

# **Documento di progetto**

1. Traccia progetto	Pag.2
2. Analisi dei requisiti del sistema e studio di fattibilità	Pag. 3
3. Design UML	Pag. 7
4. Implementazione Java	Pag. 14

# Traccia 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).

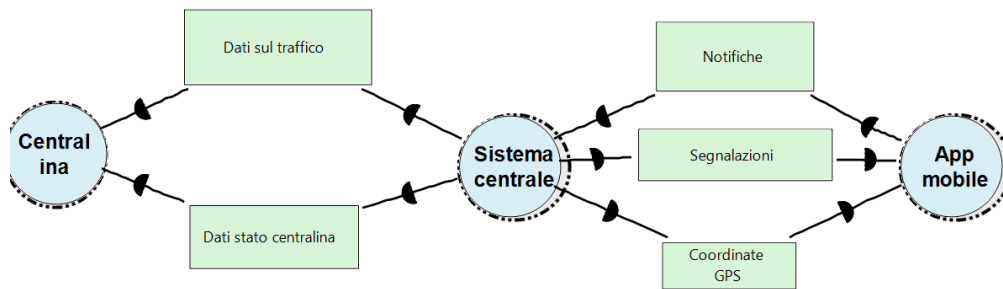
E' possibile raffinare i requisiti ed aggiungere ipotesi e assunzioni sul contesto, sensate e in linea con quanto indicato nei requisiti. Tali estensioni devono essere esplicitamente riportate nella documentazione di progetto (sezione specifica requisiti).

## Analisi dei requisiti del sistema e studio di fattibilità

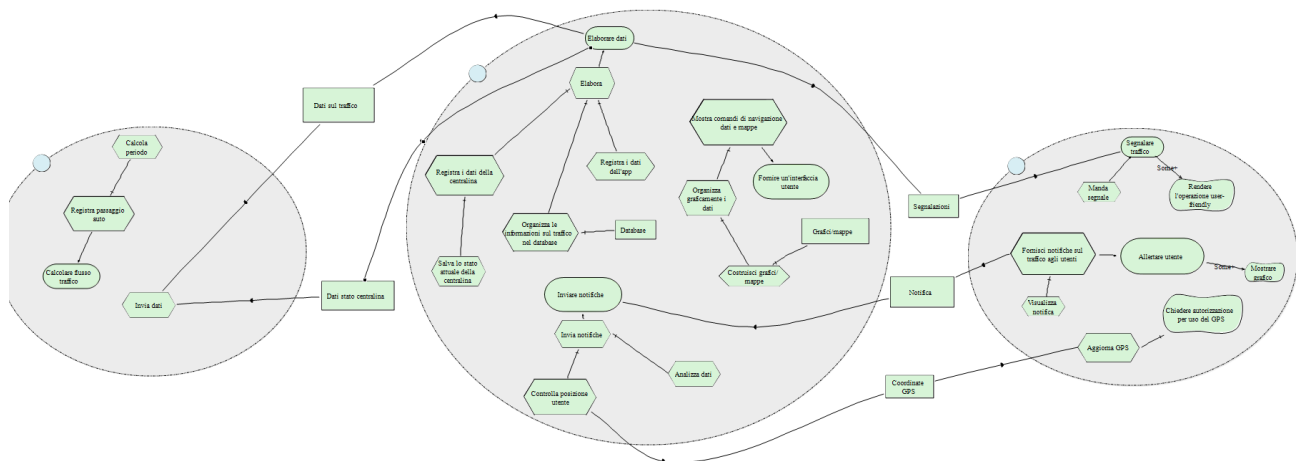
Il nostro progetto prevede l'implementazione di un software per il controllo del traffico cittadino. Esso consiste in un'elaborazione dei dati forniti dai sottosistemi (centraline e app mobili) allo scopo di ottenere una visione ampia del flusso stradale e di fornire eventuali informazioni sullo stesso agli utenti dell'app mobile. Uno scopo secondario del progetto prevede inoltre l'applicabilità della piattaforma a diversi tessuti cittadini.

L'analisi del problema si è evoluta dalla stesura del *data dictionary*, al fine di distinguere i principali componenti del software sotto forma di unità di informazioni. Si è prestata nel particolare molta attenzione a quest'ultime risorse per renderle il più universali possibile e per ottenere una maggiore efficienza. Proprio a questo fine abbiamo previsto l'astrazione del dato *informazione*, i cui attributi (posizione, data e ora) sono fondamentali al sistema per poter organizzare i dati in un contesto spazio-temporale. Si è inoltre precisata la presenza di una classe dati *utente* che identifichi gli user dell'app e le loro segnalazioni.

<u>Nome</u>	<u>Alias</u>	<u>Descrizione</u>	<u>Esempi</u>	<u>Attributi</u>	<u>Relazioni</u>	<u>Componenti</u>	<u>Supertipo</u>	<u>Sottotipo</u>
<b><u>Sistema centrale</u></b>	Calcolatore	Elaboratore e registratore dei dati inviati dai sottosistemi. Usufruisce di un database per organizzare i dati e di un gestore utenti per controllare la posizione dei clienti.		Database informativo Stato centraline Gestore utenti	Informazione	Segnala Ricevi Registra dati Elabora dati Stampa grafico		
<b><u>Informazione</u></b>		Dato scambiato tra sistema e sottosistemi		Posizione Data e ora Stato del sistema Livello di traffico	Sistema centrale	Invia a sistema		Centralina stradale App mobile
<b><u>Informazione e centralina stradale</u></b>		Informazione inviata dalla centralina al sistema con periodicità proporzionale al flusso di traffico.	"Posizione: 45° 48' 36,9 '' N 9° 05' 10,1'' E Traffico: 3"	Flusso di traffico		Calcolo periodo invio segnale	Informazione	
<b><u>Informazione e app mobile</u></b>	Notifiche Segnalazione da utente	Informazione scambiata fra sistema centrale e utente e viceversa. Nel caso della notifica il livello di traffico viene preso dal database del sistema; invece, nel caso della segnalazione si tratta del flusso di traffico percepito dall'utente.	Notifica: "Attento! C'è fila!"  Segnalazione: Posizione: 45° 48' 36,9 '' N 9° 05' 10,1'' E Tipo: Coda	Tipo di segnalazione e Utente	Utente	Ricevi Notifica a video	Informazione	
<b><u>Utente</u></b>	User Cliente app mobile	Insieme delle informazioni identificative dell'utilizzatore dell'app mobile	E-mail: <a href="mailto:mariorossi@mail.com">mariorossi@mail.com</a> Password: *****	E-mail Password	Informazione app mobile	Crea nuovo utente Cambia credenziali Elimina utente		
<b><u>Livello di traffico</u></b>	Flusso di traffico	È un indicatore del traffico percepito o registrato. Va da 0 (traffico nullo o irrilevante) a 5 (traffico elevato o congestione).	Livello di traffico: 0.		Informazione Sistema centrale			

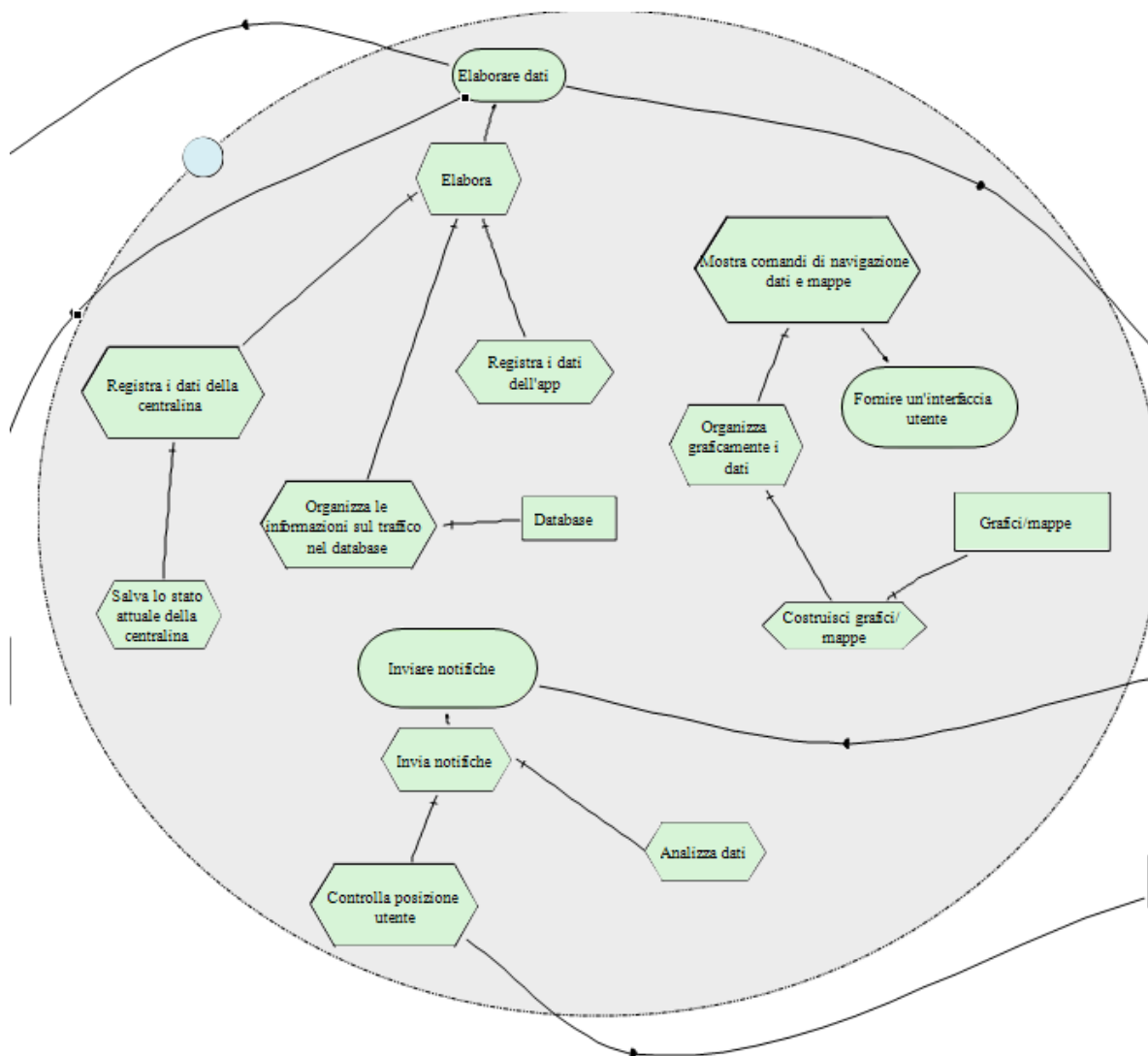


Nella creazione del *goal diagram*, invece, l'attenzione si è rivolta alle funzioni specifiche dei singoli sistemi, tenendo conto delle dipendenze che intercorrono tra di loro. Allo scopo di osservare proprio queste relazioni lo svolgimento del grafico è partito dall'SDM che è poi stato ampliato nel microscopico in SRM. In particolare abbiamo voluto evidenziare la presenza di risorse condivise tra gli attori, quali ad esempio le segnalazioni o le coordinate GPS, anche con l'intento di tracciare le basi per una dipendenza temporale e sequenziale tra i task.

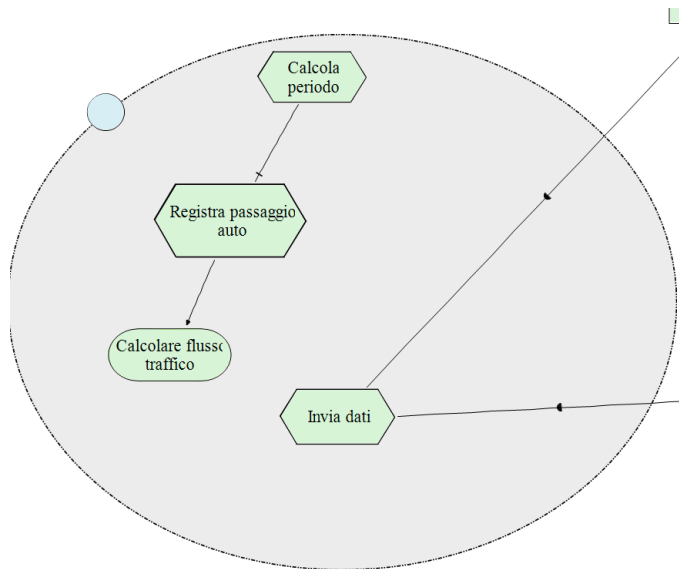


Il sistema è stato suddiviso in tre aree di competenza distinte.

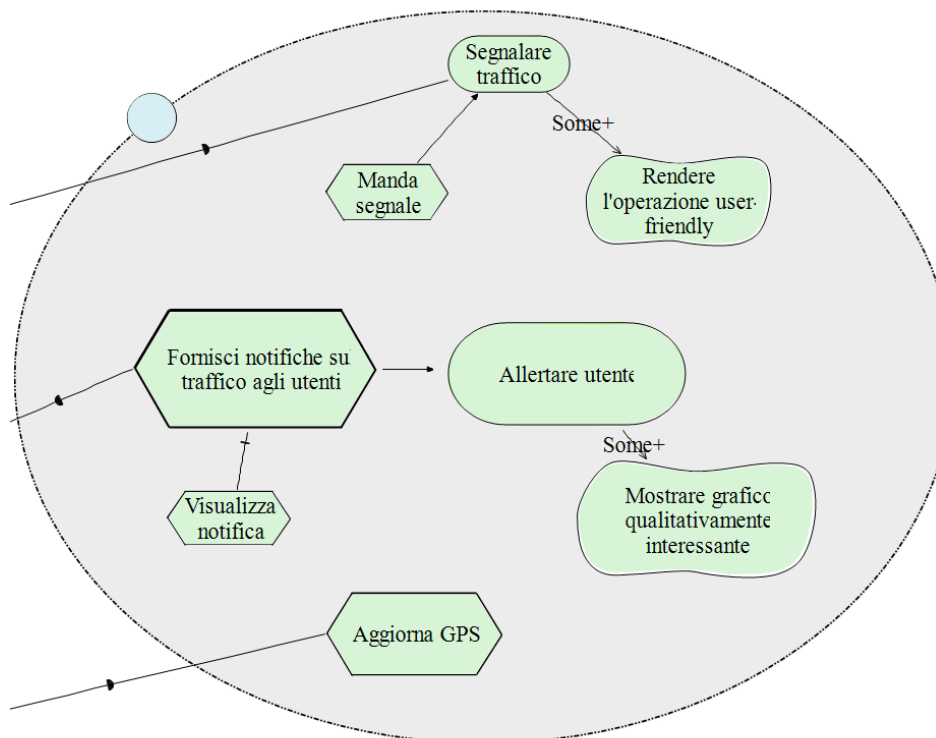
La principale, ovvero il sistema centrale, è il responsabile della catalogazione dei dati e della loro organizzazione nel database. Esso fornisce un'interfaccia utente da cui è possibile, attraverso comandi di navigazione, accedere ai dati, alla loro rappresentazione grafica e allo stato dei sottosistemi. Quest'ultimo viene aggiornato periodicamente attraverso l'invio di dati sul proprio funzionamento da parte della centralina e attraverso l'invio di coordinate GPS da parte dell'app mobile.



La centralina è capace di registrare il flusso di traffico e segnalarlo al sistema. Per fare ciò calcola il periodo di invio del dato a seconda del flusso di traffico.



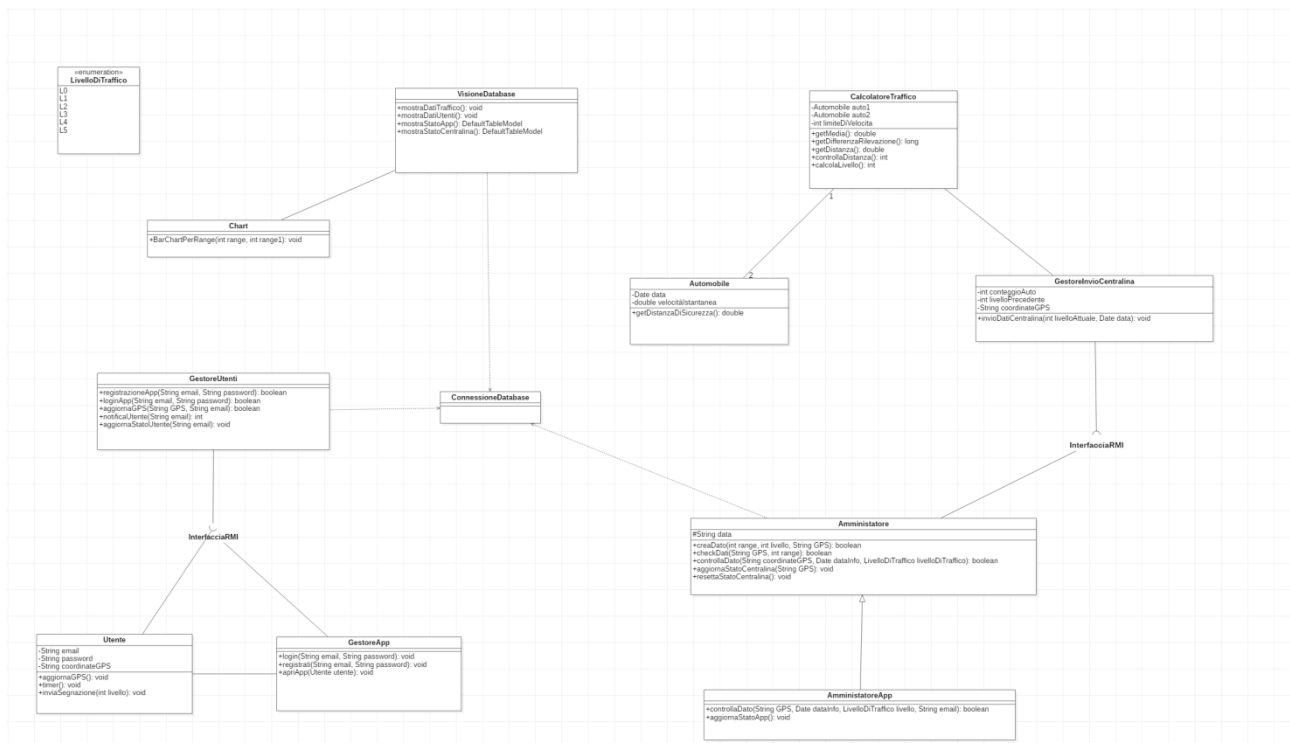
L'applicazione mobile funge da strumento di segnalazione della posizione dell'utente, che tramite la sua interfaccia può informare il sistema centrale della presenza di code. A sua volta il sistema può notificare eventi di traffico all'utente tramite l'app essendo già a conoscenza della sua posizione. Per fare ciò abbiamo ipotizzato la condivisione delle coordinate GPS dell'auto da parte dell'app con frequenza fissa previa autorizzazione dell'utente.



# Design UML

Nella stesura del progetto UML si è cercato di concentrarsi sulle parti dell'elaborato che necessitassero di più chiarezza; nello specifico si è voluto affrontare il problema dello scambio dei dati evidenziandone i passaggi con l'ausilio di diagrammi come il Sequence diagram o l'Activity diagram. Un altro obiettivo è stato quello di cercare di mantenere l'uniformità e la coerenza tra i grafici progettando per primo il Class diagram perché ci si potesse riferire ad esso nella stesura di tutti gli altri diagrammi.

## CLASS DIAGRAM



Per formare il Class diagram si è deciso di partire da tre grandi entità fondamentali ovvero il sistema, la centralina e l'app mobile. Ciascuna di esse è stata suddivisa in classi per distribuire al meglio compiti e funzioni all'interno del programma.

In primo luogo è stata inserita la classe *CalcolatoreTraffico* la quale, attraverso due oggetti di tipo *Automobile* e un numero intero (valore della velocità limite relativo alla strada su cui verrà posta fisicamente la centralina), calcola il livello di traffico. Ciò è reso possibile dalla comparazione tra la distanza reale tra le due auto (ricavata nel metodo *GetDifferenzaRilevazione*) e quella di sicurezza prevista dal codice della strada (ricavata nel metodo *GetDistanza*).

I dati di tipo *LivelloDiTraffico* (classe condivisa da tutti inserita come <<enumeration>>) calcolati in precedenza vengono poi smistati nel *GestoreInvioCentralina*. Questa classe si occupa di scegliere quando e quali dati poter inoltrare all'amministratore del sistema in relazione al livello di traffico precedente e al numero di auto passate in quel momento dall'ultimo invio a sistema.

Tramite metodi di controllo (*checkDati*) la classe *Amministratore*, attraverso *ConnessioneDatabase*, si accerta che non vi siano dati già esistenti per coordinate e range di tempo. Eventualmente crea un nuovo dato o aggiorna il database.

Rispettivamente accade per la parte relativa all'App Mobile.

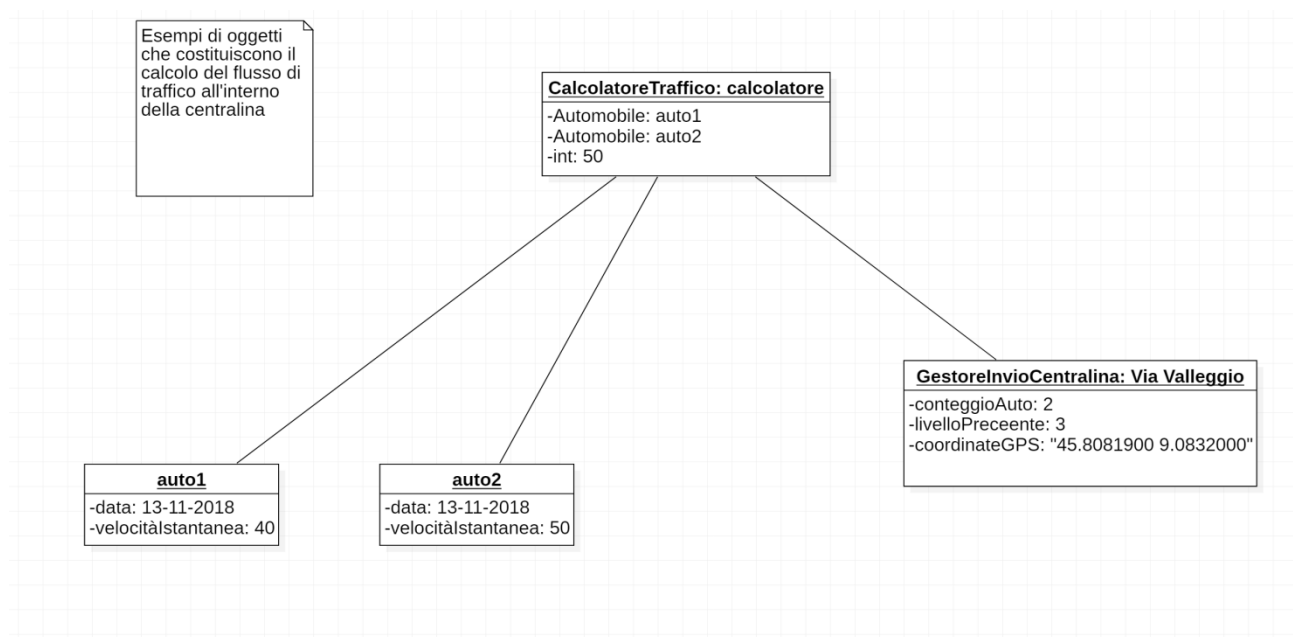
Si è voluto specificare una classe *Utente* che identifichi un cliente dell'app mobile tramite credenziali fisse (email e password) e attributi variabili (coordinateGPS). La classe si occupa inoltre dell'invio di una segnalazione volontaria sul traffico e dell'aggiornamento continuo della posizione stradale dell'utente. Le operazioni relative al login e alla registrazione dell'utente sono rese possibili dalle classi *GestoreApp* e *GestoreUtenti* le quali tengono traccia dei clienti e della loro posizione GPS.

In particolare la classe *GestoreUtenti* condivide e scambia dati con il database attraverso *ConnessioneDatabase* al fine di inviare notifiche sulla viabilità agli utenti o segnalare al sistema, da parte di questi, eventuali livelli di traffico significativi.

In ultimo si è poi voluta inserire la classe *VisioneDatabase* che mostra le informazioni contenute all'interno del database. Vi sono dunque due metodi, uno per la visione dei dati relativi agli utenti (*mostraDatiUtenti*), l'altro per la consultazione di quelli relativi al traffico (*mostraDatiTraffico*). Quest'ultimo prevede la classe *Chart*, che, inseriti due range temporali, mostra in un grafico a barre la situazione stradale relativa a ciascuna coordinata GPS.

*mostraStatoApp* e *mostraStatoCentralina* visualizzano, invece, utenti e centraline attivi.

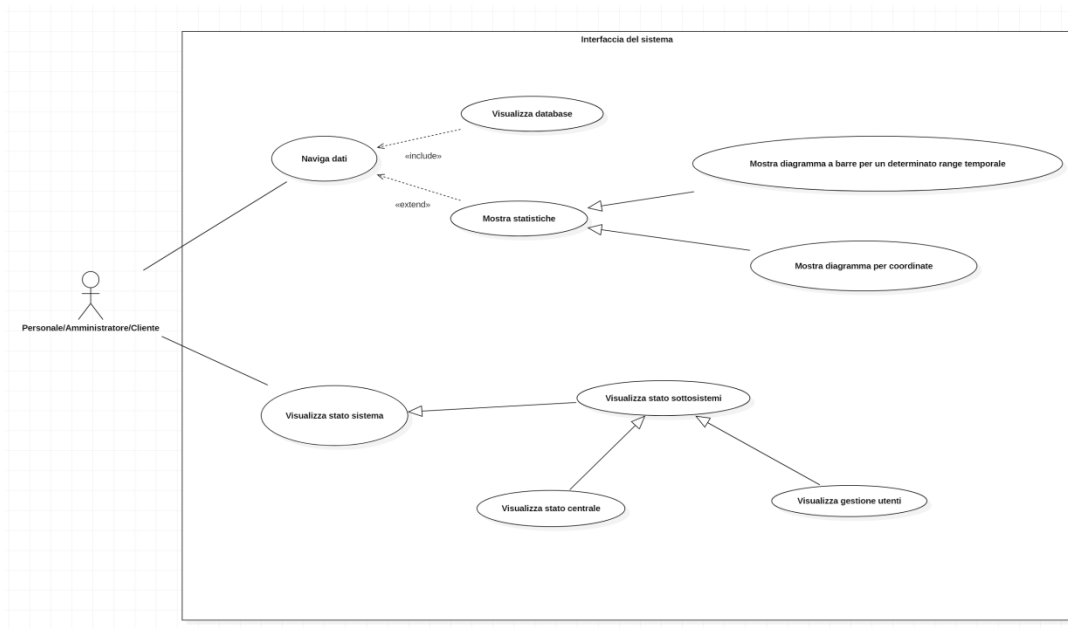
## OBJECT DIAGRAM



L'Object diagram si occupa di esemplificare alcune delle classi presentandole come oggetti.

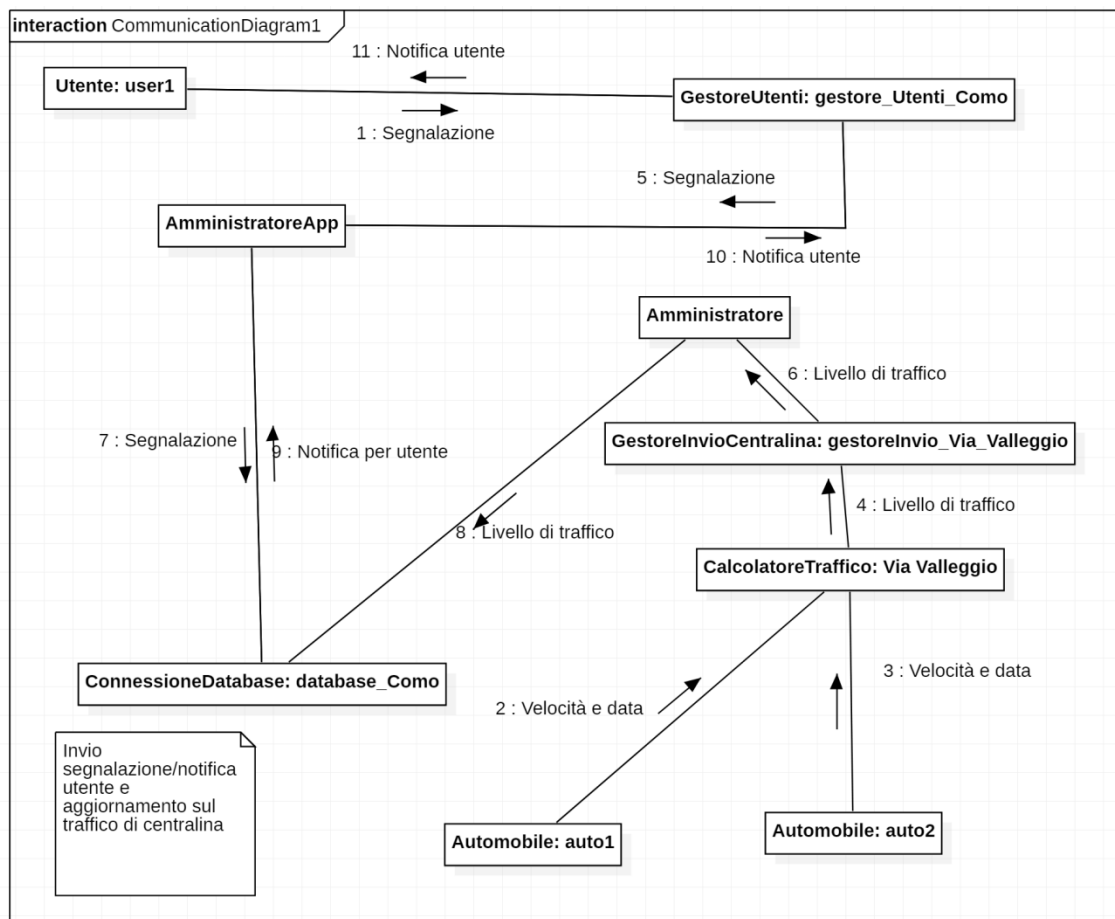


## USE CASE



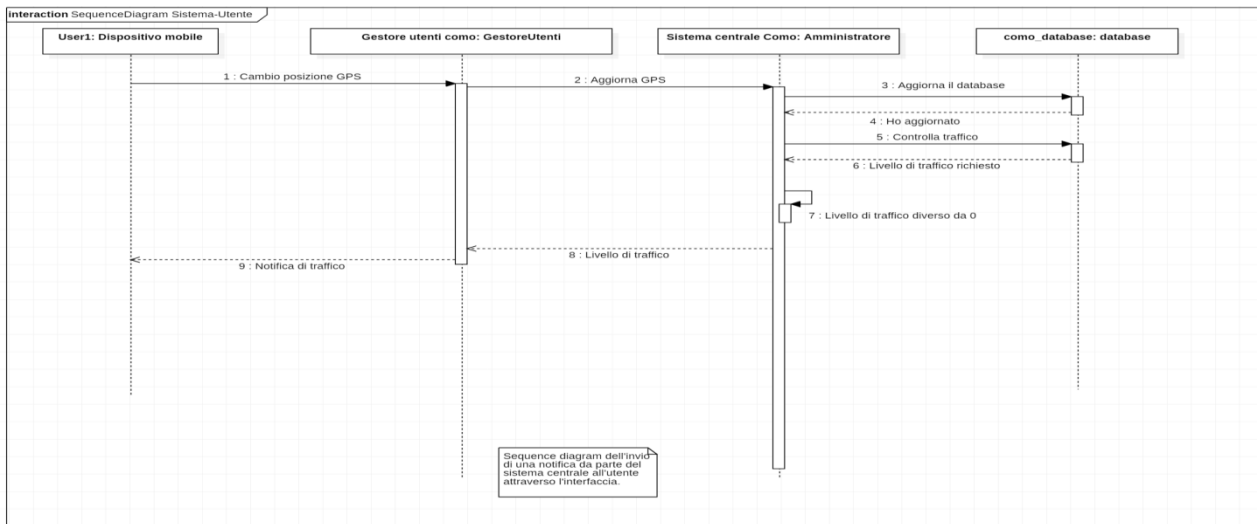
La decisione di utilizzare uno Use Case diagram, per mostrare le operazioni disponibili all'amministratore del server, è nata dal voler rendere il più chiaro possibile a quali funzionalità egli abbia accesso e come esse si presentino all'interno della sua interfaccia.

## COMMUNICATION DIAGRAM



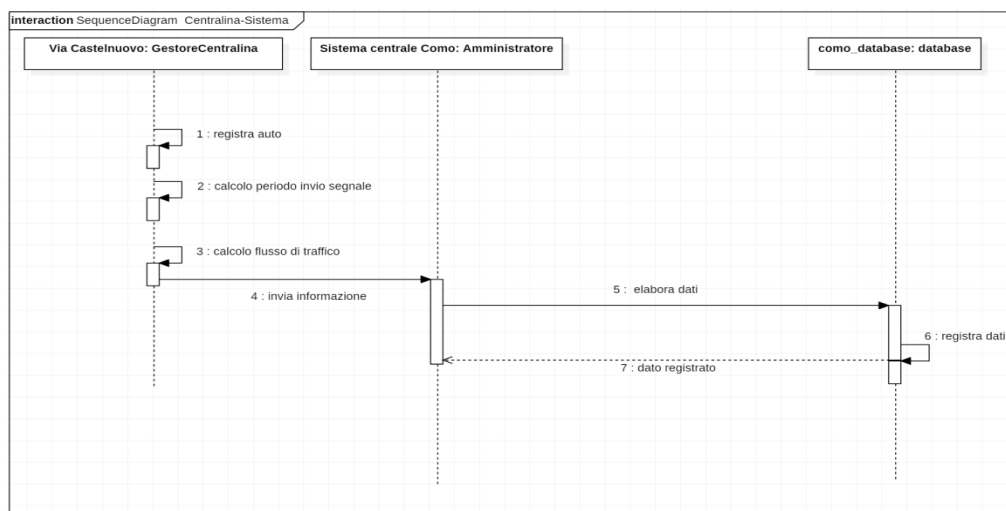
Con il Communication diagram si è voluto mostrare come i dati sul traffico, provenienti dall'utente e dal calcolatore della centralina, arrivino al database. Questi, elaborati dall'amministratore, verranno poi restituiti all'utente sotto forma di notifica.

## SEQUENCE DIAGRAM SISTEMA-UTENTE



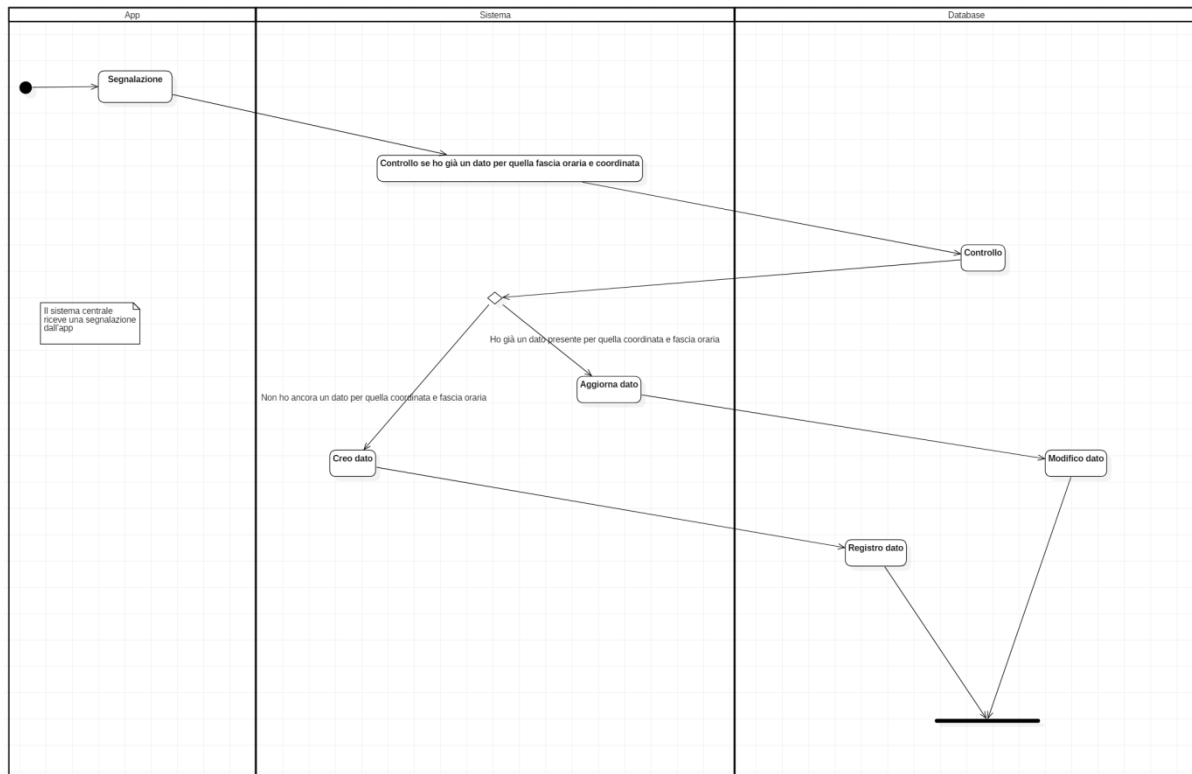
Come già accennato si è voluto utilizzare i Sequence diagram per mostrare il passaggio delle informazioni. Nello specifico in questo diagramma si vuole mostrare la sequenza di passaggi che intercorrono tra il sistema (*Amministratore*) e le classi ad esso collegato prima dell'invio di una notifica all'app mobile (*Utente*). Esso viene ricevuto sotto forma di server push e mostrato a video all'utente.

## SEQUENCE DIAGRAM CENTRALINA-SISTEMA



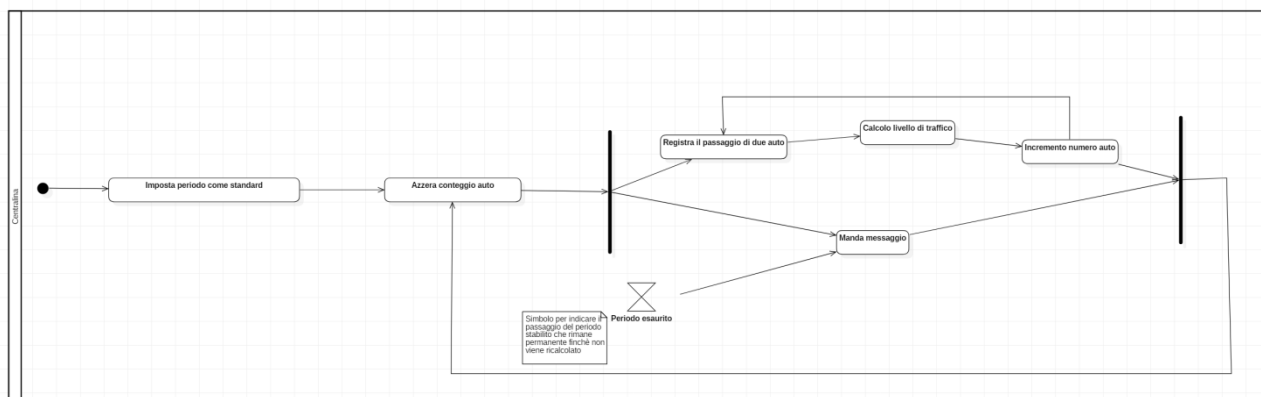
In questo diagramma si è analizzato l'invio delle informazioni riguardo il livello di traffico registrato da parte della centralina e la susseguente registrazione della stessa all'interno del Database.

## ACTIVITY DIAGRAM SISTEMA CENTRALE



Nell'activity diagram del sistema centrale abbiamo analizzato l'eventualità di una segnalazione da parte di un utente app mobile. Si è evidenziato il passaggio dell'informazione tra App, Sistema e Database.

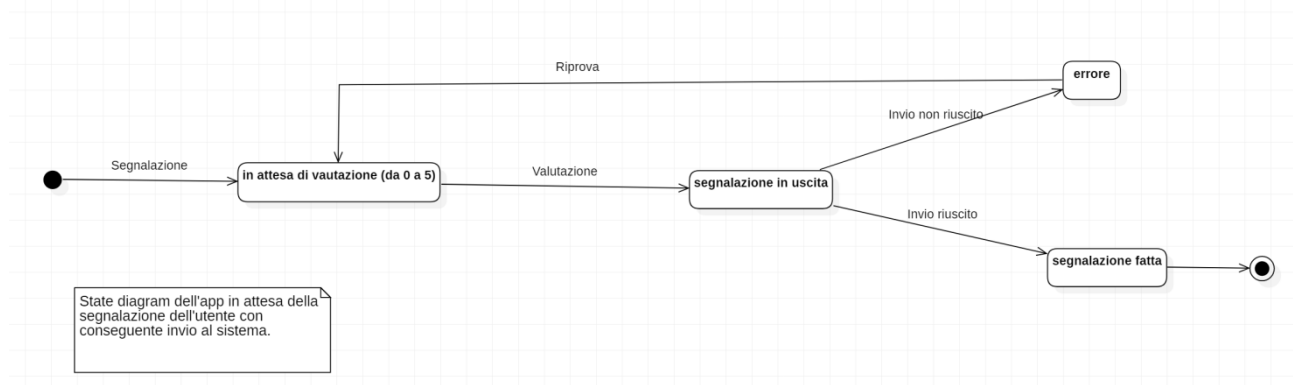
## ACTIVITY DIAGRAM CENTRALINA



Si è poi deciso di utilizzare un ulteriore Activity diagram per chiarificare il comportamento della centralina e il modo in cui essa aggiorni la variabile periodo (indicante il tempo che intercorre tra una segnalazione e la successiva). All'attivazione della centralina, infatti, il periodo viene impostato con un tempo standard, e rimarrà invariato finché il passaggio di un'auto non farà riaggiornare il conteggio auto della centralina (impostato inizialmente a 0) e di conseguenza anche il periodo.

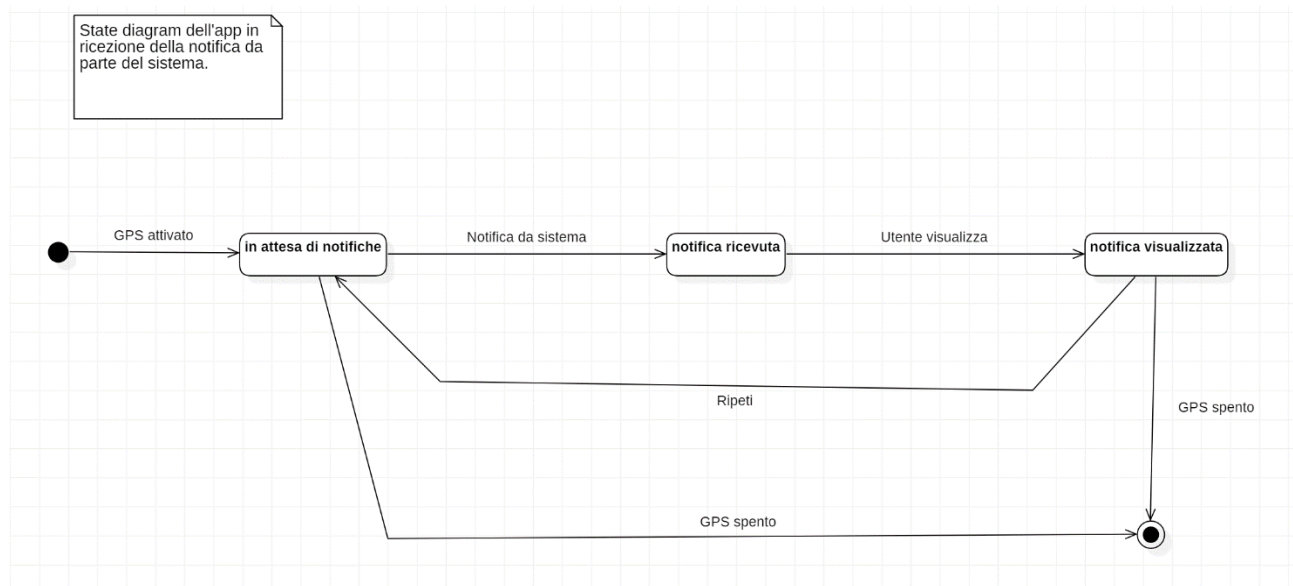
Il simbolo della clessidra (accept time event action) sta infatti ad indicare il passaggio del tempo previsto per l'invio della segnalazione.

## STATE DIAGRAM APP IN SEGNALAZIONE



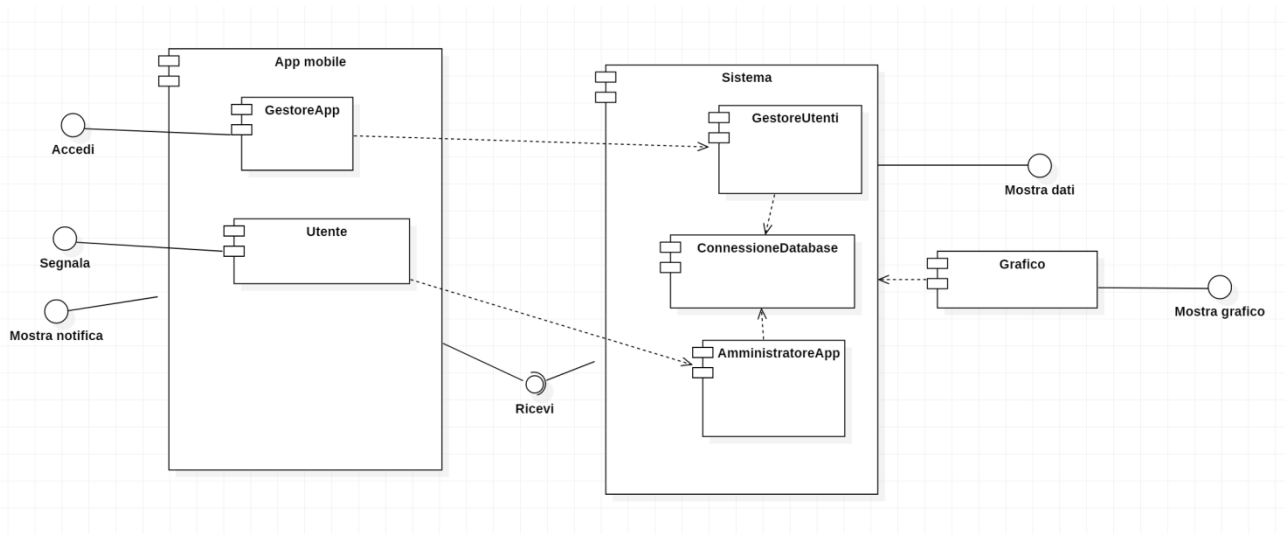
In questo State diagram vengono messi in evidenza i vari stati assunti dall'applicazione mobile quando l'utente richiede di fare una segnalazione (si è omessa, in questo caso, l'interfaccia grafica attraverso la quale avviene ciò). Si è scelto di mostrare anche la richiesta, da parte dell'app all'utente, di specificare la gravità del traffico da egli incontrato (livello di traffico) e si è considerata l'eventualità di un errore nell'invio della segnalazione, al fronte della quale l'app rinnova la richiesta (nel caso la gravità della congestione fosse cambiata nel frattempo) e ritenta l'invio.

## STATE DIAGRAM APP IN RICEZIONE



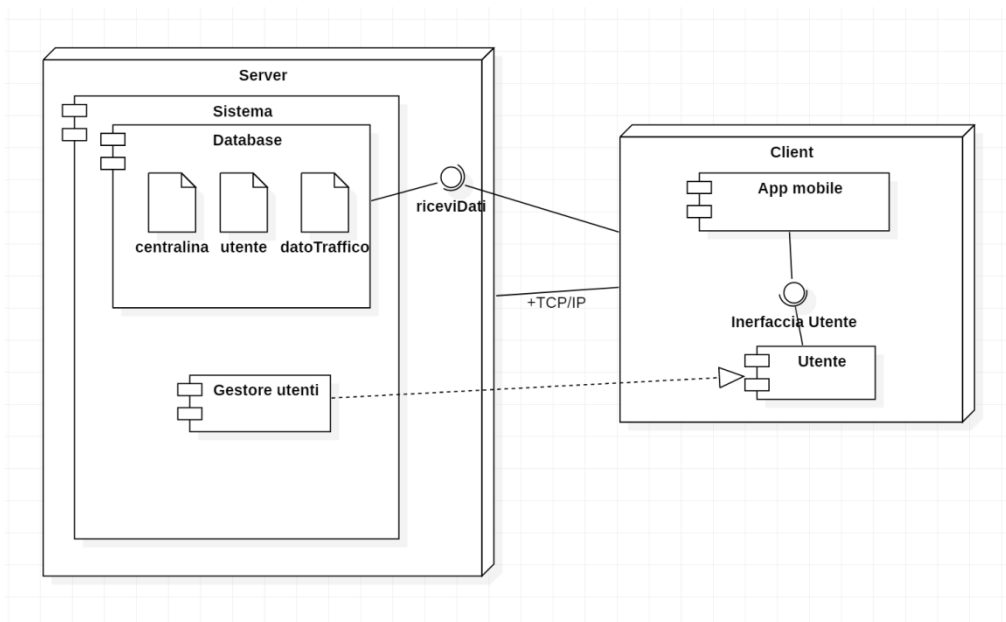
La seconda funzione principale dell'app è di ricevere notifiche push provenienti dal sistema; questo diagramma mostra gli stati da essa assunti dall'attivazione del GPS, quando è cioè possibile per il Sistema notificare l'utente.

## COMPONENT DIAGRAM



Il Component diagram è stato utile per mappare l'interezza del microcosmo Sistema-App, evidenziando la presenza delle interfacce e anticipando la distinzione Server-Client analizzata successivamente nel Deployment diagram.

## DEPLOYMENT DIAGRAM



# **Implementazione Java**

La realizzazione della terza parte del progetto si è evoluta a partire dalla creazione di un database MySQL in cui abbiamo registrato i dati relativi al traffico, all'utente e alle centraline.

All'interno del progetto Java Sistema abbiamo sviluppato il gruppo di classi relative alla gestione di questi dati. La classe che implementa l'effettiva connessione con il database è `ConnessioneDatabase` che utilizza i metodi `executeQuery` e `executeUpdate`.

I comandi SQL sono invece inizializzati come `String` dalla classe `Amministratore` e dalla sua sottoclasse `AmministratoreApp`.

Queste due classi si occupano di raccogliere i dati provenienti dalla Centralina e dall'App e di salvarli all'interno del database. Per fare ciò esse fanno utilizzo dei metodi `creaDato` e `checkDati`, che si occupa di controllare se le informazioni ricevute vadano viste come aggiornamenti di dati già esistenti nel database.

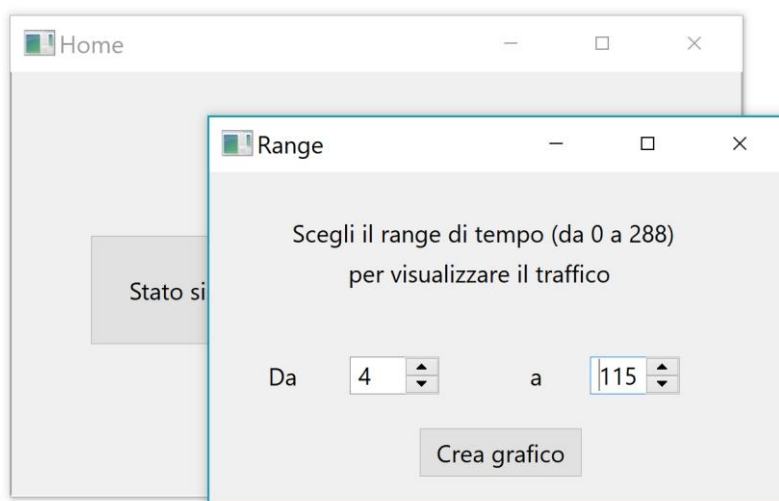
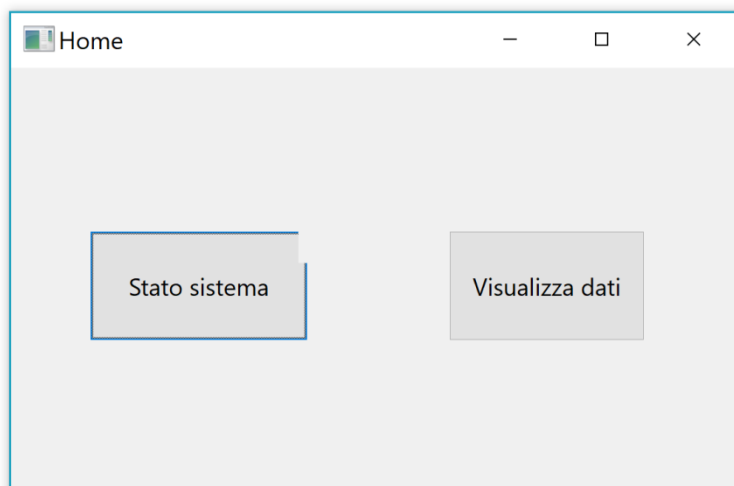
In `Amministratore` sono inoltre presenti i metodi `aggiornaStatoCentralina` e `resettaStatoCentralina`. Il primo aggiorna la variabile stato contenuta all'interno del database ogni qual volta si riceve un'informazione dalla centralina. Il secondo va invocato all'avvio del sistema: esso inizializza un timer che periodicamente resetta lo stato delle centraline al fine di ottenere una visione veritiera di quali di esse siano attive.

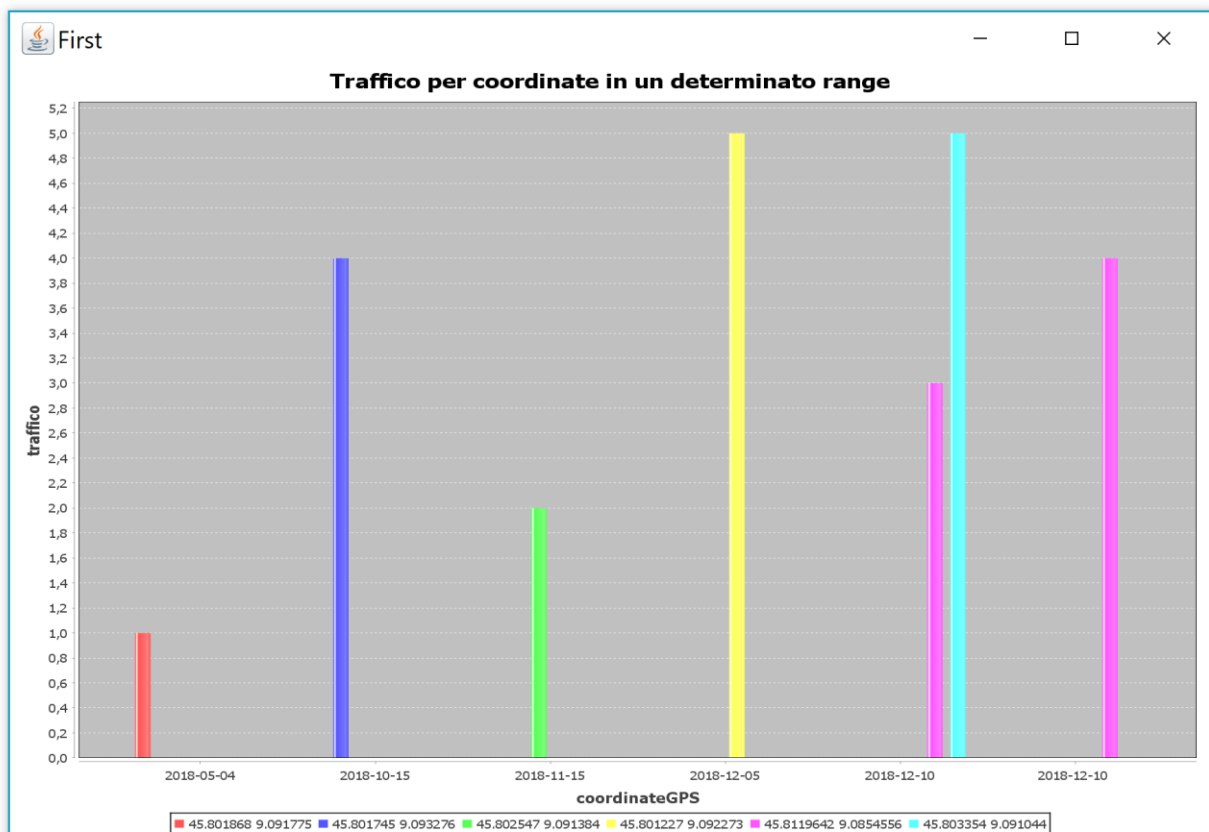
`AmministratoreApp` implementa un simile timer nel metodo `aggiornaStatoApp` che resetta a sua volta la variabile stato relativa agli utenti.

Anche la classe `GestoreUtenti` utilizza i metodi di `ConnessioneDatabase` per effettuare modifiche e inserimenti all'interno della tabella utente. I suoi metodi `registrazioneApp`, `loginApp` e `aggiornaGPS` sono infatti richiamati tramite RMI dall'utente dell'app. Inoltre la classe implementa `notificaUtente` che, quando viene chiamato, restituisce un `int` corrispondente al livello di traffico presente nelle vicinanze dell'utente.

La classe `VisioneDatabase` è stata creata con lo scopo di estrarre i dati dal database in modo da renderli utilizzabili dall'interfaccia grafica. I metodi `mostraStatoApp` e `mostraStatoCentraline` restituiscono infatti un `DefaultTableModel` che verrà poi realizzato graficamente da `GraficaAdmin`.

GraficaAdmin rappresenta la schermata iniziale di un ipotetico amministratore del software. Da essa si può accedere alle due tabelle contenenti le informazioni sulle centraline e sugli utenti correntemente attivi e ai grafici relativi al traffico registrato.





Messaggio				
email	pass	coordinateGPS	stato	
utente1@mail.com	utente1		true	
utente4@mail.com	utente4		true	

La seconda parte della realizzazione del progetto ha coinvolto l'implementazione del software della centralina. Il progetto relativo si compone di una parte di simulazione del passaggio delle auto e di una parte di invio delle informazioni. Quest'ultima è realizzata dalla classe `CalcolatoreTraffico` che, usando due oggetti di tipo `Automobile`, calcola il livello di traffico istantaneo. Per fare ciò essa confronta la distanza di sicurezza prevista dal codice della strada con la distanza reale tra le due auto. Se queste risultano troppo vicine tra loro viene effettuato un controllo sulla loro velocità utilizzando dei parametri basati sul limite di velocità. In questo modo viene assegnato un livello che rispecchi abbastanza fedelmente il traffico reale.

La simulazione viene effettuata attraverso tre metodi presenti nella classe `SimulatoreCentralina`, che rappresentano tre fasce diverse di traffico (`trafficoBasso`, `trafficoMedio`, `trafficoAlto`). All'interno di questi metodi vengono scelti randomicamente velocità e tempo intercorso tra il passaggio delle due auto attenendosi a dei parametri prestabiliti per fascia di traffico. Questi metodi vengono chiamati

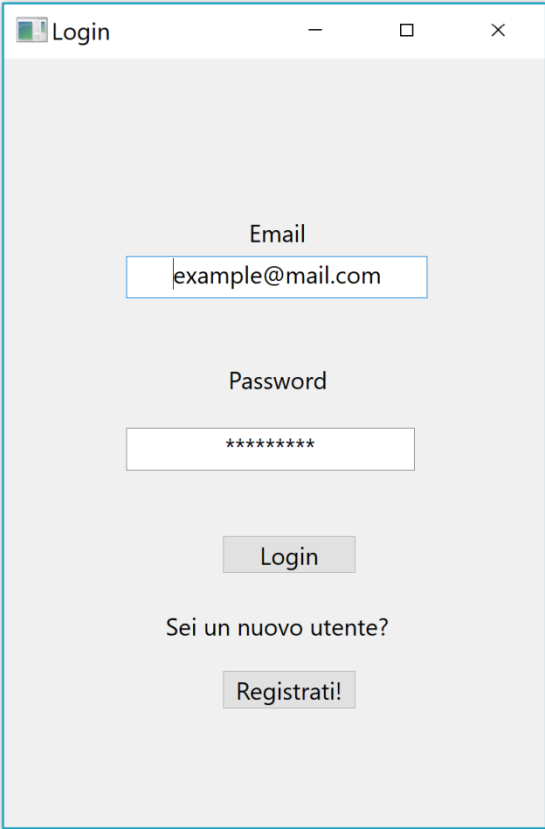


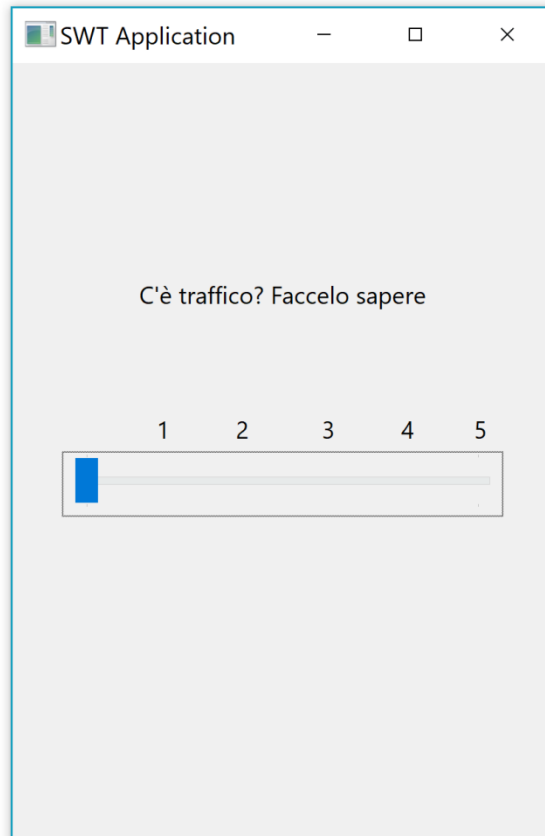
in crescendo per simulare una distribuzione di Poisson (che approssima al meglio lo sviluppo giornaliero del traffico reale) nell'inizializzazione delle variabili Automobile.

La Centralina utilizza una classe ClientCentralina per l'invio delle informazioni, la quale richiama il metodo `invia` presente in `InterfacciaInvio`, classe condivisa con il Sistema (`SharedSistemaCentralina`).

Un sistema simile viene utilizzato anche dall'`AppMobile` che sfrutta ugualmente RMI per lo scambio di informazioni. Essa differisce però dalla Centralina in quanto quando viene inviato un aggiornamento riguardante le coordinate GPS dell'utente il Sistema risponde con un'eventuale notifica di traffico.

Anche in questa parte di progetto è prevista un'interfaccia grafica. L'utente visualizza prima di tutto una schermata di Login connessa al database. Se l'utente non è ancora registrato può scegliere di visualizzare una seconda finestra per la registrazione. Entrambe queste funzioni sono implementate dal `GestoreApp` che, una volta effettuato il Login o portata a termine la registrazione inizializza un oggetto `Utente`. Crea poi in parallelo un timer che aggiorna il Sistema con le coordinate dell'utente e una schermata da usare per segnalare il traffico.





Ai fini della simulazione è stato creato un progetto `Simulatore` in cui viene realizzato a partire da delle coordinate, precedentemente inizializzate, un `ArrayList` che rappresenta il percorso di un'auto. Vengono quindi fatti concorrere due thread, l'uno per navigare l'`ArrayList` e associare all'utente coordinate sempre nuove, l'altro per simulare l'invio del GPS al Sistema. In questo modo si ipotizza lo spostamento dell'auto al fine di controllare il normale funzionamento del software.

