di garantire una migliore esperienza in fase di test. In ogni caso, anche portando le dimensioni ad una grandezza realistica (matrice 1000x1000) il modello grafico riuscirebbe comunque a sostenere le

modifiche essendo che è stato implementato un metodo per la gestione delle dimensioni (zoom, applicabile anche per la matrice 50x50).

Spostando il cursore all'interno della mappa è possibile visualizzare le caratteristiche principali di una via semplicemente puntandola. Nel riquadro in alto verranno rappresentati in forma scritta il nome della via, il suo livello di correttezza e lo stato del traffico (correttezza e traffico verranno impostati a ??? in caso di mancanza di rilevazioni). Nella mappa, invece, a via puntata, verrà colorata del colore rispettivo al suo stato del traffico (blu in assenza di rilevazioni) specificando in nero il suo punto d'inizio ed in grigio le posizioni degli incroci (utile per identificare i segmenti).



STRADA: in questa sezione, invece, è possibile visualizzare la mappa stradale tramite una grafica 3D appositamente sviluppata. In questa modalità le strade assumono ed assumeranno il colore dello stato del traffico nell'istante attuale (verde -> assente, verdino giallo -> scorrevole, giallo arancio -> medio, rosso -> elevato, rosso scuro -> bloccato). Spostando il cursore sulla rappresentazione grafica di una via, verranno stampati nel riquadro apposito nome

e stato del traffico (“indefinito” se assente). È previsto un metodo per lo spostamento della visuale premendo e trascinando il mouse nella direzione interessata.

CENTRALINA STRADALE

STRUTTURA E GENERALITA' CENTRALINA

La centralina stradale è stata sviluppata partendo dall'estensione di due classi astratte (Centralina Stradale A Sensore Singolo estende Centralina Stradale Generica che estende Centralina) nell'idea di consentire, se dovesse essere necessario, l'implementazione di differenti tipologie di queste. Si è deciso di sviluppare la CentralinaStradaleASensoreSingolo che, come definito dal nome, si tratta di una centralina stradale che opera sui dati prelevati da un singolo sensore. Questo sensore comunicherà alla centralina i Timestamp in cui inizia a rilevare il passaggio di un'automobile e di quando ne terminerà la rilevazione. Per effettuare il testing del sistema è stato implementato un algoritmo di simulazione stradale basilare. Questo metodo consiste nell'estrarrre casualmente, di volta in volta, il nuovo valore della velocità da rilevare (l'estrazione è manipolata in maniera tale da rendere maggiormente probabile l'estrazione di un dato simile a quello precedentemente rilevato) e segnalarlo alla centralina tramite la segnalazione di inizio e fine rilevazione (è stato quindi necessario utilizzare la formula $velocità = spazio / tempo$, così da poter calcolare dopo quanto tempo far scattare il segnale di fine rilevazione affinché la centralina possa leggere la velocità desiderata).

CALCOLO STATO- ANALISI E STRATEGIA

La centralina stradale si troverà a dover far fronte a numerosi dati mentitori (dati che mentono sul loro significato). Esempio: in una strada avente limite di velocità 50km/h viene effettuata la rilevazione di una velocità di 20 km/h. questa velocità, tuttavia, non significa necessariamente "Trafico elevato", ma potrebbe essere dovuta da infinite altre possibilità. È necessario quindi che il sistema di calcolo della centralina stradale riesca a considerare la possibilità che un dato rilevato sia errato. Per far fronte, quindi, a questa necessità si è sviluppato un metodo di considerazione dell'errore in grado di limitarne i danni. Per fare ciò sono stati necessari 2 dati; velocità media rilevata in un arco temporale TOT e velocità generale rilevata. La velocità media rilevata andrà ad agire direttamente sulla velocità generale, e più i valori dei due dati risulteranno compatibili (simili), più l'effettivo peso della modifica apportata dalla velocità media rilevata sarà elevato. Dopodiché verrà modificato anche il valore dello stato del traffico (basandosi sulla velocità generale) ed eventualmente anche la frequenza di notifica (in dettaglio nella sezione successiva).

CALCOLO STATO- REALIZZAZIONE

CentralinaStradaleASensoreSingolo:

- calcolo(): è il metodo della classe tramite il quale la rilevazione effettuata viene convertita in velocità effettiva (formula $v = \text{spazio} / \text{tempo}$, dove lo spazio si ipotizza essere costante a 4,5 metri, ovvero la lunghezza media di un'auto) per poi essere utilizzata nel calcolo della velocità media

```
/* METODO PER IL CALCOLO DELLA VELOCITA DATI T1, T2 E LUNGHEZZA */
public void calcolo()
{
    long i1=this.rilevazione.getTime();
    long i2=this.rilevazione.getTime();
    float t=i2-i1;
    float v=(this.lunghMediaAuto*36)/(t*10);
    if (v>this.strada.getLimiteVelocita())
    {
        v=this.strada.getLimiteVelocita();
    }
    this.velocitaMedia+=v;
    this.nRilevazioni++;
}
```

- analizzaTrafico(): la prima sezione per l'analisi del traffico (metodo attivato ogni 5 secondi tramite un timer implementato nella classe CentralinaStradale) consiste nel gestire le situazioni critiche nelle quali è stata iniziata la rilevazione di un'auto ma non è stata terminata. Se, ipotizzando il termine della rilevazione nell'istante successivo a quello attuale, si dovesse ottenere una velocità maggiore rispetto alla media rilevata, essa verrebbe scartata (potrebbe metterci di più, ed il dato potrebbe apportare un errore), altrimenti lo si applica al calcolo della media (essendo minore della media, sicuramente comporterebbe una sua riduzione).

```

97         if (this.rilevazione.getRilevato()==true)
98     {
99         long i1=this.rilevazione.getT1().getTime();
100        long i2=getTime();
101        float t=i1-i2;
102        float v=(this.lunghMediaAuto*36)/(t*10);
103        if (nRilevazioni==0 || v<this.velocitaMedia)
104        {
105            this.nRilevazioni++;
106            this.velocitaMedia+=v;
107        }
108    }

```

La seconda sezione consiste, invece, nella conversione della velocità media ad un valore assoluto in percentuale (così da poter considerare nello stesso modo range di valori differenti). È stato riportato nel codice sotto uno spunto su come è stato realizzato il sistema del “più la media è coerente con il valore generale, più lo influenza”.

```

if (this.nRilevazioni!=0)
{
    this.hoLavorato=true;
    float vpm=this.velocitaMedia*100/this.strada.getLimiteVelocita(); // velocità media
    float vgm=this.velocitaGenerale*100/this.strada.getLimiteVelocita(); // velocità generale
    if ((vgm-vpm)>50)
    {
        this.velocitaGenerale=(this.velocitaGenerale+this.velocitaMedia)/2;
    }
    else if ((vgm-vpm)>25)
    {
        this.velocitaGenerale=this.velocitaGenerale*25/100+this.velocitaMedia*75/100;
    }
    else if ((vgm-vpm)>10)
    {
        this.velocitaGenerale=this.velocitaGenerale*10/100+this.velocitaMedia*90/100;
    }
}

```

FREQUENZA DI AGGIORNAMENTO

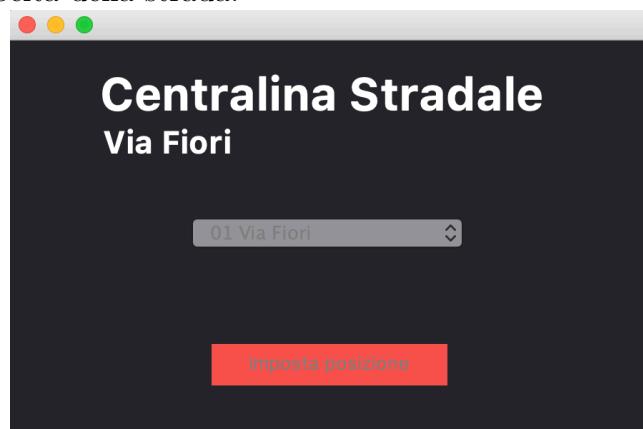
La frequenza di aggiornamento verrà modificata di volta in volta in base allo stato del traffico appena rilevato. Attualmente è stata impostata a valori elevati per potervi effettuare un migliore test; 10s per il traffico bloccato, 20s traffico elevato, 40s traffico medio, 80s traffico scorrevole e 140s traffico assente (in assenza di necessità di test i valori risulterebbero impostati a 1min per il traffico bloccato, 2min traffico elevato, 4min traffico medio, 8min traffico scorrevole e 14min traffico assente).

GRAFICA

La centralina stradale è simulata dall’interfaccia grafica che permette all’avvio di impostare in che via sarà posizionata.



Dopo aver impostato la via e premuto il pulsante “Imposta posizione” verrà invocato il metodo **accendi()** a cui vengono passati come argomenti la via e il limite di velocità della strada.



Dentro il metodo viene fatto partire l’RMI mettendo effettivamente online centralina, da questo momento inizierà a rilevare le velocità, calcolare lo stato del traffico e mandare le informazioni al sistema centrale.

RMI – SETTAGGIO LETTURA

```
69  Registry registry = LocateRegistry.getRegistry("127.0.0.1", 12345);
70  c1 = new CentralinaStradaleASensoreSingolo(via, limiteVelocita, Tipologia.SECONDARIA, SensoDiMarcia.AB);
71
72  IRmiGestoreTrafico server = (IRmiGestoreTrafico) registry.lookup("GestoreTrafico");
73  c1.setServer(server);
74
```

APPLICAZIONE MOBILE

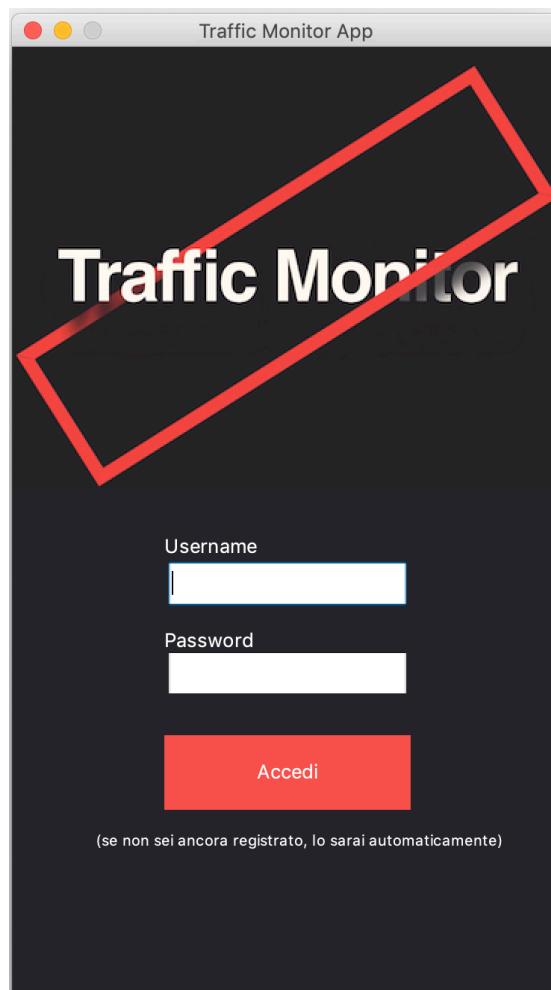
AUTENTICAZIONE

Quando l'applicazione viene aperta, tramite il metodo **mostraFormAutenticazione()** viene mostrato il form, in cui l'utente inserisce le proprie credenziali (username, password).

L'autenticazione dell'utente è stata fatta tramite una logica di login semplificata, dato che l'autenticazione dell'utente è una funzionalità marginale all'interno del progetto.

Se l'utente è registrato controlla se la password è corretta.

Se invece l'utente inserito non è registrato, quindi non presente nel database, lo registra con la password indicata.



Premuto il tasto “accedi” viene chiamato, tramite l’interfaccia IGestoreInterfaccia, il metodo **accedi()**, in cui vengono controllate le credenziali, o in caso non sia registrato viene registrato con la password che ha inserito.

Vengono fatti basilari controlli sull’inserimento e viene poi controllato se l’utente è registrato tramite la funzione **isUtenteRegistrato()** per poi chiamare **autenticaUtente()** oppure **registraUtente()**.

```
public void accedi(String username, String password) {  
  
    //CONDIZIONI PER I CAMPI INSERITI DALL'UTENTE  
    if(username.equals("") || password.equals("")) {  
        JOptionPane.showMessageDialog(null, "Si prega di compilare tutti i campi");  
    }  
    if (username.trim().length() == 0 || password.trim().length() == 0) {  
        JOptionPane.showMessageDialog(null, "Si prega di non inserire spazi");  
    }  
  
    //LOGICA AUTENTICAZIONE  
    if( getAppManager().isUtenteRegistrato(username) ) {  
  
        if ( getAppManager().autenticaUtente(username, password) ) {  
            setUtente(username, password);  
            setPosizione(getAppManager().getUltimaPosizione(getUtente()));  
            mostraPaginaPrincipale();  
        }  
        else {  
            JOptionPane.showMessageDialog(null, "Password errata");  
        }  
    }  
    else {  
        if( getAppManager().registraUtente(username, password) ) {  
            setUtente(username, password);  
            setPosizione(getAppManager().getUltimaPosizione(getUtente()));  
            JOptionPane.showMessageDialog(null, "Ti sei registrato con successo");  
            mostraPaginaPrincipale();  
        }  
        else {  
            JOptionPane.showMessageDialog(null, "Errore nella registrazione");  
        }  
    }  
}
```

PAGINA PRINCIPALE

Terminato la connessione con il sistema centrale, viene visualizzata la pagina principale dell'applicazione tramite il metodo **paginaPrincipale()**, in cui si può decidere se inviare una segnalazione oppure mostrare la lista delle notifiche ricevute in questa sessione.



La logica della grafica per il caricamento pagina principale, è sviluppata utilizzando l'istanza dell'interfaccia grafica all'interno della classe GestoreInterfaccia.

Rappresenta un esempio del modello Model – Control - View per la comunicazione tra front end e back end.

```
public void mostraPaginaPrincipale() {  
    getGUI().getPanelNotifica().setVisible(false);  
    getGUI().getPanelPaginaPrincipale().setVisible(true);  
    getGUI().getPanelAutenticazione().setVisible(false);  
    getGUI().getPanelSegnalazione().setVisible(true);  
    getGUI().getPanelListaNotifiche().setVisible(false);  
}
```

SEGNALAZIONE

Viene mostrato il form segnalazione dove l'utente compila il campo prestabilito riguardante lo stato del traffico e con il tasto “invia segnalazione” viene mandata il sistema centrale.



La segnalazione viene mandata al sistema centrale connesso tramite RMI invocando il metodo **nuovoDato()**.

```
public void notify(Traffico t, String posizione, int importanza) {
    Timestamp orario = new Timestamp (System.currentTimeMillis());
    try {
        getServer().nuovoDato(posizione.substring(3), t.toString(), orario, importanza);
    } catch (RemoteException e) {
        e.printStackTrace();
    }
    JOptionPane.showMessageDialog(null, "Segnalazione inviata per la: " + posizione.substring(3));
}
```

CAMBIO POSIZIONE

Secondo specifica richiesta del tutor del progetto, per favorire il testing (come se fosse una simulazione de GPS), è stata prevista una combobox da cui è possibile cambiare la propria posizione selezionando una via.

```
JComboBox<String> comboBoxPosizione = new JComboBox<String>();
ArrayList<String> vie = new ArrayList<String>(gInterfaccia.getAppManager().popolaComboBoxPosizione());
//sort per via escludendo i segmenti (primi 3 caratteri)
Collections.sort(vie, Comparator.comparing(s -> s.substring(4)));
//popolamento con le vie nel db
for (String via : vie) {
    comboBoxPosizione.addItem(via);
}
comboBoxPosizione.addItemListener(new ItemListener() {
    public void itemStateChanged(ItemEvent e) {
        if (e.getStateChange() == ItemEvent.SELECTED) {
            String v = (String) comboBoxPosizione.getSelectedItem();
            gInterfaccia.posizioneCambiata(v);
        }
    }
});
```

Al cambio della posizione viene mandata, tramite **posizioneCambiata()**, una richiesta al sistema centrale per ricevere le notifiche relative alla nuova via in cui si trova ed in quelle adiacenti.

La funzione chiama **rilevaNotifiche()** che comunicando con il sistema centrale (chiamato server) tramite RMI, in questo modo il sistema centrale conosce la nuova posizione e manda all'applicazione le notifiche inerenti.

```
public void rilevaNotifiche(String via){
    eliminaNotifiche(); //elimina le notifiche relative alla posizione precedente
    try {
        //quando cambia la posizione chiama l'rmii del sist centrale
        for (String el : this.server.rilevaNotifiche(via, this.io, 200)){
            addNotifica(el); //aggiunge le nuove notifiche
        }
    } catch (RemoteException e) {
        e.printStackTrace();
    }
}
```

Infine, viene salvata la via relativa all'utente nella tabella utenti sul database, per poter accedere all'ultima posizione registrata alla prossima riapertura dell'applicazione.

```
public boolean setUltimaPosizione(Utente u, String via) {
    Connection conn = null;
    PreparedStatement stmt = null;
    try {
        String sql = "UPDATE utenti SET via = ? WHERE username = ?"; //sql da eseguire
        conn = ApplicazioneDataManager.connect();
        stmt = conn.prepareStatement(sql);
        stmt.setString(1, via);
        stmt.setString(2, u.getUsername());
        stmt.executeUpdate();
        System.out.println("posizione salvata");
        return true;
    } catch (SQLException e) {
        System.out.println("Salvataggio ultima posizione: tentativo fallito");
    }
}
```

NOTIFICA

La notifica viene mandata dal sistema centrale all'applicazione in modo compresso, tramite una stringa contenente testo e posizione separati da un carattere “/”

```
public void addNotifica(String notifica) throws RemoteException{  
  
    //notifica ricevuta "compressa" come stringa con le info separate da /  
    String[] tmp = notifica.split("/");  
  
    Notifica n = new Notifica(tmp[1], tmp[0]);  
    this.arrayNotifiche.add(n);  
  
    //aggiorna lista notifiche (grafica)  
    getGUI().getListaNotifiche().addElement(n.toString());  
    //mostra notifica push (grafica)  
    getGIInterfaccia().aggiungiNotificaPush(n);  
  
}
```

NOTIFICA PUSH

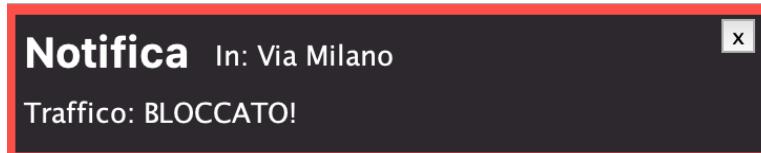
Sulla pagina principale, viene visualizzata come notifica push l'ultima notifica ricevuta dal sistema centrale, con informazioni relative alla via in cui ci si trova oppure alle vie adiacenti.

Il panel che racchiude la notifica, viene mostrato quando si invoca il metodo **addNotifica()** della classe GestoreNotificheApp, che a sua volta invoca il metodo del GestoreInterfaccia chiamato **aggiungiNotificaPush()**.

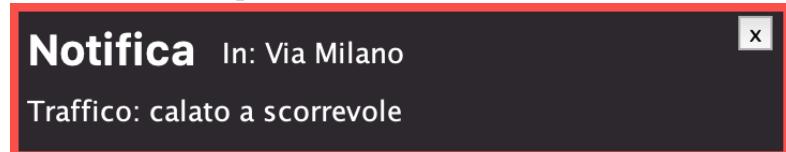
Il testo della notifica è generato dal sistema centrale, ha informazioni sullo stato del traffico attuale della via, unitamente a quelle relative alle condizioni precedenti.

Vi sono due tipi di notifica:

- Notifica che segnala lo stato del traffico



- Notifica che segnala un cambiamento dello stato del traffico basandosi sullo stato rilevato precedentemente

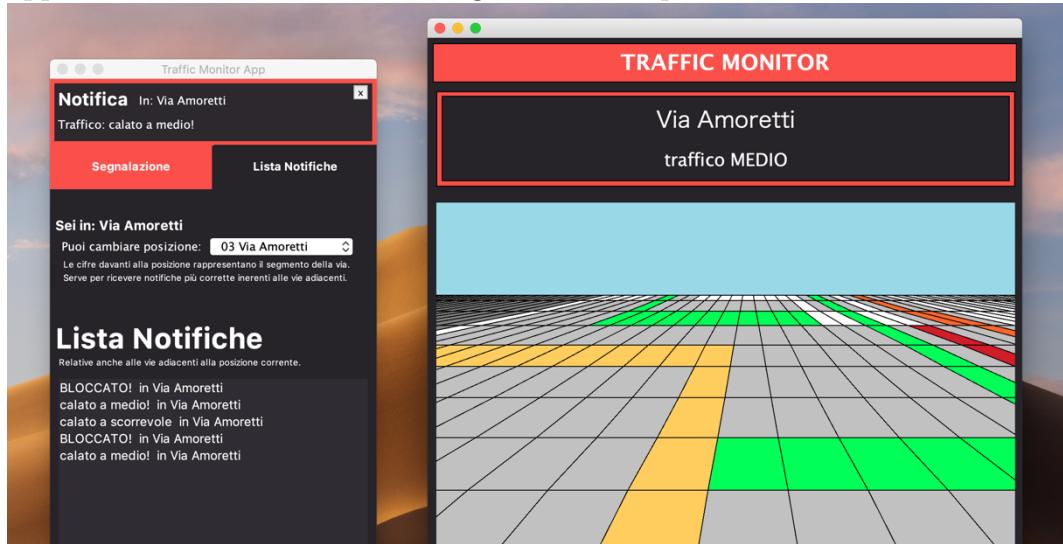


LISTA NOTIFICHE

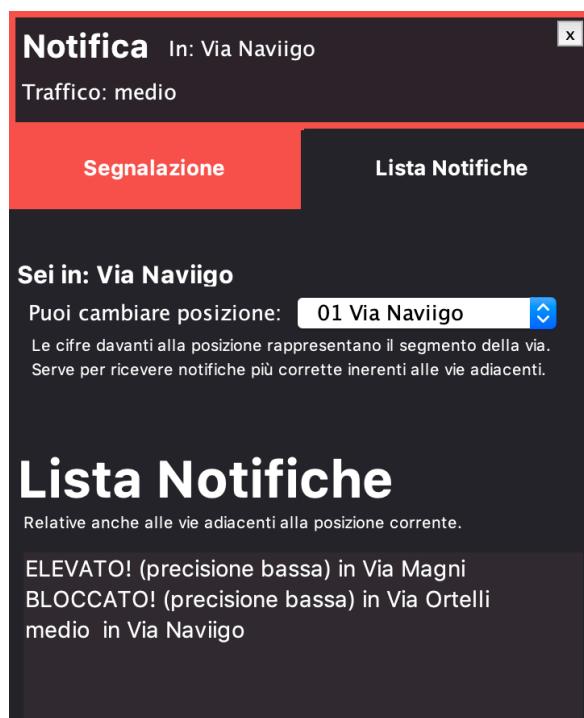
Oltre alla notifica push è possibile accedere alla schermata in cui vengono salvate tutte le notifiche ricevute, riguardo anche le vie adiacenti, memorizzate dal momento in cui l'utente ha cambiato posizione.

Relativamente ad una stessa via, le notifiche ricevute sono ordinate in termini di tempo. L'ultima rappresenta quella più recente, creando una sorta di storico relativo alle sue condizioni di traffico nel tempo.

In questa schermata viene mostrata la corrispondenza delle informazioni, tra applicazione e sistema centrale, riguardo la via presa in considerazione.



Inoltre, al cambio di posizione sono salvate le notifiche relative alle condizioni di traffico rilevanti riguardo le vie adiacenti.



DATABASE

TABELLE DATABASE

La tabella **utente** è necessaria all'applicazione mobile per l'autenticazione e registrazione dell'utente oltre al salvataggio della sua ultima posizione, che verrà poi utilizzata come prima posizione al successivo utilizzo dell'app.

← T →		▼	id	▼ 1	username	password	via
<input type="checkbox"/>	Modifica	Copia	Elimina	7	test	test	01 Via Bassetti

La tabella **vie** è necessaria per il popolamento della combobox, richiesta da tutor per favorire il testing dell'applicazione mobile, e per istanziare la posizione della centralina stradale.

← T →		▼	via	limite
<input type="checkbox"/>	Modifica	Copia	Elimina	01 Via Amoretti 90
				01 Via Anzani 50

CONNESSIONE AL DATABASE

La connessione al database viene gestita dalla classe **ApplicazioneDataManager**, che utilizza una libreria Java chiamata JDBC. La classe implementa il metodo **connect()** che viene poi utilizzato per generare una connessione col database, per poter eseguire query SQL.

```
public class ApplicazioneDataManager {  
    private static String url = "jdbc:mysql://localhost:3306/gruppo09"; //stringa di connessione  
    private static String driver = "com.mysql.cj.jdbc.Driver";  
    private static String username = "gruppo09";  
    private static String password = "gruppo09";  
    private static Connection con; //connessione col db  
  
    public static Connection connect() {  
        //Source https://stackoverflow.com/questions/10915375/create-a-class-to-connect-to-any-da  
        try {  
            Class.forName(driver);  
            try {  
                //tentativo di connessione al db  
                con = DriverManager.getConnection(url, username, password);  
            } catch (SQLException ex) {  
                // log exception  
                System.out.println("Fallito il tentativo di connessione al database");  
            }  
        } catch (ClassNotFoundException ex) {  
            // log exception  
            System.out.println("Driver non disponibile");  
        }  
        return con;  
    }  
}
```

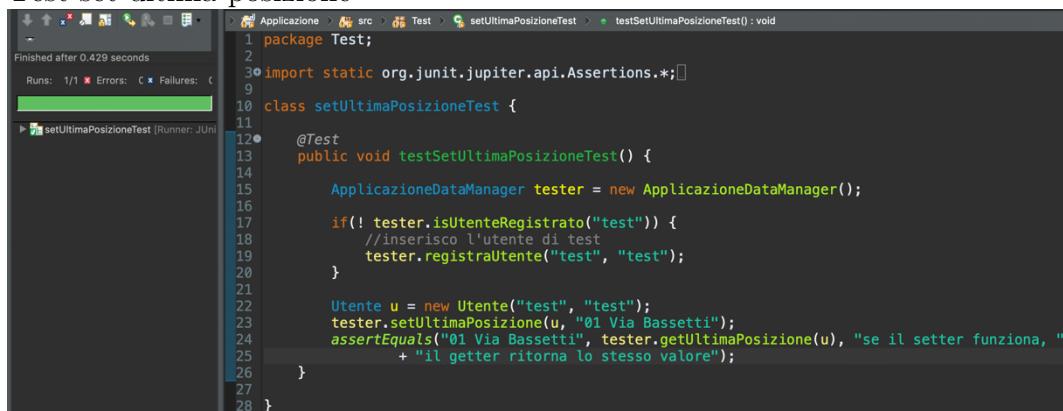
TESTING CON JUNIT

Come da specifiche sono stati creati, utilizzando JUnit, specifici test per le funzioni più importanti, il cui funzionamento era imprescindibile al funzionamento delle funzionalità del progetto. All'interno del codice sorgente dei tre progetti (applicazione, sistema centrale, centralina stradale), vi è un package chiamato "Test" in cui sono presenti i test.

Di seguito sono spiegati i JUnit test:

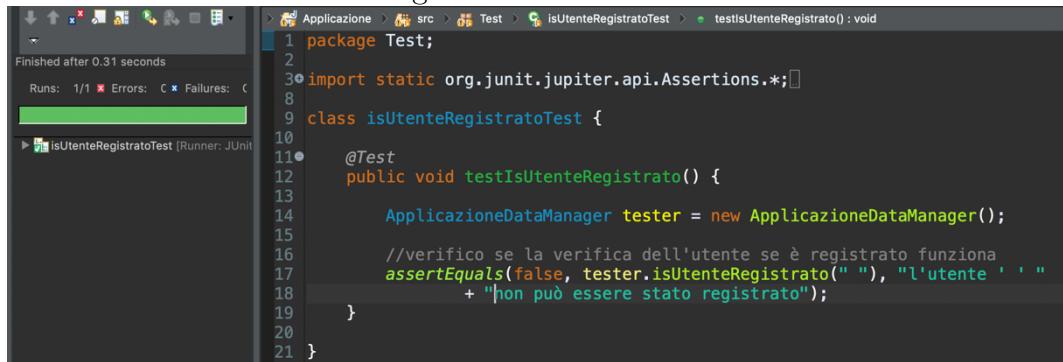
JUNIT TEST APPLICAZIONE

Test set ultima posizione



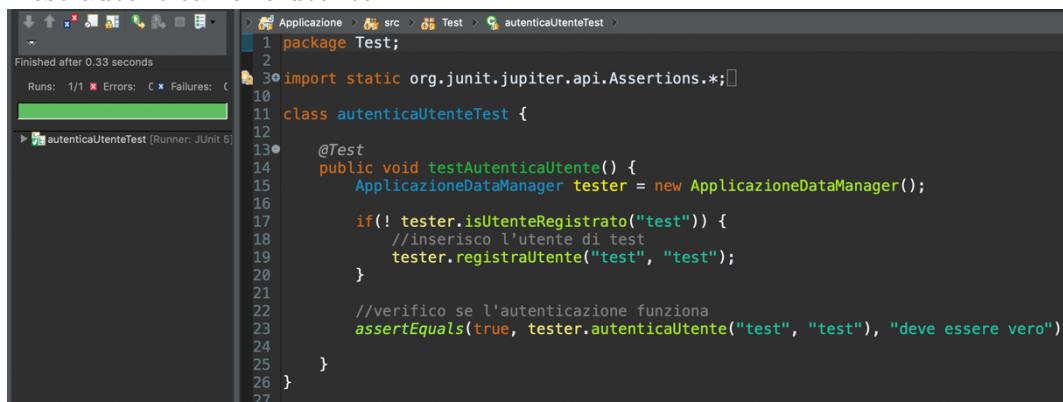
```
1 package Test;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 class setUltimaPosizioneTest {
6
7     @Test
8     public void testSetUltimaPosizione() {
9
10         ApplicazioneDataManager tester = new ApplicazioneDataManager();
11
12         if(! tester.isUtenteRegistrato("test")) {
13             //inserisco l'utente di test
14             tester.registraUtente("test", "test");
15
16             Utente u = new Utente("test", "test");
17             tester.setUltimaPosizione(u, "01 Via Bassetti");
18             assertEquals("01 Via Bassetti", tester.getUltimaPosizione(u), "se il setter funziona, "
19                         + "il getter ritorna lo stesso valore");
20
21         }
22
23     }
24
25 }
26
27
28 }
```

Test controllo se l'utente è registrato



```
1 package Test;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 class isUtenteRegistratoTest {
6
7     @Test
8     public void testIsUtenteRegistrato() {
9
10         ApplicazioneDataManager tester = new ApplicazioneDataManager();
11
12         //verifico se la verifica dell'utente se è registrato funziona
13         assertEquals(false, tester.isUtenteRegistrato(" "), "l'utente ' ' "
14                         + "non può essere stato registrato");
15
16     }
17
18 }
19
20
21 }
```

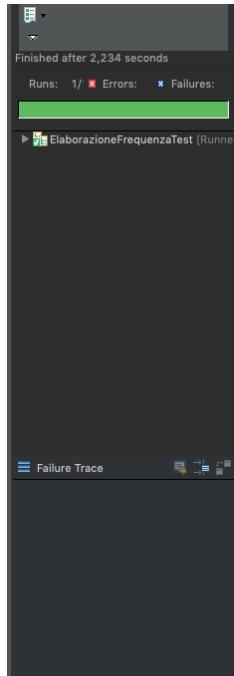
Test autenticazione utente



```
1 package Test;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 class autenticaUtenteTest {
6
7     @Test
8     public void testAutenticaUtente() {
9
10         ApplicazioneDataManager tester = new ApplicazioneDataManager();
11
12         if(! tester.isUtenteRegistrato("test")) {
13             //inserisco l'utente di test
14             tester.registraUtente("test", "test");
15
16             //verifico se l'autenticazione funziona
17             assertEquals(true, tester.autenticaUtente("test", "test"), "deve essere vero");
18
19         }
20
21     }
22
23 }
24
25
26
27 }
```

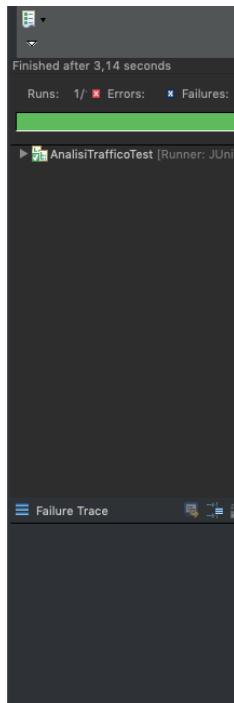
JUNIT TEST CENTRALINA STRADALE

Test per verificare che la centralina modifica la frequenza con la quale manda i dati al sistema in base all'intensità di traffico



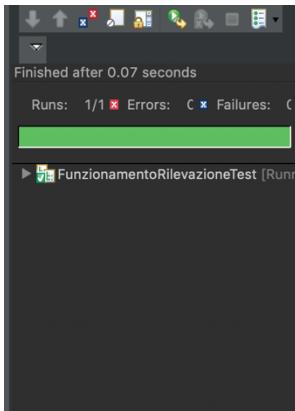
```
1 package Test;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 class ElaborazioneFrequenzaTest {
6
7     @Test
8     public void testFrequenza() throws RemoteException {
9         CentralinaStradaleASensoreSingolo tester =
10             new CentralinaStradaleASensoreSingolo("Via Prova", 50, Tipologia.SECONDARIA, SensoDiMarcia.AB);
11         tester.accendi();
12         Timestamp t=new Timestamp (System.currentTimeMillis());
13         tester.nuovaRilevazione(new Timestamp (System.currentTimeMillis()), ID.A);
14         Calendar c=Calendar.getInstance();
15         c.setTime(t);
16         c.set(Calendar.SECOND, c.get(Calendar.SECOND)+3);
17         t.setTime(c.getTimeInMillis());
18         Calendar c2=Calendar.getInstance();
19         t=new Timestamp (System.currentTimeMillis());
20         c2.setTime(t);
21         while(c.get(Calendar.SECOND)!=c2.get(Calendar.SECOND))
22         {
23             t=new Timestamp (System.currentTimeMillis());
24             c2.setTime(t);
25         }
26         tester.nuovaRilevazione(new Timestamp (System.currentTimeMillis()), ID.B);
27         tester.analizzaTraffico();
28         int risultato2=tester.getFrequenza();
29         Traffico risultato=tester.getTraffico();
30         assertEquals(Traffico.BLOCCATO, risultato);
31         assertEquals(1, risultato2);
32
33         tester.nuovaRilevazione(new Timestamp (System.currentTimeMillis()), ID.A);
34         tester.nuovaRilevazione(new Timestamp (System.currentTimeMillis()), ID.B);
35         tester.analizzaTraffico();
36         tester.nuovaRilevazione(new Timestamp (System.currentTimeMillis()), ID.A);
37         tester.nuovaRilevazione(new Timestamp (System.currentTimeMillis()), ID.B);
38         tester.analizzaTraffico();
39         risultato=tester.getTraffico();
40         risultato2=tester.getFrequenza();
41         assertEquals(Traffico.SCORREVOLE, risultato);
42         assertEquals(8, risultato2);
43
44         tester.nuovaRilevazione(new Timestamp (System.currentTimeMillis()), ID.A);
45         tester.nuovaRilevazione(new Timestamp (System.currentTimeMillis()), ID.B);
46         tester.analizzaTraffico();
47         tester.nuovaRilevazione(new Timestamp (System.currentTimeMillis()), ID.A);
48         tester.nuovaRilevazione(new Timestamp (System.currentTimeMillis()), ID.B);
49         tester.analizzaTraffico();
50         risultato=tester.getTraffico();
51         risultato2=tester.getFrequenza();
52         assertEquals(Traffico.SCORREVOLE, risultato);
53         assertEquals(8, risultato2);
54
55     }
56 }
57 }
```

Test calcolo del traffico



```
1 package Test;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 class AnalisiTrafficoTest {
6
7     @Test
8     public void testNuovoDato() throws RemoteException {
9         CentralinaStradaleASensoreSingolo tester =
10             new CentralinaStradaleASensoreSingolo("Via Prova", 50, Tipologia.SECONDARIA, SensoDiMarcia.AB);
11         tester.accendi();
12         Timestamp t=new Timestamp (System.currentTimeMillis());
13         tester.nuovaRilevazione(new Timestamp (System.currentTimeMillis()), ID.A);
14         Calendar c=Calendar.getInstance();
15         c.setTime(t);
16         c.set(Calendar.SECOND, c.get(Calendar.SECOND)+3);
17         t.setTime(c.getTimeInMillis());
18         Calendar c2=Calendar.getInstance();
19         t=new Timestamp (System.currentTimeMillis());
20         c2.setTime(t);
21         while(c.get(Calendar.SECOND)!=c2.get(Calendar.SECOND))
22         {
23             t=new Timestamp (System.currentTimeMillis());
24             c2.setTime(t);
25         }
26         tester.nuovaRilevazione(new Timestamp (System.currentTimeMillis()), ID.B);
27         tester.analizzaTraffico();
28         Traffico risultato=tester.getTraffico();
29         assertEquals(Traffico.BLOCCATO, risultato);
30
31         tester.nuovaRilevazione(new Timestamp (System.currentTimeMillis()), ID.A);
32         tester.nuovaRilevazione(new Timestamp (System.currentTimeMillis()), ID.B);
33         tester.analizzaTraffico();
34         risultato=tester.getTraffico();
35         assertEquals(Traffico.ELEVATO, risultato);
36
37         tester.nuovaRilevazione(new Timestamp (System.currentTimeMillis()), ID.A);
38         tester.nuovaRilevazione(new Timestamp (System.currentTimeMillis()), ID.B);
39         tester.analizzaTraffico();
40         risultato=tester.getTraffico();
41         assertEquals(Traffico.SCORREVOLE, risultato);
42
43         tester.nuovaRilevazione(new Timestamp (System.currentTimeMillis()), ID.A);
44         tester.nuovaRilevazione(new Timestamp (System.currentTimeMillis()), ID.B);
45         tester.analizzaTraffico();
46         risultato=tester.getTraffico();
47         assertEquals(Traffico.SCORREVOLE, risultato);
48
49         tester.nuovaRilevazione(new Timestamp (System.currentTimeMillis()), ID.A);
50         tester.nuovaRilevazione(new Timestamp (System.currentTimeMillis()), ID.B);
51         tester.analizzaTraffico();
52         risultato=tester.getTraffico();
53         assertEquals(Traffico.ASSENTE, risultato);
54
55     }
56 }
57 }
```

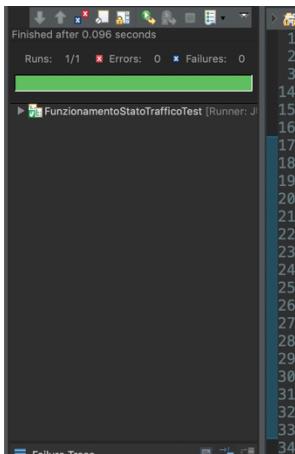
Test rilevazione nuovo dato



```
1 package Test;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 class FunzionamentoRilevazioneTest {
6
7     @Test
8     public void testRilevazione() throws RemoteException {
9         Rilevazione tester= new Rilevazione();
10        assertEquals(false,tester.getRilevato());
11        tester.setT1();
12        tester.setT2();
13        tester.setRilevato(true);
14        assertEquals(true,tester.getRilevato());
15    }
16}
```

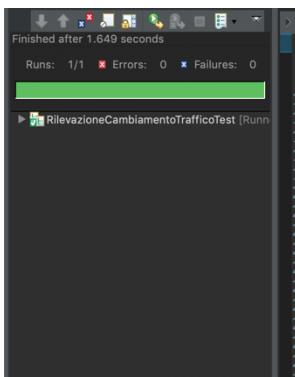
JUNIT TEST SISTEMA CENTRALE

Test rilevazione nuove notifiche



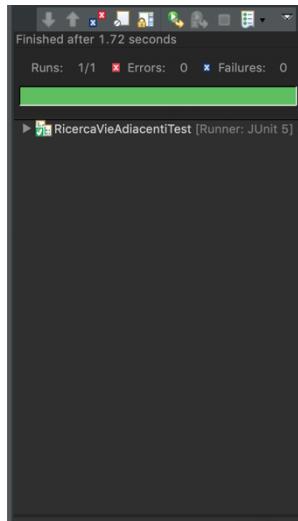
```
1 package Test;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 class RicercaNotificheTest {
6
7     @Test
8     public void testRilevaNotifiche() throws RemoteException {
9         GestoreNotifiche tester = new GestoreNotifiche();
10        GestoreTraffico gt=new GestoreTraffico();
11        GestoreClient gc = new GestoreClient();
12        gt.setGestoreClient(gc);
13        tester.setGestoreClient(gc);
14        tester.setGestoreTraffico(gt);
15        ArrayList<String> risultato = new ArrayList<String>();
16        risultato=tester.rilevaNotifiche("01 Via Fiori", 1, 200);
17        int nNotifiche=risultato.size();
18        Timestamp t1 = new Timestamp (System.currentTimeMillis());
19        gt.nuovoDato("Via Calvi", "BLOCCATO", t1, 4);
20        risultato=tester.rilevaNotifiche("01 Via Barone", 1, 200);
21        assertEquals(0 ,nNotifiche);
22        assertEquals("BLOCCATO! (precisione elevata)/Via Calvi" ,risultato.get(0));
23    }
24}
```

Test rilevazione cambiamento del traffico



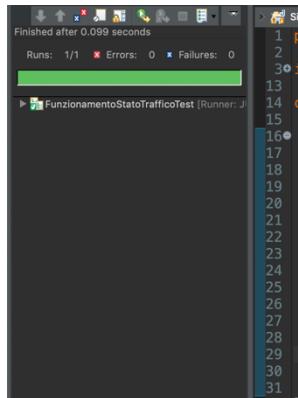
```
1 package Test;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 class RilevazioneCambiamentoTrafficoTest {
6
7     @Test
8     public void testRilevaCambiamentoTraffico() throws RemoteException {
9         GestoreTraffico tester=new GestoreTraffico();
10        GestoreClient gc = new GestoreClient();
11        Timestamp t1 = new Timestamp (System.currentTimeMillis());
12        tester.setGestoreClient(gc);
13        StatoTraffico tmp1= new StatoTraffico(0, Traffico.ELEVATO, 1, t1);
14        StatoTraffico tmp2= new StatoTraffico(0, Traffico.SCORREVOLE, 1, t1);
15        Evento risultato=tester.rilevaCambiamentoTraffico(tmp1, tmp2);
16        assertEquals(Evento.CALATOSCORREVOLE,risultato);
17    }
18}
```

Test rilevazione vie adiacenti



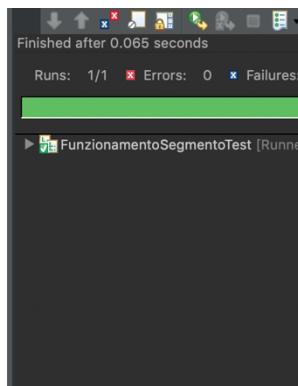
```
1 package Test;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 class RicercaVieAdiacentiTest {
6
7     @Test
8     public void testRilevaVieDintorni() throws RemoteException {
9         GestoreNotifiche tester = new GestoreNotifiche();
10        GestoreTrafico gt=new GestoreTrafico();
11        GestoreClient gc = new GestoreClient();
12        gt.setGestoreClient(gc);
13        tester.setGestoreClient(gc);
14        tester.setGestoreTrafico(gt);
15        ArrayList<String> risultato = new ArrayList<String>();
16        tester.rilevaVieintorni("01 Via Barone", 200, risultato);
17        assertEquals("01 Via Calvi",risultato.get(0));
18        assertEquals("01 Via Balestra",risultato.get(1));
19        assertEquals("03 Via Barone",risultato.get(2));
20        assertEquals("02 Via Calvi",risultato.get(3));
21        assertEquals("02 Via Barone",risultato.get(4));
22    }
23
24 }
```

Test creazione dato completo stato del traffico



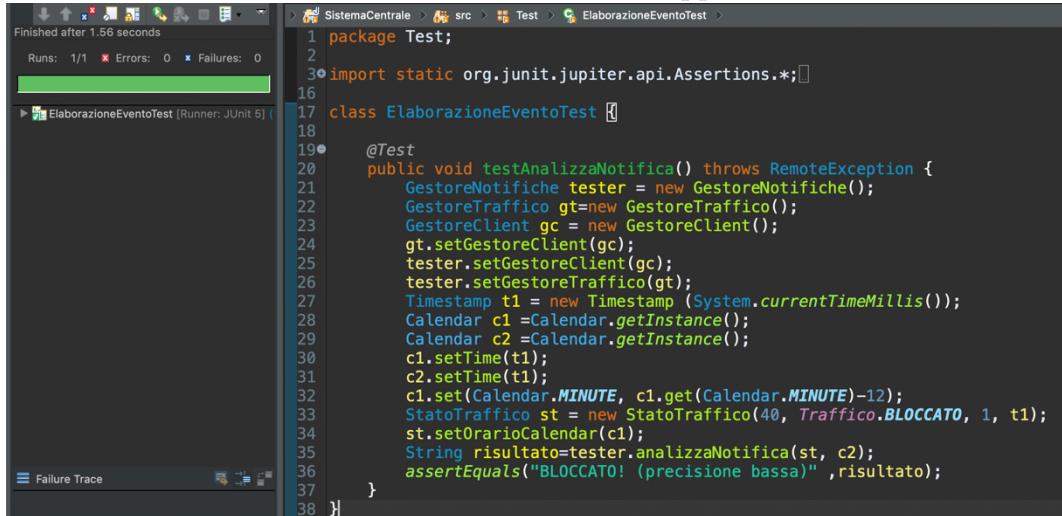
```
1 package Test;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 class FunzionamentoStatoTraficoTest {
6
7     @Test
8     public void testStatoTrafico()throws RemoteException {
9         StatoTraficoPeriodico tester =new StatoTraficoPeriodico();
10        DatoCompleto dc=new DatoCompleto();
11        dc.setMittente(Mittente.APPLICAZIONE);
12        dc.setOraCorrente();
13        dc.setPosizione("Via Barone");
14        dc.setTrafico(Traffico.ELEVATO);
15        tester.setDato(dc);
16        tester.setPrimoOraCalendar(tester.getOra());
17        tester.diminuisceCorrettezza(Mittente.CENTRALINASTRADALE, Traffico.SCORREVOLE);
18        int risultato=tester.getCorrettezza();
19        Traffico risultato2=tester.getTrafico();
20        assertEquals(4,risultato);
21        assertEquals(Traffico.SCORREVOLE,risultato2);
22    }
23
24 }
```

Test estrazione via da coppia via e segmento



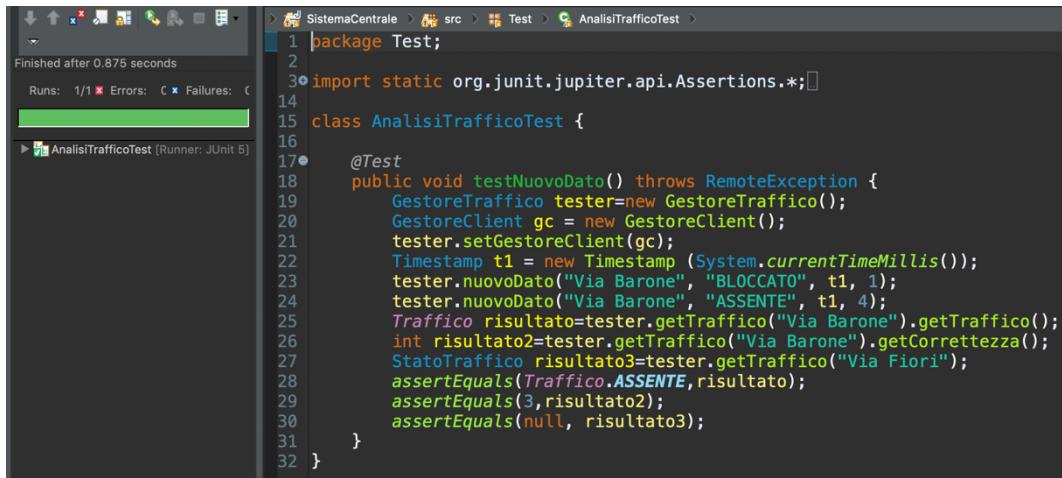
```
1 package Test;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 class FunzionamentoSegmentoTest {
6
7     @Test
8     public void testSegmento()throws RemoteException {
9         Segmento tester=new Segmento();
10        tester.setLunghezza(20);
11        tester.setSegmento("01 Via Barone");
12        String risultato=tester.getVia();
13        assertEquals("Via Barone",risultato);
14    }
15
16 }
```

Test ricerca se vi sono notifiche da mandare all'applicazione mobile



```
1 package Test;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5
6 class ElaborazioneEventoTest {
7
8     @Test
9     public void testAnalizzaNotifica() throws RemoteException {
10         GestoreNotifiche tester = new GestoreNotifiche();
11         GestoreTraffic gt=new GestoreTraffic();
12         GestoreClient gc = new GestoreClient();
13         gt.setGestoreClient(gc);
14         tester.setGestoreClient(gc);
15         tester.setGestoreTraffic(gt);
16         Timestamp t1 = new Timestamp (System.currentTimeMillis());
17         Calendar c1 =Calendar.getInstance();
18         Calendar c2 =Calendar.getInstance();
19         c1.setTime(t1);
20         c2.setTime(t1);
21         c1.set(Calendar.MINUTE, c1.get(Calendar.MINUTE)-12);
22         StatoTraffic st = new StatoTraffic(40, Traffic.BLOCCATO, 1, t1);
23         st.setOraioCalendar(c1);
24         String risultato=tester.analizzaNotifica(st, c2);
25         assertEquals("BLOCCATO! (precisione bassa)", risultato);
26     }
27 }
28 }
```

Test nuovo dato sistema centrale



```
1 package Test;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5
6 class AnalisiTrafficTest {
7
8     @Test
9     public void testNuovoDato() throws RemoteException {
10         GestoreTraffic tester=new GestoreTraffic();
11         GestoreClient gc = new GestoreClient();
12         tester.setGestoreClient(gc);
13         Timestamp t1 = new Timestamp (System.currentTimeMillis());
14         tester.nuovoDato("Via Barone", "BLOCCATO", t1, 1);
15         tester.nuovoDato("Via Barone", "ASSENTE", t1, 4);
16         Traffic risultato=tester.getTraffic("Via Barone").getTraffic();
17         int risultato2=tester.getTraffic("Via Barone").getCorrettezza();
18         StatoTraffic risultato3=tester.getTraffic("Via Fiori");
19         assertEquals(Traffic.ASENTE, risultato);
20         assertEquals(3, risultato2);
21         assertEquals(null, risultato3);
22     }
23 }
24 }
```