

SAMUELE MOSCATELLI, NICOLÒ PINCIROLI,  
ANDREA POZZOLI

## GESTIONE DEL TRAFFICO



DOCUMENTAZIONE DEL PROGETTO DI  
PROVA FINALE

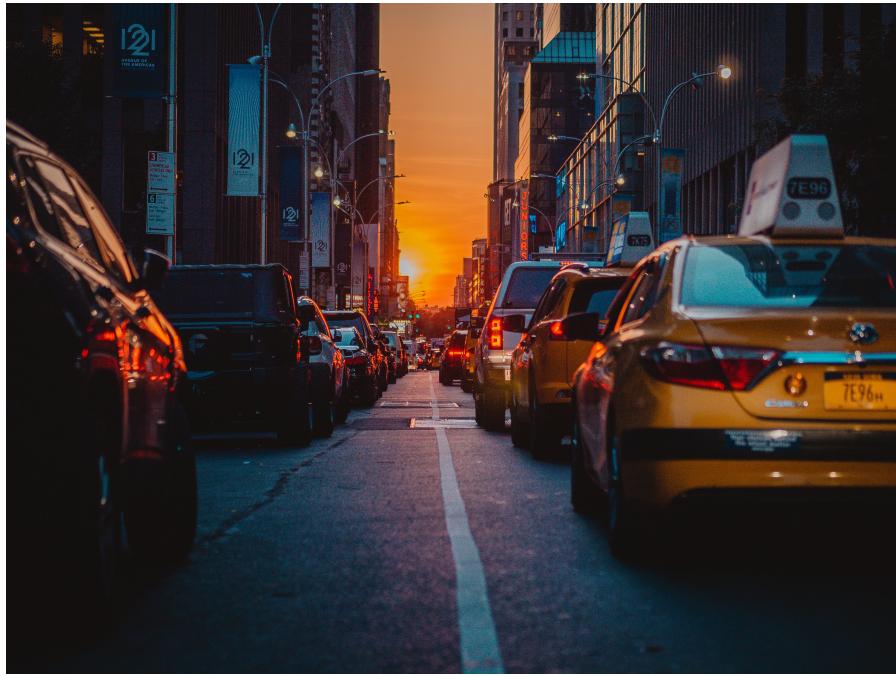
POLIMI

INGEGNERIA INFORMATICA

A.A. 2018 - 2019

Samuele Moscatelli, Nicolò Pincioli,  
Andrea Pozzoli,  
*Gestione del traffico*  
*Documentazione del progetto di prova finale*

E-MAIL:  
Samuele Moscatelli - sem.mosca97@libero.it  
Nicolò Pincioli - nicolopinci.1997@gmail.com  
Andrea Pozzoli - pozzoliandrea97@gmail.com



**Figura 1:** Un esempio di traffico

## INDICE

<b>1 ANALISI DEI REQUISITI . . . . .</b>	<b>1</b>
1.1 Tema del progetto	1
1.2 Analisi di fattibilità	2
1.2.1 Analisi dei costi	3
1.2.2 Dati trasmessi	4
1.2.3 Conclusioni	4
1.3 Diagramma i*	5
1.3.1 Sistema centrale	6
1.3.2 Centralina automobilistica	7
1.3.3 Centralina stradale	8
1.3.4 Applicazione mobile	9
1.3.5 Utente	10
1.3.6 Amministratore	11
1.4 Data dictionary	12
<b>2 DESIGN . . . . .</b>	<b>22</b>
2.1 Use case Modeling	23
2.1.1 Use case diagram	23
2.2 Structural Modeling - Static diagrams	25
2.2.1 Class diagram	25
2.2.2 Object diagram	30
2.3 Structural Modeling - Implementation diagrams	33

2.3.1	Deployment diagram	33
2.3.2	Component diagram	35
2.4	Dynamic Modeling - Interaction diagrams	37
2.4.1	Sequence diagrams	37
2.4.2	Collaboration diagram	40
2.5	Dynamic Modeling - Statechart diagrams	41
2.6	Dynamic Modeling - Activity diagrams	44
<b>3</b>	<b>IMPLEMENTAZIONE IN JAVA . . . . .</b>	<b>48</b>
3.1	Sistema Centrale	48
3.2	Centralina stradale	49
3.3	Centralina automobilistica	49
3.4	Applicazione	50
3.5	Algoritmi significativi	51
3.5.1	Calcolo delle applicazioni da notificare	51
3.5.2	Aggiornamento della tabella di traffico	52
3.5.3	Rilevazione della posizione	53
3.5.4	Connessione con RMI e messaggio di errore	54
3.5.5	Calcolo dell'intervallo di aggiornamento della centralina stradale	55
3.5.6	Variazione di velocità rilevata dalla centralina stradale	56
3.5.7	Calcolo della nuova posizione della centralina automobilistica	56
3.6	Interfaccia grafica	58
3.6.1	Login e registrazione	58
3.6.2	Sistema centrale	58
3.6.3	Centralina stradale	59
3.6.4	Centralina automobilistica	60
3.6.5	Applicazione mobile	60
3.6.6	Utilizzo di Swing	62
3.7	Formato delle notifiche	62
3.8	Protocollo di comunicazione	63
3.9	Casi di test JUnit	63
3.10	Simulazione	67
<b>4</b>	<b>MODIFICHE . . . . .</b>	<b>69</b>
4.1	Protocollo di comunicazione	69
4.2	Aggiunta di nuovi metodi, attributi e classi	69
4.3	Interfaccia grafica	69
4.4	Scelta della modalità di rappresentazione dei dati	70
4.5	Stima dei dati trasmessi	70
4.6	Proposte di soluzione ai potenziali problemi	70
4.7	Introduzione Centralina Automobilistica	70
<b>A</b>	<b>APPENDICE . . . . .</b>	<b>71</b>
A.1	Calcolo dei costi	71

A.2 Calcolo dei dati trasmessi	<a href="#">72</a>
B LISTA DI INDIRIZZI POSSIBILI . . . . .	<a href="#">73</a>
BIBLIOGRAFIA . . . . .	<a href="#">77</a>

# 1

## ANALISI DEI REQUISITI

### 1.1 TEMA 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).

## 1.2 ANALISI DI FATTIBILITÀ

Il progetto in analisi consiste nel realizzare un sistema di gestione e controllo del traffico cittadino, in particolare per il Comune di Como. In questa sezione si vuole analizzare la fattibilità del progetto individuando i clienti, l'obiettivo, i benefici, le risorse disponibili, i possibili ostacoli e rischi e come minimizzarli. Inoltre, in due sottosezioni, vengono studiati i costi di creazione e mantenimento del sistema e la quantità di dati trasmessi quando esso è in esecuzione.

I clienti del progetto proposto sono:

- il docente di ingegneria del software;
- il Comune di Como.

L'obiettivo è quello di monitorare il traffico cittadino e ottimizzarne la gestione. Inizialmente la portata del progetto è limitata ad un contesto comunale, ma il campo di azione può essere esteso poi ad aree geografiche più ampie senza che vengano introdotte modifiche significative.

Lo sviluppo del sistema avrebbe un impatto positivo sia a livello ambientale, in quanto ridurrebbe le emissioni dovute alle code, sia in termini di efficienza degli spostamenti dei cittadini, grazie alla possibilità di scegliere percorsi meno trafficati. Inoltre si migliorerebbe anche il trasporto pubblico, in quanto le strade diventerebbero più scorrevoli, e si avrebbe una tempestività maggiore nelle situazioni di emergenza. Nel Comune di Como si aiuterebbe infine a gestire più efficacemente la distribuzione dei flussi stradali, uno dei maggiori problemi della città.

Le risorse disponibili possono essere suddivise in tre categorie: persone, tempo e attrezzature. Per quanto riguarda le persone, è prevista la collaborazione di tre studenti, supportati da due esercitatori e due assistenti di laboratorio. Il tempo stimato per la realizzazione del progetto è di tre mesi, cadenzati da quattro scadenze. Le attrezzature necessarie sono di due tipologie:

- Software: Eclipse, OpenOme, StarUML, GitLab, L<sup>A</sup>T<sub>E</sub>X;
- Dispositivi: centraline stradali, centraline per veicoli, sistema centrale, personal computer.

Esistono alcuni potenziali rischi ed ostacoli che potrebbero rallentare lo sviluppo del sistema e che, di conseguenza, devono essere risolti, come mostrato nella tabella mostrata nella pagina seguente.

Ostacoli e rischi	Possibili soluzioni
Tempo ridotto	Organizzazione del lavoro, incontri frequenti per lo sviluppo del progetto
Requisiti generici	Colloqui con l'insegnante, i tutor e gli esercitatori, cercando di comprendere a fondo le richieste
Eventuali problemi di comunicazione	Incontri frequenti e richieste tempestive in caso di dubbi sulle richieste
Difficoltà nell'acquisizione di nuove competenze	Studio di ingegneria del software, utilizzo della documentazione OpenOme, UML e Java per eventuali approfondimenti
Insoddisfazione del cliente	Confronto continuo con il cliente, in modo da comprendere a fondo le sue esigenze
Scarsa estensibilità del software	Rispettare i principi dell'object orientation durante le fasi di progettazione e di implementazione
Scarsa comprensibilità del codice da parte di altri sviluppatori	Commento approfondito del codice, redazione di una documentazione chiara, completa e organizzata, utilizzo di strutture standard (ad es.: design pattern)
Eccessiva ambiziosità dei progettisti rispetto alle proprie competenze tecniche	Organizzazione e suddivisione del lavoro, rispetto dei principi dell'object orientation, acquisizione di nuove competenze tecniche, sovrastima dei tempi necessari per la realizzazione del sistema complessivo
Difficoltà d'uso del software realizzato	Utilizzo di interfacce grafiche, corredate da una documentazione completa, chiara e approfondita a disposizione degli utenti finali

#### 1.2.1 Analisi dei costi

Si analizzano ora i costi di creazione e gestione nel tempo del sistema. Il risultato dei calcoli dipende da tre fattori: il costo dei dispositivi che costituiscono il sistema, il consumo elettrico all'ora, e il costo orario di manutenzione (sezione A.1). Dai conti effettuati e riportati in appendice, si ricava che il costo totale in funzione del tempo t espresso in ore vale:

$$C(t) = 440000 + 1.52t$$

dove il primo numero è il costo di realizzazione mentre il secondo è il costo dell'elettricità e della manutenzione in funzione del tempo espresso in ore. Annualmente il sistema necessita di una spesa pari a circa 14000 €(ottenuto ponendo t=8760h).

### 1.2.2 Dati trasmessi

La stima dei dati trasmessi, approfondita all'interno dell'appendice [A.2](#), tiene conto di margini di sicurezza molto elevati, in modo da tenere in considerazione eventuali imprevisti e sviluppi del sistema.

In particolare, le centraline automobilistiche possono raggiungere una velocità di trasmissione pari a circa  $10^7$  bit/s. Le centraline stradali, invece, scambiano dati per un volume complessivo di  $10^5$  bit/s. Infine è possibile stimare la quantità di dati inviati e ricevuti dall'applicazione mobile, pari a circa  $10^7$  bit/s. A partire da questi dati è possibile ottenere che il traffico di dati massimo sarebbe nell'ordine di grandezza di 10 milioni di bit/s ( $10^7$  bit/s).

### 1.2.3 Conclusioni

Dallo studio complessivo di fattibilità risulta che le risorse e le tempestiche a disposizione sono sufficienti affinché gli obiettivi prefissati vengano raggiunti. L'analisi dei rischi e degli ostacoli che possono presentarsi non ha rivelato seri motivi che possano condurre ad un possibile insuccesso, dal momento che sono state individuate delle strategie per minimizzare i rischi.

L'attuazione delle possibili soluzioni permette infatti di gestire al meglio i tempi tenendo in considerazione eventuali imprevisti o ritardi, di migliorare la comunicazione e la comprensione all'interno della squadra e con i clienti e di acquisire e padroneggiare nuovi strumenti e competenze necessari per il miglior sviluppo del progetto.

Si considera comunque la possibilità che si verifichino problemi imprevisti, ma l'analisi effettuata dovrebbe permettere di gestirli senza avere ripercussioni significative sul progetto.

Infine, i benefici che deriverebbero dalla realizzazione del sistema giustificherebbero i costi, comunque contenuti per il Comune preso in considerazione.

## 1.3 DIAGRAMMA I\*

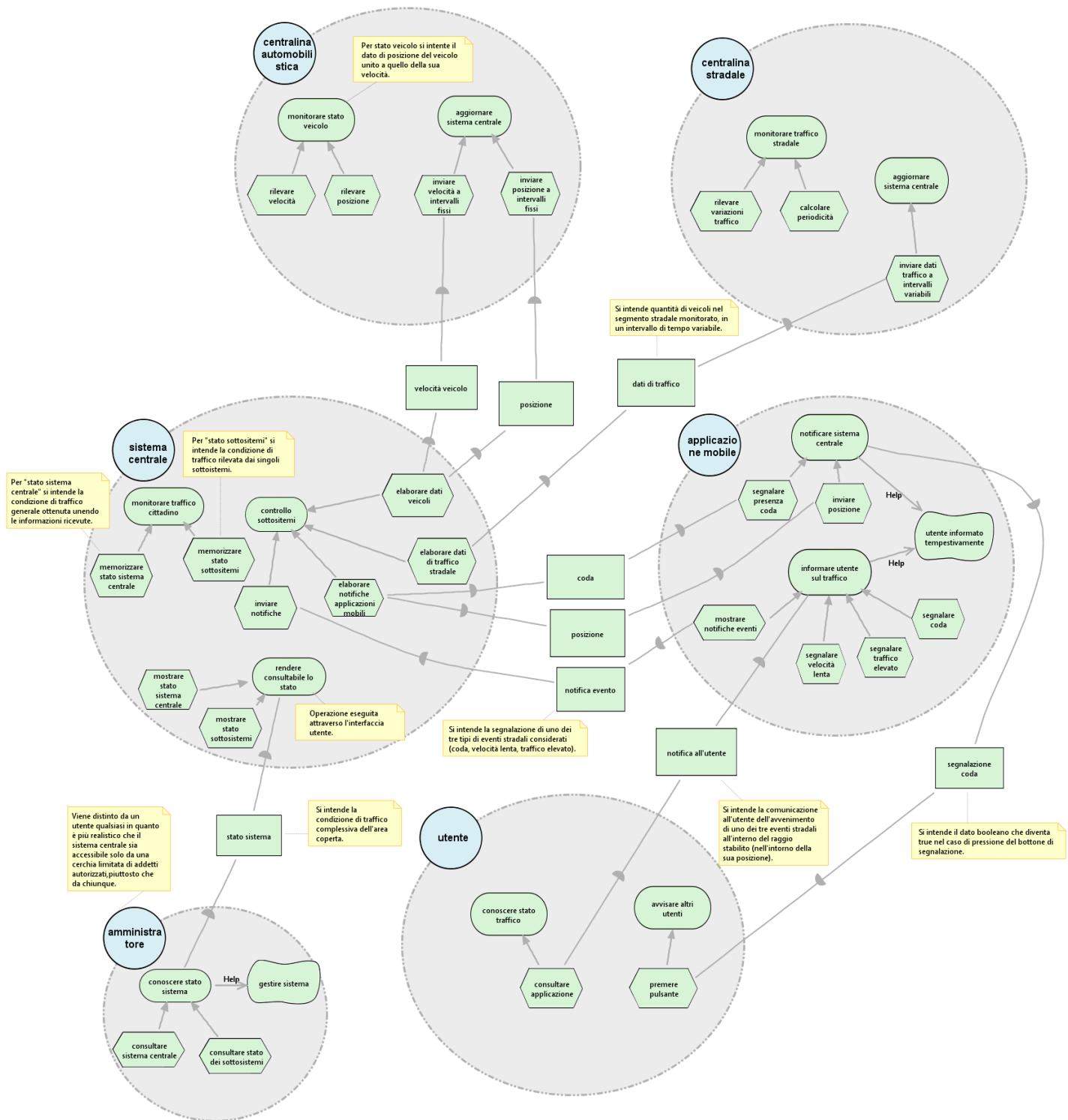


Figura 2: Diagramma i\* del sistema di gestione del traffico

### 1.3.1 Sistema centrale

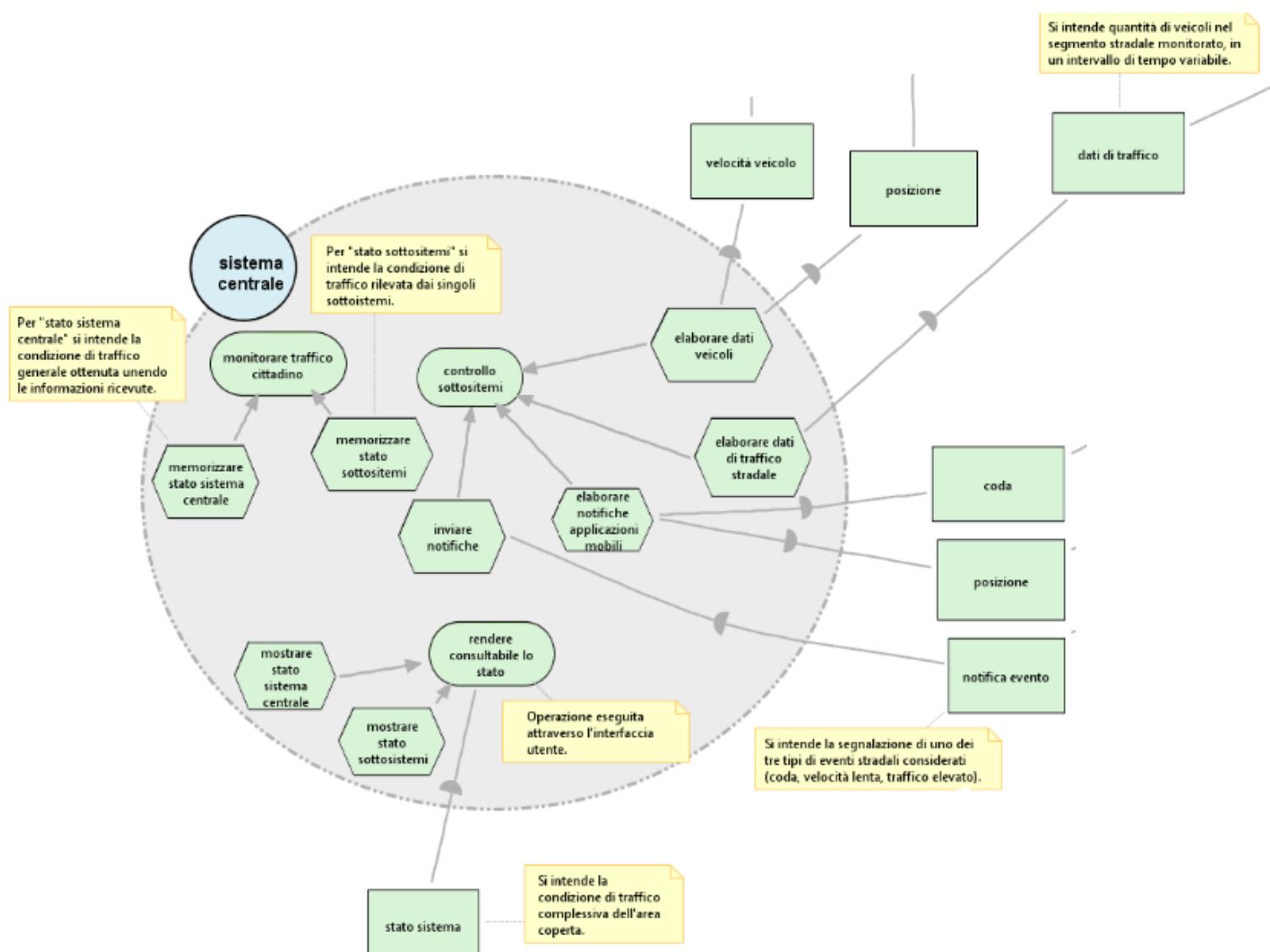


Figura 3: Il sistema centrale

Il sistema centrale è caratterizzato da tre hard goal. Il primo di essi riguarda il monitoraggio del traffico cittadino, ovvero la memorizzazione di tutte le informazioni di stato. I task relativi alla memorizzazione dello stato del sistema centrale e alla memorizzazione dello stato dei sottosistemi specificano in che modo viene raggiunto questo obiettivo. Con il termine stato si intende l'insieme dei dati raccolti riguardanti il traffico.

Il secondo hard goal del sistema centrale è rendere consultabile lo stato del sistema complessivo. Per fare ciò vengono utilizzate le informazioni memorizzate attraverso l'adempimento dell'obiettivo precedentemente discusso. Tali dati vengono poi organizzati in un'interfaccia

apposita in modo da mostrarli in modo intuitivo all'amministratore. Anche in questo caso, è possibile suddividere il goal in due task che specificano lo stato mostrato, ovvero lo stato del sistema centrale e lo stato dei singoli sottosistemi. Infine il sistema centrale ha il compito di controllare i sottosistemi. Quest'ultima operazione richiede quattro task:

- elaborare dati veicoli, che riceve dalle centraline automobilistiche i dati di posizione e velocità come risorse;
- elaborare dati di traffico stradale, che richiede alle centraline stradali i dati di traffico che rilevano;
- elaborare notifiche applicazioni mobili, che riceve la posizione e la segnalazione di coda inviata dall'applicazione mobile.
- inviare notifiche, che comunica con l'applicazione dopo aver elaborato una condizione di traffico.

#### 1.3.2 Centralina automobilistica

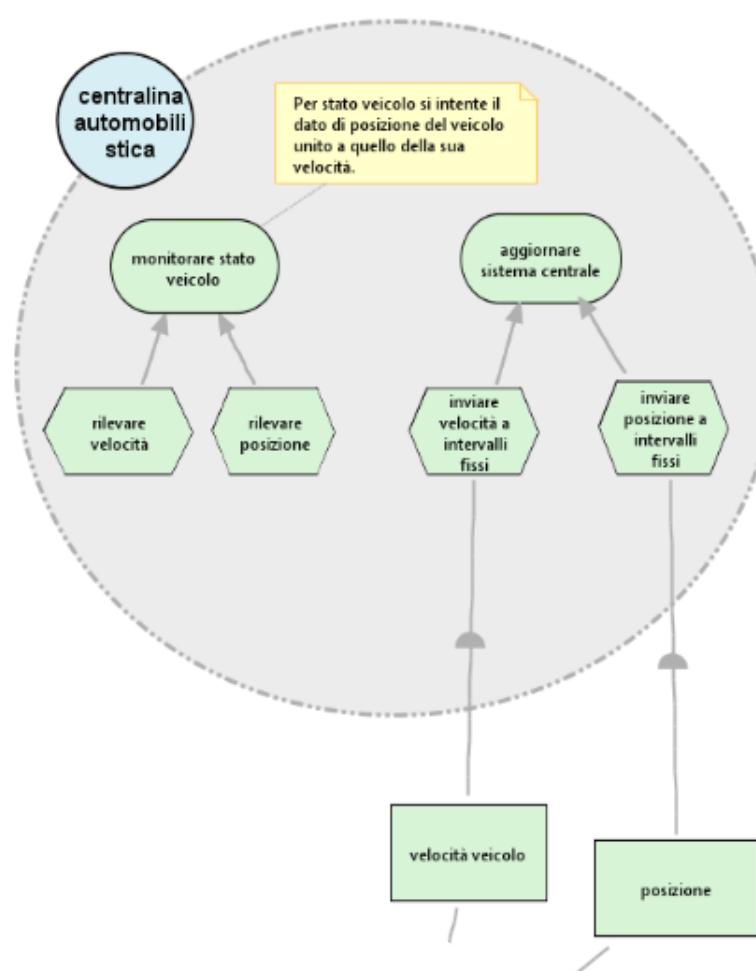


Figura 4: Centralina automobilistica

La centralina automobilistica ha due hard goal. Il primo obiettivo è quello di monitorare lo stato del veicolo. Questo hard goal è raggiunto grazie a due task, ovvero rilevare la velocità e rilevare la posizione dell'automobile. Il secondo obiettivo, aggiornare il sistema centrale, viene raggiunto ricevendo lo stato del veicolo (i dati raccolti) rilevato attraverso l'obiettivo precedente e inviando velocità e posizione a intervalli di tempo regolari al sistema centrale, ricorrendo quindi a due task.

#### 1.3.3 Centralina stradale

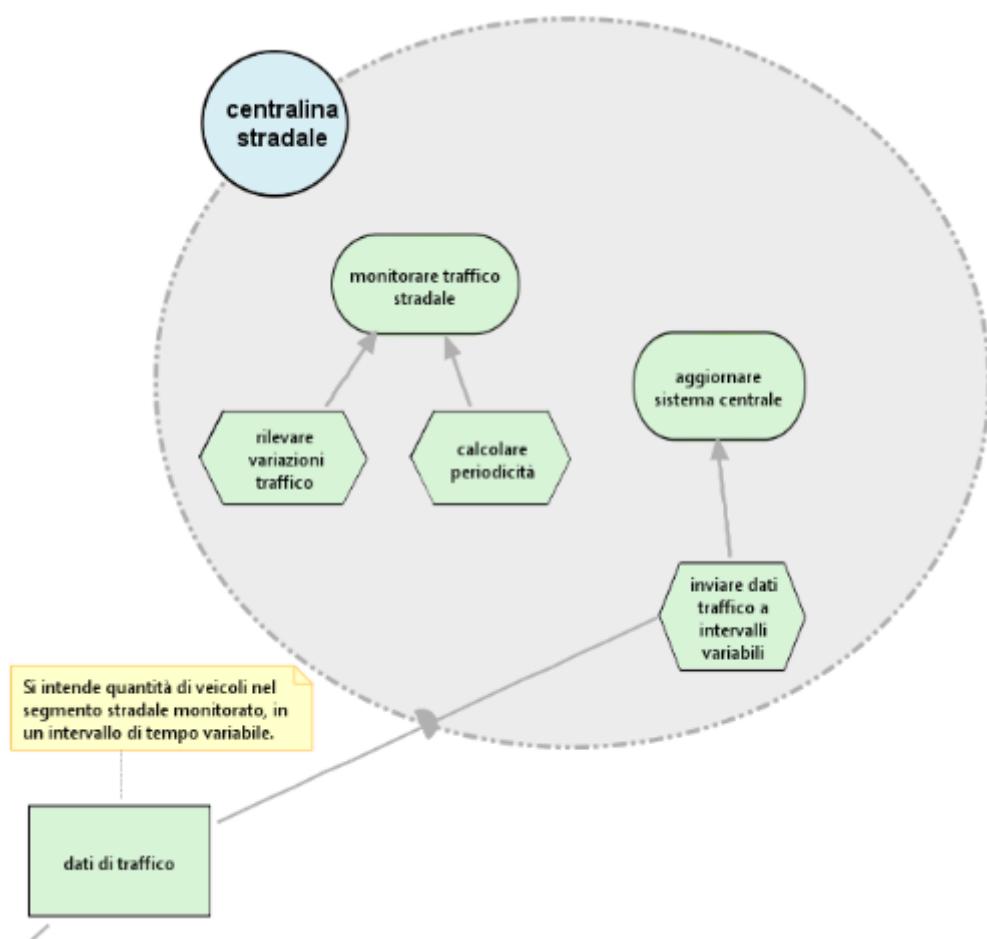


Figura 5: Centralina stradale

La centralina stradale ha due hard goal molto simili a quelli della centralina automobilistica. Uno degli obiettivi di questo attore è quello di monitorare il traffico stradale. Questo viene raggiunto tramite i due task per il rilevamento delle variazioni di traffico e il calcolo della periodicità. Per variazioni di traffico si intende l'aumento o la diminuzione del numero di veicoli che vengono rilevati dalla centralina in

un intervallo di tempo. Con periodicità invece si intende l'intervallo di tempo che intercorre tra l'invio di un blocco di dati da parte della centralina e l'invio del blocco successivo da parte della stessa. Tale periodo di tempo varia a seconda del numero di auto presenti nel segmento di strada. In particolare, nel caso in cui aumenti il traffico la periodicità si riduce, mentre nel caso in cui il traffico diminuisca la periodicità aumenta. Il secondo obiettivo è l'aggiornamento del sistema centrale, portato a termine inviando i dati di traffico rilevati al sistema centrale sotto forma di risorsa.

#### 1.3.4 Applicazione mobile

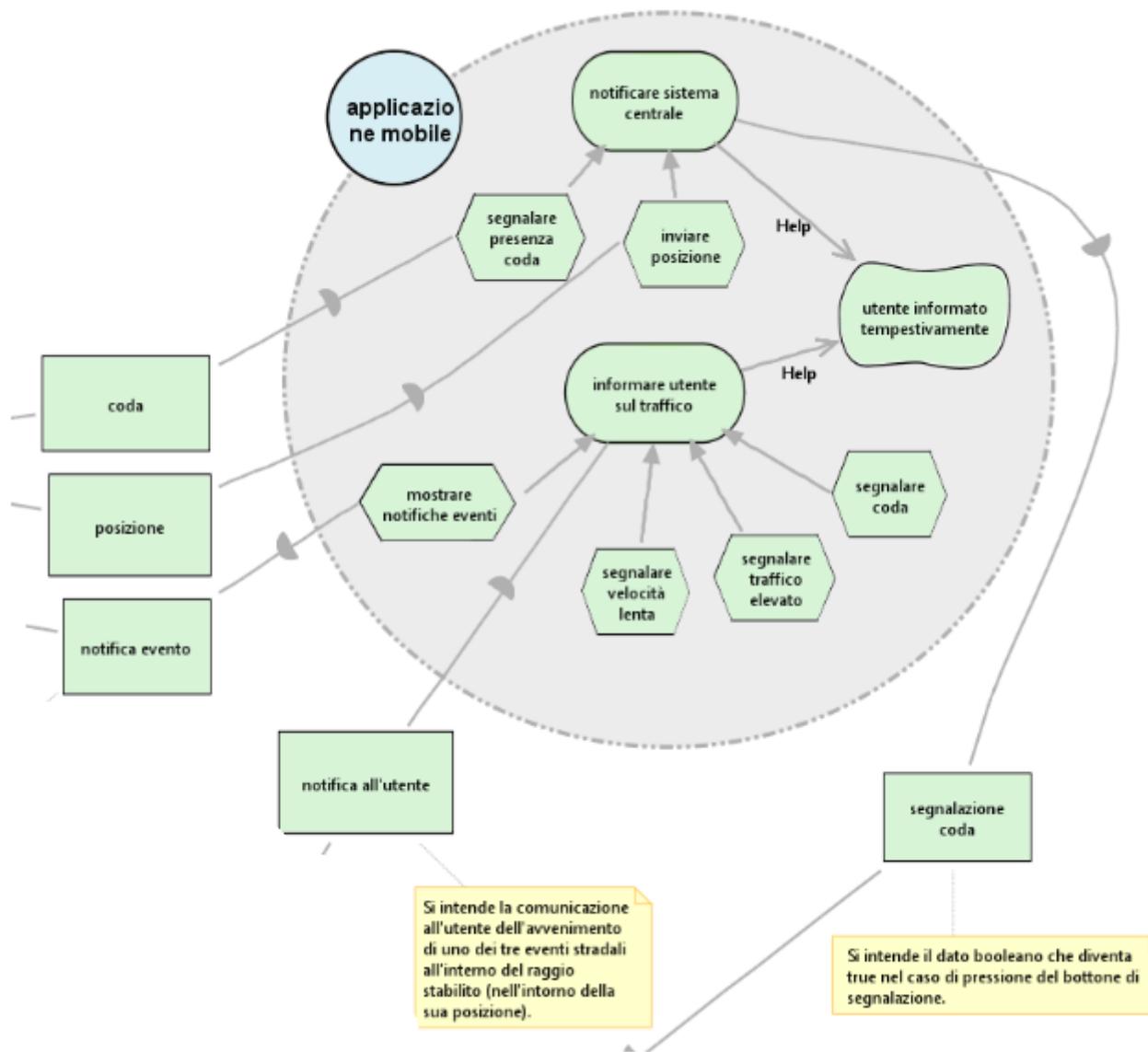


Figura 6: L'applicazione mobile

L'applicazione mobile comunica con il sistema centrale e con l'utente. Presenta quindi due obiettivi, che sono notificare il sistema centrale e informare l'utente sul traffico. Il primo obiettivo, che riceve dall'utente una segnalazione di coda come risorsa, si raggiunge tramite due task (segnalare la presenza di coda come risorsa, e inviare la posizione da cui si manda la segnalazione), che inviano le risorse già analizzate nella descrizione del sistema centrale. Il secondo obiettivo è più articolato e si suddivide in quattro task:

- mostrare notifiche eventi, che riceve dal sistema centrale la notifica di un particolare evento di traffico e lo invia all'utente;
- segnalare velocità lenta, ovvero comunicare all'utente questo specifico evento nella notifica;
- segnalare traffico elevato, ovvero comunicare all'utente questo specifico evento nella notifica;
- segnalare coda, ovvero comunicare all'utente questo specifico evento nella notifica;

Inoltre, può essere preso in considerazione il soft goal riguardante la tempestività nell'aggiornare l'utente, favorita dai due hard goal precedentemente descritti.

#### 1.3.5 Utente

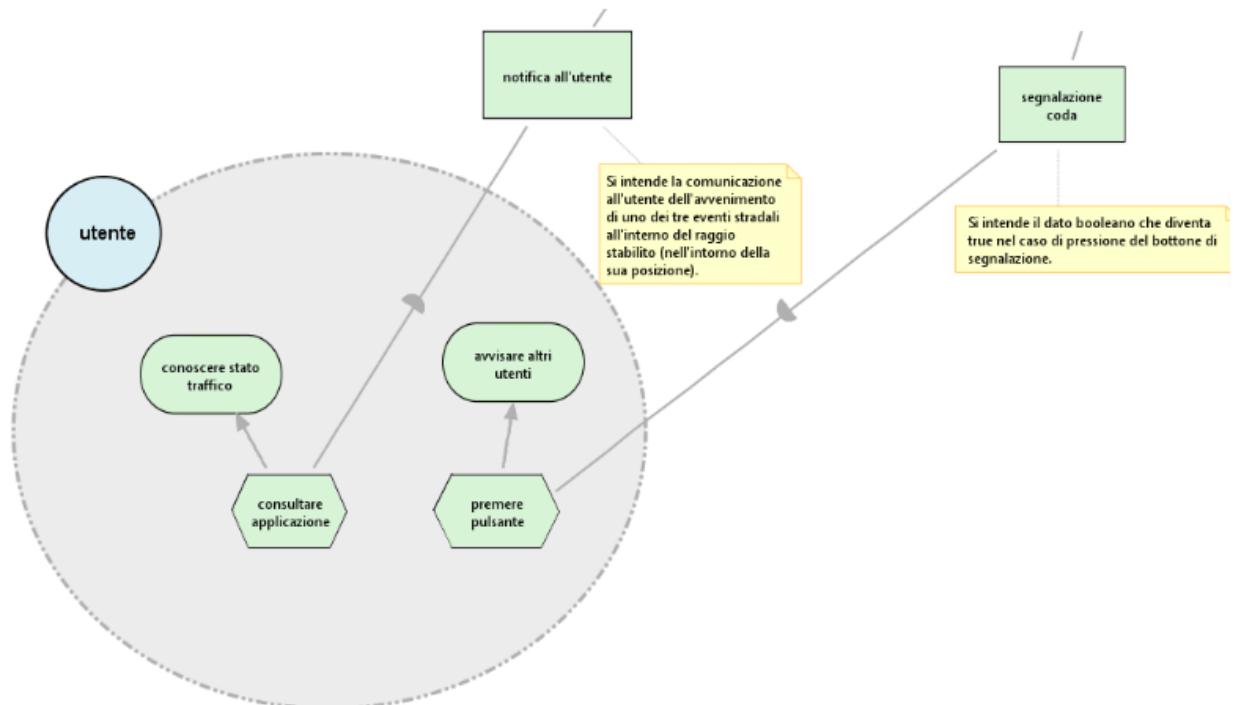


Figura 7: L'utente

L'utente ha due hard goal: conoscere lo stato del traffico e avvisare gli altri utenti della situazione del traffico. Il primo viene eseguito grazie a un task (consultare l'applicazione), che riceve come risorsa la notifica che l'applicazione mobile invia all'utente per comunicargli un particolare evento di traffico. L'altro obiettivo viene raggiunto grazie ad un task (pressione del pulsante), che invia all'applicazione un avviso indicante la presenza di coda nel tratto stradale in cui si trova l'utente.

#### 1.3.6 Amministratore



Figura 8: L'amministratore

L'amministratore ha un singolo hard goal, ovvero conoscere la condizione di traffico dell'area geografica coperta dal sistema. Tale obiettivo viene raggiunto attraverso due task, ovvero la consultazione dello stato del sistema centrale e la consultazione dello stato dei singoli sottosistemi. Per fare ciò il sistema centrale fornisce l'amministratore di un'interfaccia utente apposita, che nel diagramma i\* è rappresentata dalla risorsa "stato sistema". L'amministratore avrà quindi l'obiettivo di gestire il sistema, rappresentato nel diagramma come soft goal in quanto la verifica del suo raggiungimento non è quantificabile in modo esatto.

#### 1.4 DATA DICTIONARY

<b>Nome</b>	Amministratore
<b>Definizione</b>	Utente incaricato della gestione e della consultazione del sistema centrale.
<b>Tipo di dato</b>	È formato da una stringa che indica il suo nome utente e da una stringa che indica la sua password.
<b>Dimensione del dato</b>	512 bit
<b>Sinonimi</b>	Gestore, addetto
<b>Esempi</b>	Luca   passluca
<b>Sottotipi</b>	
<b>Supertipi</b>	
<b>Attributi</b>	Nomeutente, password
<b>Componenti</b>	
<b>Relazioni</b>	Sistema centrale

<b>Nome</b>	Applicazione mobile
<b>Definizione</b>	Componente software che serve per instaurare la comunicazione tra l'utente e il sistema centrale.
<b>Tipo di dato</b>	È formato da un intero positivo necessario per la sua identificazione e da un dato di tipo posizione.
<b>Dimensione del dato</b>	variabile
<b>Sinonimi</b>	
<b>Esempi</b>	34576   -36.82065, 175.07823
<b>Sottotipi</b>	
<b>Supertipi</b>	
<b>Attributi</b>	Identificativo
<b>Componenti</b>	Posizione
<b>Relazioni</b>	Utente, sistema centrale, posizione, notifica

<b>Nome</b>	Bottone
<b>Definizione</b>	Tasto a disposizione dell'utente di una applicazione mobile per segnalare la presenza di coda in una determinata posizione
<b>Tipo di dato</b>	È formato da un booleano che indica lo stato del bottone (premuto o no).
<b>Dimensione del dato</b>	1 bit
<b>Sinonimi</b>	Tasto, pulsante
<b>Esempi</b>	True (premuto), false (non premuto)
<b>Sottotipi</b>	
<b>Supertipi</b>	
<b>Attributi</b>	Stato(bool)
<b>Componenti</b>	
<b>Relazioni</b>	Notifica, dati di traffico, posizione, coda

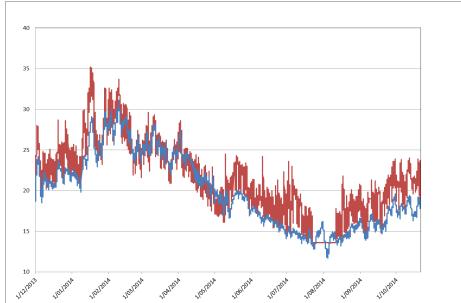
<b>Nome</b>	Centralina automobilistica
<b>Definizione</b>	Centralina installata all'interno di un veicolo in grado di trasmettere al sistema centrale i dati di posizione e velocità del veicolo stesso.
<b>Tipo di dato</b>	È formato da un dato di tipo posizione, un dato di tipo velocità e uno di tipo intervallo di tempo.
<b>Dimensione del dato</b>	variabile
<b>Sinonimi</b>	Centralina del veicolo
<b>Esempi</b>	-36.82065, 175.07823   30   25 (Il sistema, attraverso dei sensori, riceve la velocità e la posizione del veicolo, trasmettendola a intervalli di tempo fissati al sistema centrale)
<b>Sottotipi</b>	
<b>Supertipi</b>	Centralina
<b>Attributi</b>	
<b>Componenti</b>	Posizione, intervallo di tempo, Velocità
<b>Relazioni</b>	Dati di traffico, Velocità, Posizione, Intervallo di tempo, Automobile

<b>Nome</b>	Centralina stradale
<b>Definizione</b>	Dispositivo adibito al controllo costante del flusso automobilistico in un determinato segmento stradale.
<b>Tipo di dato</b>	È formato da una stringa che indica lo stato, da un dato di tipo posizione, un dato di tipo velocità, uno di tipo conteggio e uno di tipo intervallo di tempo.
<b>Dimensione del dato</b>	variabile
<b>Sinonimi</b>	Stazione stradale, cellula stradale
<b>Esempi</b>	"accesa"   32.82065, 95.07823   30 (m/s)   230 (intervallo di tempo)   50 (conteggio), "spenta"   -24.82065, 95.07823   null   null   null
<b>Sottotipi</b>	
<b>Supertipi</b>	
<b>Attributi</b>	Stato(string)
<b>Componenti</b>	Intervallo di tempo, velocità, conteggio, posizione
<b>Relazioni</b>	Intervallo di tempo, notifica, dati di traffico, velocità, conteggio, posizione

<b>Nome</b>	Coda
<b>Definizione</b>	Indicazione della presenza o dell'assenza di una coda in una determinata posizione.
<b>Tipo di dato</b>	È formato da un booleano che indica la presenza di coda e da un dato di tipo posizione.
<b>Dimensione del dato</b>	71 bit
<b>Sinonimi</b>	Incolonramento
<b>Esempi</b>	1, 36.82065, 175.07823 significa che in posizione 36.82065, 175.07823 è presente una coda.
<b>Sottotipi</b>	
<b>Supertipi</b>	
<b>Attributi</b>	esiste_coda
<b>Componenti</b>	Posizione
<b>Relazioni</b>	Dati di traffico, Notifica, Automobile

<b>Nome</b>	Conteggio veicoli
<b>Definizione</b>	Enumerazione dei veicoli rilevati da una centralina stradale in un determinato intervallo di tempo.
<b>Tipo di dato</b>	È formato da un intero positivo che rappresenta il numero di veicoli, da due flag che indicano il comando di reset del conteggio e la presenza di overflow.
<b>Dimensione del dato</b>	32 bit (può contare un numero di automobili dell'ordine di grandezza della popolazione mondiale)
<b>Sinonimi</b>	Conteggio passaggi
<b>Esempi</b>	75   0   0 (esempio di dato) Per ogni veicolo rilevato il contatore aumenta di un'unità, a meno che reset non valga 1. Se supera il limite massimo del conteggio segnala un errore di overflow ponendo l'attributo overflow a 1.
<b>Sottotipi</b>	
<b>Supertipi</b>	
<b>Attributi</b>	numero_veicoli (30 bit), reset (1 bit), overflow (1 bit)
<b>Componenti</b>	
<b>Relazioni</b>	Dati di traffico, Centralina stradale, Intervallo di tempo

<b>Nome</b>	Dati di traffico
<b>Definizione</b>	Variabile che identifica la mobilità di un tratto stradale in funzione della velocità dei veicoli e del loro numero in un intervallo di tempo
<b>Tipo di dato</b>	È formato da un intero positivo che rappresenta la velocità media, dalla posizione geografica espressa con due numeri di dieci cifre, e da una stringa di massimo 16 caratteri per il tipo di traffico
<b>Dimensione del dato</b>	16bit+70bit+ 256bit=342bit
<b>Sinonimi</b>	Stato del traffico, flusso
<b>Esempi</b>	30   -36.82065, 175.07823   "velocità lenta"
<b>Sottotipi</b>	Coda, traffico elevato, velocità lenta
<b>Supertipi</b>	
<b>Attributi</b>	Tipo di traffico (string)
<b>Componenti</b>	Velocità, posizione
<b>Relazioni</b>	Velocità, posizione, applicazione mobile, sistema centrale, centralina stradale, centralina automobile

<b>Nome</b>	Diagramma
<b>Definizione</b>	Rappresentazione grafica di dati riguardanti le condizioni del traffico.
<b>Tipo di dato</b>	È formato da una lista puntata di coordinate e da informazioni sul colore espresse in formato RGBA.
<b>Dimensione del dato</b>	variabile
<b>Sinonimi</b>	Grafico
<b>Esempi</b>	
<b>Sottotipi</b>	Diagramma2D, Diagramma3D, Diagramma4D
<b>Supertipi</b>	
<b>Attributi</b>	Lista_punti (ogni punto da 20 a 40 bit), colore (10 bit)
<b>Componenti</b>	
<b>Relazioni</b>	Punto, Dati di traffico, Mappa

<b>Nome</b>	Intervallo di tempo
<b>Definizione</b>	Periodicità con cui viene notificato lo stato del traffico da parte dei sottosistemi al sistema centrale. Può essere un intervallo fisso o variabile a seconda del sottosistema da cui viene inviato.
<b>Tipo di dato</b>	Intero positivo che indica la durata della periodicità.
<b>Dimensione del dato</b>	15 bit (espresso in secondi)
<b>Sinonimi</b>	Periodo, frequenza
<b>Esempi</b>	100, 300, 640, 120 (secondi)
<b>Sottotipi</b>	
<b>Supertipi</b>	
<b>Attributi</b>	
<b>Componenti</b>	
<b>Relazioni</b>	Notifica, centralina stradale, traffico elevato, dati di traffico, conteggio, centralina auto

<b>Nome</b>	Mappa
<b>Definizione</b>	Rappresentazione grafica delle strade caricabili dalla posizione individuata dall'applicazione mobile con i relativi dati di traffico
<b>Tipo di dato</b>	Dato che rappresenta un'interpolazione di coordinate colorate in RGBA e che segnala i tratti di strada che presentano traffico
<b>Dimensione del dato</b>	Variabile
<b>Sinonimi</b>	Cartina stradale
<b>Esempi</b>	
<b>Sottotipi</b>	
<b>Supertipi</b>	
<b>Attributi</b>	Traffico, colore, segnali, lista punti
<b>Componenti</b>	Posizione, dati di traffico
<b>Relazioni</b>	Coda, velocità lenta, traffico elevato, posizione, raggio, veicolo, applicazione mobile

<b>Nome</b>	Notifica
<b>Definizione</b>	Segnalazione dell'avvenimento di un dato evento stradale riguardante il traffico. Può essere indirizzata da un sottosistema verso il sistema centrale o dal sistema centrale verso le applicazioni mobili, oppure anche da queste ultime verso l'utente.
<b>Tipo di dato</b>	È formato da una stringa che indica il tipo di notifica, e un dato di tipo posizione.
<b>Dimensione del dato</b>	Variabile
<b>Sinonimi</b>	Segnalazione, avviso, allarme
<b>Esempi</b>	"coda"   36.82065, 175.07823, "velocità lenta"   36.82065, 175.07823
<b>Sottotipi</b>	NotificaDaSC,notificaASC
<b>Supertipi</b>	
<b>Attributi</b>	Tipo(string)
<b>Componenti</b>	Posizione
<b>Relazioni</b>	Dati di traffico, velocità lenta, raggio, conteggio, posizione, coda, intervallo di tempo, bottone, traffico elevato, applicazione mobile

<b>Nome</b>	Posizione
<b>Definizione</b>	Posizione espressa in coordinate GPS (latitudine e longitudine)
<b>Tipo di dato</b>	È formato da due numeri espressi in virgola mobile.
<b>Dimensione del dato</b>	70 bit
<b>Sinonimi</b>	Posizione GPS, Coordinate
<b>Esempi</b>	-36.82065, 175.07823 (esempio di dato)
<b>Sottotipi</b>	
<b>Supertipi</b>	Posizione
<b>Attributi</b>	Latitudine (35 bit, virgola mobile), longitudine (35 bit, virgola mobile)
<b>Componenti</b>	
<b>Relazioni</b>	Mappa, Coda, Centralina stradale, Centralina automobilistica, Traffico elevato, Automobile, applicazione mobile

<b>Nome</b>	Raggio
<b>Definizione</b>	Distanza in metri tra la posizione rilevata dall'applicazione mobile e la porzione di mappa caricabile sullo smartphone (viene caricata un'area circolare)
<b>Tipo di dato</b>	Intero positivo
<b>Dimensione del dato</b>	16 bit
<b>Sinonimi</b>	
<b>Esempi</b>	500 m
<b>Sottotipi</b>	
<b>Supertipi</b>	
<b>Attributi</b>	
<b>Componenti</b>	
<b>Relazioni</b>	Mappa, applicazione mobile, posizione

<b>Nome</b>	Sistema centrale
<b>Definizione</b>	Sistema informatico composto da un server che mantenga in memoria i dati relativi al traffico e una potente unità di elaborazione che gestisca i dati ricevuti dalle unità esterne.
<b>Tipo di dato</b>	È formato dall'insieme dei dati di traffico e delle notifiche scambiati con gli altri sistemi.
<b>Dimensione del dato</b>	variabile
<b>Sinonimi</b>	unità centrale
<b>Esempi</b>	
<b>Sottotipi</b>	
<b>Supertipi</b>	
<b>Attributi</b>	
<b>Componenti</b>	dati di traffico, notifica
<b>Relazioni</b>	Centralina auto, centralina stradale, utente, applicazione, dati di traffico, velocità, mappa, posizione, diagramma, coda, velocità lenta, traffico elevato

<b>Nome</b>	Stato veicolo
<b>Definizione</b>	Insieme di dati inviati dalla centralina automobilistica riguardanti la posizione, la velocità e l'identificativo del veicolo.
<b>Tipo di dato</b>	È formato da un intero positivo che definisce l'identificativo e da un dato posizione e uno velocità.
<b>Dimensione del dato</b>	100 bit
<b>Sinonimi</b>	Auto, automobile, macchina, unità
<b>Esempi</b>	1154 (id)   -24.82065, 95.07823   50 (velocità)
<b>Sottotipi</b>	
<b>Supertipi</b>	
<b>Attributi</b>	NumeroID
<b>Componenti</b>	Posizione, velocità
<b>Relazioni</b>	Centralina auto, centralina stradale, utente, app

<b>Nome</b>	Traffico elevato
<b>Definizione</b>	Dato che definisce la presenza di molti veicoli in un determinato tratto stradale; viene scambiato tra sistema centrale e applicazioni mobili
<b>Tipo di dato</b>	È formato da un booleano che indica la presenza di traffico elevato e da un dato di tipo posizione.
<b>Dimensione del dato</b>	71 bit
<b>Sinonimi</b>	
<b>Esempi</b>	True   -36.82065, 175.07823 (presenza traffico elevato)
<b>Sottotipi</b>	
<b>Supertipi</b>	
<b>Attributi</b>	Traffico(bool)
<b>Componenti</b>	Posizione
<b>Relazioni</b>	Notifica, dati di traffico, mappa, conteggio, posizione

<b>Nome</b>	Utente
<b>Definizione</b>	Utilizzatore dell'applicazione mobile.
<b>Tipo di dato</b>	È formato da una stringa che indica il suo nome utente e da una stringa che indica la sua password.
<b>Dimensione del dato</b>	512 bit
<b>Sinonimi</b>	Guidatore, utilizzatore
<b>Esempi</b>	marco   passmarco
<b>Sottotipi</b>	
<b>Supertipi</b>	
<b>Attributi</b>	nomeutente, password
<b>Componenti</b>	
<b>Relazioni</b>	Applicazione mobile

<b>Nome</b>	Velocità
<b>Definizione</b>	Spazio percorso da un veicolo in un intervallo di tempo, viene raccolta da appositi sensori, misurata in metri al secondo
<b>Tipo di dato</b>	Intero positivo.
<b>Dimensione del dato</b>	16 bit
<b>Sinonimi</b>	
<b>Esempi</b>	10 m/s , 50m/s
<b>Sottotipi</b>	
<b>Supertipi</b>	
<b>Attributi</b>	
<b>Componenti</b>	
<b>Relazioni</b>	Dati di traffico, veicolo

<b>Nome</b>	Velocità lenta
<b>Definizione</b>	Tratto di strada in cui i veicoli si muovono a una velocità inferiore alla media
<b>Tipo di dato</b>	È formato da un booleano che indica la presenza di velocità lenta e da un dato di tipo posizione.
<b>Dimensione del dato</b>	71 bit
<b>Sinonimi</b>	Rallentamento
<b>Esempi</b>	1   -36.82065, 175.07823, significa che nella posizione indicata dai due numeri con la virgola c'è velocità lenta
<b>Sottotipi</b>	
<b>Supertipi</b>	
<b>Attributi</b>	Vlenta (bool)
<b>Componenti</b>	Posizione
<b>Relazioni</b>	Velocità, posizione, dati di traffico, notifica

# 2 | DESIGN

I diagrammi sono stati studiati, sviluppati e presentati seguendo il seguente schema:

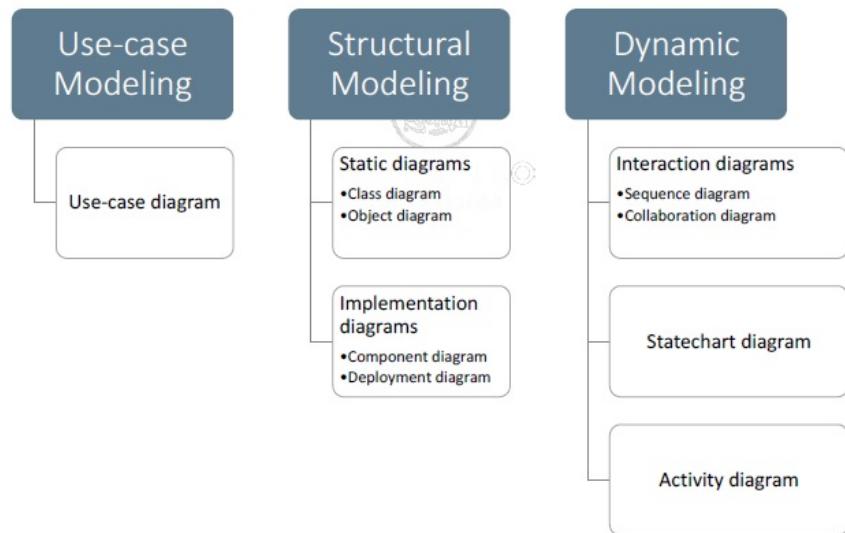


Figura 9: Schema UML [sof]

## 2.1 USE CASE MODELING

### 2.1.1 Use case diagram

Lo use case diagram è una tecnica utilizzata per identificare i requisiti funzionali di un sistema e si basa sulla descrizione delle interazioni tipiche tra gli utenti e il sistema.

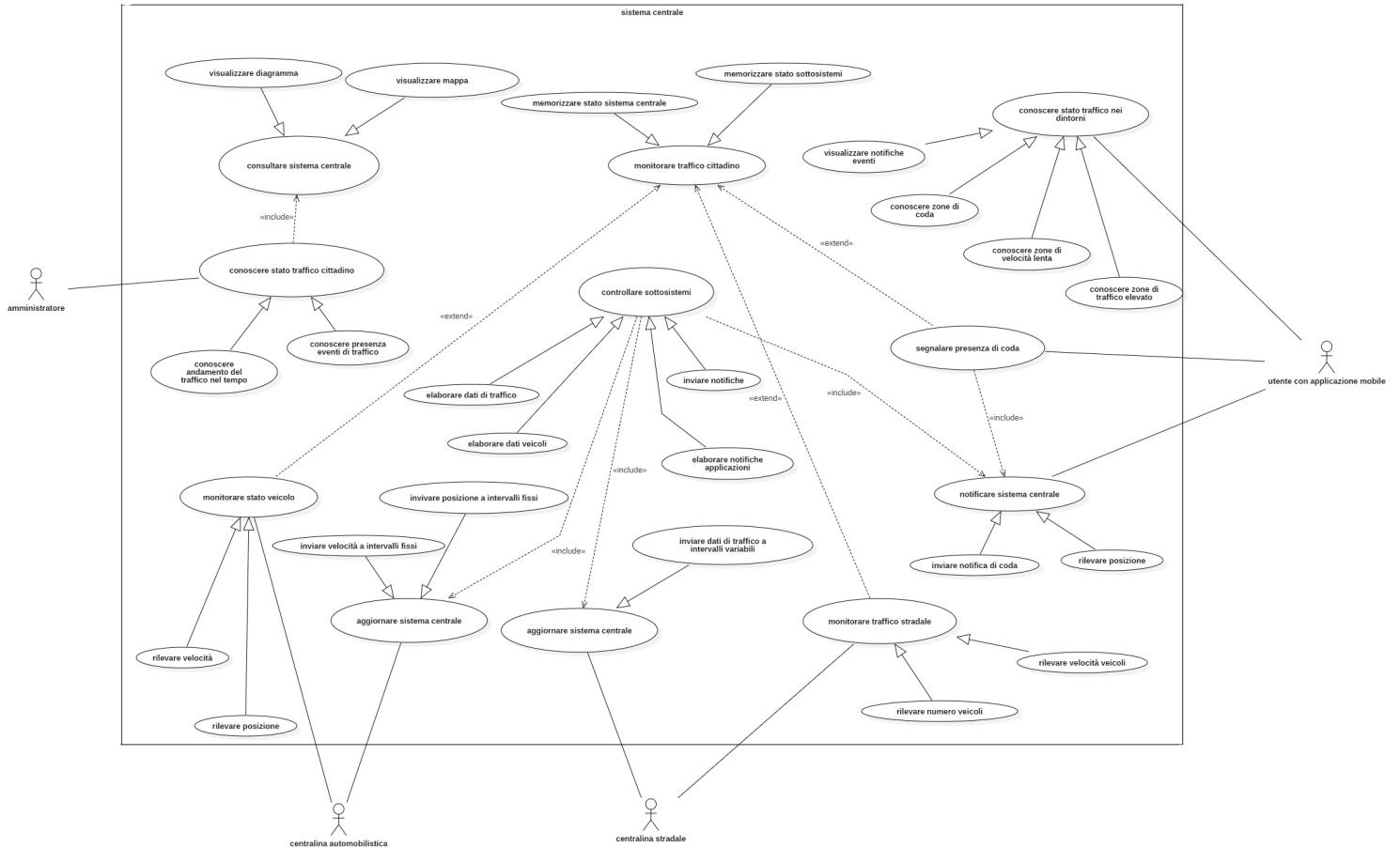


Figura 10: Use case diagram

In questo use case diagram, in particolare, come attori sono stati scelti l'amministratore, la centralina stradale, la centralina automobilistica e l'utente che utilizza l'applicazione. L'amministratore ha il compito di conoscere lo stato del traffico consultando il sistema centrale, il quale gli presenta i dati graficamente con un diagramma e una mappa. Le centraline sono state inserite come attori in quanto assumono un ruolo attivo nei confronti del sistema centrale. Esse infatti monitorano i veicoli su cui sono installate (nel caso delle centraline automobilistiche) e il traffico in una determinata posizione (per le centraline stradali), e inviano i dati raccolti al sistema centrale. L'utente e l'applicazione sono stati uniti in un unico attore in quanto è l'utente

che utilizza l'applicazione a interagire con il sistema. Le interazioni consistono nel segnalare al sistema centrale la presenza di coda in una determinata posizione, e apprendere dal sistema quali sono le zone che presentano eventi di traffico.

L'insieme delle interazioni tra gli attori e il sistema centrale portano al monitoraggio completo del traffico cittadino e al controllo dei singoli sottosistemi da parte del sistema centrale.

## 2.2 STRUCTURAL MODELING – STATIC DIAGRAMS

### 2.2.1 Class diagram

Il class diagram descrive i tipi degli oggetti che fanno parte di un sistema e le varie tipologie di relazioni statiche tra di essi. Inoltre, il class diagram mostra le proprietà e le operazioni di ogni classe e i vincoli che si applicano ai collegamenti tra gli oggetti.

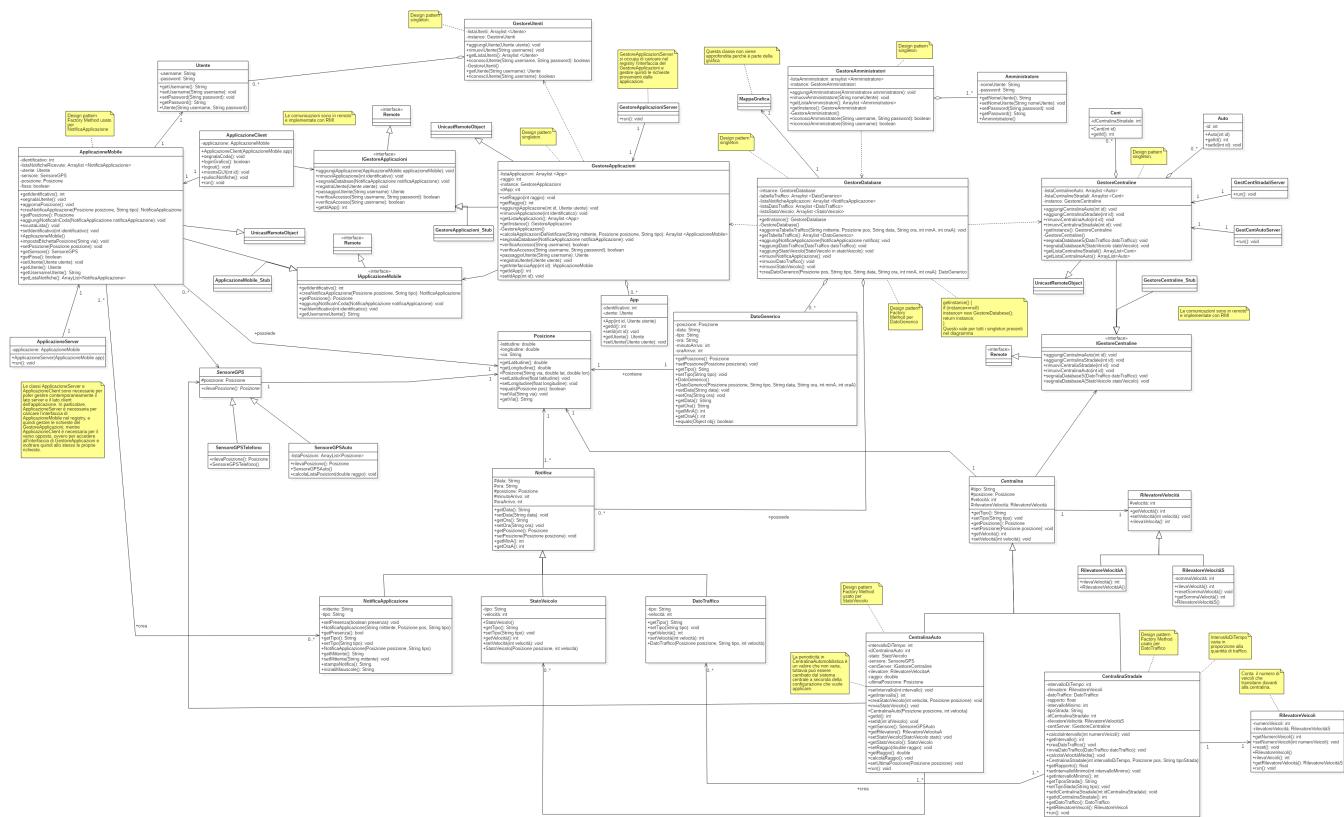


Figura 11: Class diagram

In questo class diagram si è deciso di rappresentare il sistema complessivo. In particolare si è suddiviso ciò che nei requisiti era stato individuato come sistema centrale in cinque classi, ognuna adibita a svolgere uno specifico compito di gestione. Due di esse, GestoreApplicazioni e GestoreCentraline, si occupano, rispettivamente, di gestire le applicazioni mobili e le centraline, sfruttando l'utilità dell'interfaccia di comunicazione remota RMI fornita dalla piattaforma Java per scambiare dati con queste ultime. Le restanti tre classi si occupano invece della gestione delle informazioni relative allo stato del traffico (GestoreDatabase), agli utenti che utilizzano le applicazioni mobili (GestoreUtenti) e agli amministratori di sistema (GestoreAmministratori). Sono state poi sviluppate le classi centralina, ap-

plicazione, utente e amministratore, con i relativi componenti e i dati che creano. Di seguito vengono illustrate più nel dettaglio le classi individuate.

#### *GestoreDatabase*

La classe GestoreDatabase è costruita usando il design pattern singleton. Il suo compito è sostanzialmente quello di ricevere e memorizzare tutti i dati provenienti dai sottosistemi esterni (centraline e applicazioni mobili), elaborarli e unirli sotto forma di dati di un solo tipo. Per questo motivo il GestoreDatabase ricorre anche al design pattern Factory Method, utilizzato per la creazione del DatoGenerico. Inoltre, tale classe si occupa anche della mappa (rappresentata dalla classe MappaGrafica) che mostra all'amministratore lo stato del sistema complessivo. In particolare il suo compito è quello di aggiungere nuovi marcatori in corrispondenza delle informazioni ricevute e rimuovere quelli ormai obsoleti (ovvero presenti sulla mappa da oltre 3 minuti).

#### *GestoreApplicazioni*

La classe GestoreApplicazioni è costruita usando il design pattern singleton. Essa è adibita alla gestione delle applicazioni mobili: in primo luogo deve salvare tutte le applicazioni che sono state installate e che sono effettivamente funzionanti; secondariamente, in caso di nuovo evento di traffico, deve calcolare le applicazioni che si trovano all'interno della zona circolare centrata nella posizione dell'evento (e con raggio dato) e notificarle; infine, deve agire da tramite tra l'applicazione mobile e il GestoreDatabase, ricevendo le segnalazioni dalle applicazioni e inoltrandole a quest'ultimo.

#### *GestoreCentraline*

La classe GestoreCentraline è costruita usando il design pattern singleton. Al pari del GestoreApplicazioni, il suo compito è quello di gestire le centraline, cioè ricevere da esse i dati per poi inoltrarli al GestoreDatabase. A differenza del gestore delle applicazioni, il gestore centraline non deve occuparsi del processo inverso, ovvero quello di inviare notifiche a queste ultime.

#### *GestoreUtenti*

La classe GestoreUtenti è costruita usando il design pattern singleton. Il suo compito è soltanto quello di salvare gli utenti registrati al servizio, memorizzando un elenco di oggetti di tipo Utente.

### *GestoreAmministratori*

La classe GestoreAmministratori è costruita usando il design pattern singleton. Al pari del GetoreUtenti ha il compito di memorizzare l'elenco di oggetti di tipo Amministratore, e gestire quindi l'autenticazione di uno di essi nel momento in cui viene fatto il login.

### *Applicazione mobile*

La classe ApplicazioneMobile si occupa di raccogliere le notifiche che il GestoreApplicazioni invia alle applicazioni (creando una nuova NotificaApplicazione e inserendola in una lista), e di creare una NotificaApplicazione quando l'utente preme il bottone per segnalare il sistema centrale di un evento di traffico. Per svolgere queste operazioni, ApplicazioneMobile è costruita utilizzando il design pattern Factory Method per creare NotificaApplicazione. Inoltre gestisce l'accesso e la registrazione degli utenti. Per quanto riguarda la comunicazione con il GestoreApplicazioni, questa avviene tramite RMI, mentre la rilevazione della posizione è svolta attraverso il sensore GPS del telefono su cui è installata.

### *Centralina (astratta)*

La classe astratta Centralina raccoglie gli attributi e i metodi in comune tra CentralinaStradale e CentralinaAutomobilistica. Per questo motivo i suoi attributi sono protetti e non privati. Inoltre è collegata alla classe astratta RilevatoreVelocità.

### *Centralina stradale*

La classe CentralinaStradale è costruita utilizzando il design pattern Factory Method per creare un DatoTraffico. La centralina stradale in un intervallo di tempo raccoglie il numero di veicoli (con RilevatoreVeicoli) e la loro velocità (con RilevatoreVelocitàS) che transitano davanti alla centralina. Al termine dell'intervallo viene creato il Dato-Traffico e inviato al GestoreCentraline tramite RMI e viene calcolato il nuovo intervallo di tempo in proporzione al traffico.

### *Centralina automobilistica*

La classe CentralinaAutomobilistica è costruita utilizzando il design pattern Factory Method per creare uno StatoVeicolo. La centralina automobilistica al termine dell'intervallo di tempo (che rimane costante) raccoglie posizione (con SensoreGPS) e velocità (con RilevatoreVelocitàA) del veicolo, crea uno StatoVeicolo e lo invia al GestoreCentraline tramite RMI.

### *Utente*

La classe Utente ha il compito di descrivere le informazioni relative ad ogni specifico utente che ha installato l'applicazione sul proprio dispositivo mobile e che si è registrato al servizio, consentendo così di effettuare il login.

### *Amministratore*

La classe Amministratore, come la classe Utente, ha il compito di descrivere le informazioni relative ad ogni specifico amministratore, in modo da consentire loro il login e visualizzare quindi lo stato del sistema in forma di mappa.

### *SensoreGPS (astratta)*

La classe astratta SensoreGPS rappresenta il sensore generico necessario alla rilevazione della posizione. Essa si specializza in SensoreGPTelefono e SensoreGPSCentralina, differenziando così il rilevatore della posizione del telefono, e quindi quello associato all'applicazione mobile, da quello che caratterizza la centralina automobilistica. Ognuna di queste due classi, nel momento in cui viene chiamato il metodo rilevaPosizione(), istanzia un nuovo oggetto di tipo Posizione salvando in esso le coordinate appena rilevate.

### *Posizione*

La classe Posizione esprime la posizione GPS in termini di coordinate (latitudine e longitudine). In particolare, è necessaria per descrivere la posizione in cui si trovano l'applicazione, la centralina stradale o la centralina automobilistica, così da poter essere usata nell'eventualità in cui debba essere creata la rispettiva notifica di un evento di traffico.

### *RilevatoreVelocità (astratta)*

La classe astratta RilevatoreVelocità raccoglie gli attributi e i metodi in comune tra RilevatoreVelocitàA e RilevatoreVelocitàS, quindi presenta gli attributi protetti in modo tale da poter essere acceduti dalle sottoclassi.

### *RilevatoreVelocitàA*

La classe RilevatoreVelocitàA rappresenta il sensore necessario alla centralina automobilistica per rilevare la velocità corrente del veicolo su cui è installata.

### *RilevatoreVelocitàS*

La classe RilevatoreVelocitàS rappresenta il sensore necessario alla centralina stradale per rilevare la velocità del veicolo che transita di fronte ad essa.

### *RilevatoreVeicoli*

La classe RilevatoreVeicoli conta il numero di veicoli che transitano davanti alla centralina stradale e li salva in una variabile che viene resettata ogni volta che inizia un nuovo intervallo di tempo.

### *Notifica (astratta)*

La classe astratta Notifica rappresenta il formato generico dei dati scambiati all'interno del sistema complessivo di gestione del traffico. In particolare, essa raccoglie gli attributi e i metodi comuni alle sue sottoclassi, le quali rappresentano in modo più dettagliato i tre tipi di dati utilizzati.

### *NotificaApplicazioni*

La classe NotificaApplicazioni rappresenta il dato che viene scambiato tra l'applicazione mobile e il sistema centrale. In particolare, essa definisce sia il formato dell'informazione passata da ApplicazioneMobile a GestoreApplicazioni (il quale poi la passa a GestoreDatabase) sia il formato del dato che viene passato nel verso contrario.

### *DatoTrafico*

La classe DatoTrafico rappresenta il dato che viene scambiato tra la centralina stradale e il sistema centrale. In particolare, essa definisce il formato dell'informazione passata da CentralinaStradale a GestoreCentraline (il quale poi la passa a GestoreDatabase).

### *StatoVeicolo*

La classe StatoVeicolo rappresenta il dato che viene scambiato tra la centralina automobilistica e il sistema centrale. In particolare, essa definisce il formato dell'informazione passata da CentralinaAutomobilistica a GestoreCentraline (il quale poi la passa a GestoreDatabase).

### *DatoGenerico*

La classe DatoGenerico serve per convertire i diversi dati provenienti dalle applicazioni e dalle centraline in un formato uniforme. In questo modo il database contiene dati dello stesso tipo e non di tre tipi differenti.

## *MappaGrafica*

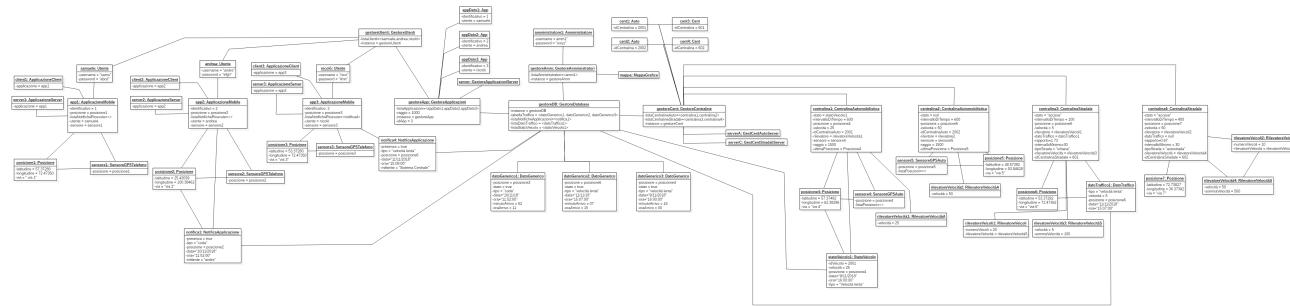
La classe Mappa viene solo accennata nel class diagram in quanto è parte dell'interfaccia grafica.

RMI

Le restanti classi e interfacce sono quelle necessarie per consentire la comunicazione remota tramite RMI. In particolare, vi sono tre tipi di comunicazione: quella da ApplicazioneMobile a GestoreApplicazioni, quella da GestoreApplicazioni a ApplicazioneMobile e quella da Centralina a GestoreCentraline. Per ognuna di esse è stata creata un'interfaccia contenente i metodi che la classe che la implementa offre per consentire quel tipo specifico di comunicazione. Tale interfaccia estende l'interfaccia Remote resa disponibile dalla piattaforma Java.

## 2.2.2 Object diagram

L'object diagram è una fotografia, in un dato momento, degli oggetti che compongono un sistema. Mostra delle istanze piuttosto che delle classi, e per questo spesso è chiamato diagramma delle istanze.



**Figura 12:** Object diagram

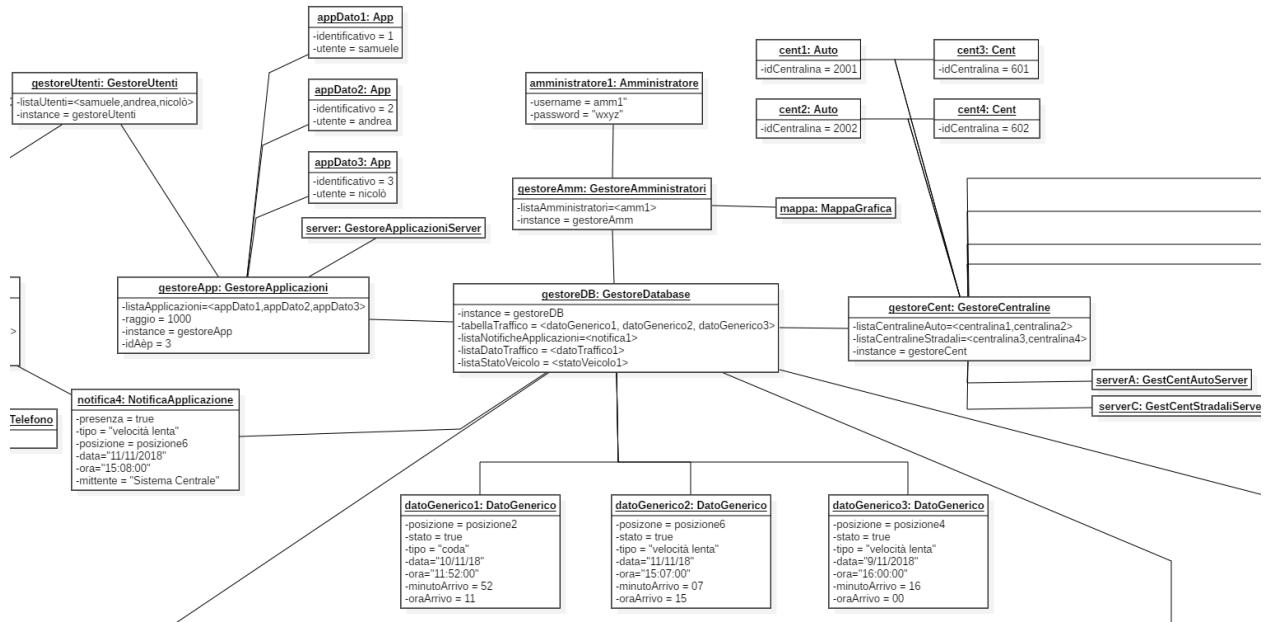


Figura 13: Object diagram - gestori

In questo object diagram, in particolare, viene analizzata la situazione in cui sono presenti, oltre ai gestori che costituiscono il sistema centrale, tre applicazioni mobili attive, associate a tre utenti diversi, due centraline automobilistiche differenti e due centraline stradali differenti. Con questa struttura si cerca di mostrare nel modo più chiaro e completo possibile come possa presentarsi il sistema complessivo nel momento in cui è in funzione.

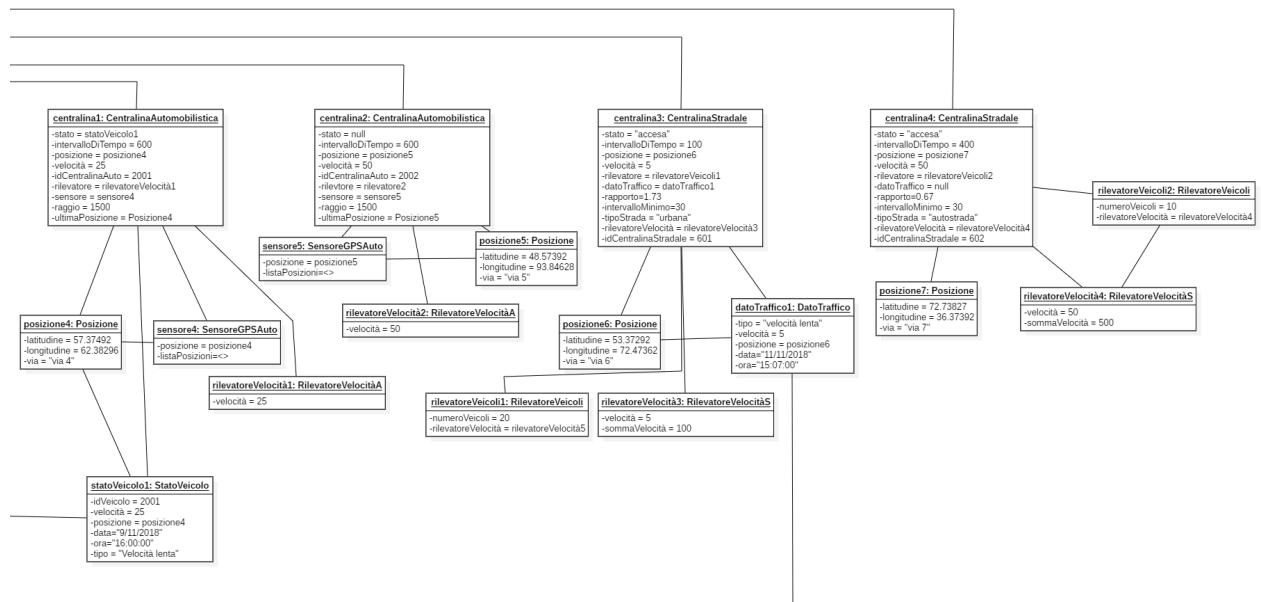


Figura 14: Object diagram - centraline

Andando più nel dettaglio, si può notare che due delle centraline,

una automobilistica e una stradale, in seguito alla scadenza del loro intervallo di tempo, hanno creato rispettivamente una notifica di tipo StatoVeicolo e una di tipo DatoTrafico. Si può notare che tali informazioni sono state già salvate nella TabellaTrafico sotto forma di dati del tipo DatoGenerico ed elaborate dal GestoreDatabase, il quale, dopo aver trovato un evento di traffico e dopo aver avvisato di ciò il GestoreApplicazioni, ha innescato il procedimento di segnalazione alle applicazioni presenti nella zona critica. È stata quindi creata una notifica del tipo NotificaApplicazioni e associata all'applicazione mobile posta nelle immediate vicinanze dell'evento (si intende che essa si trova all'interno dell'area circolare di raggio preimpostato).

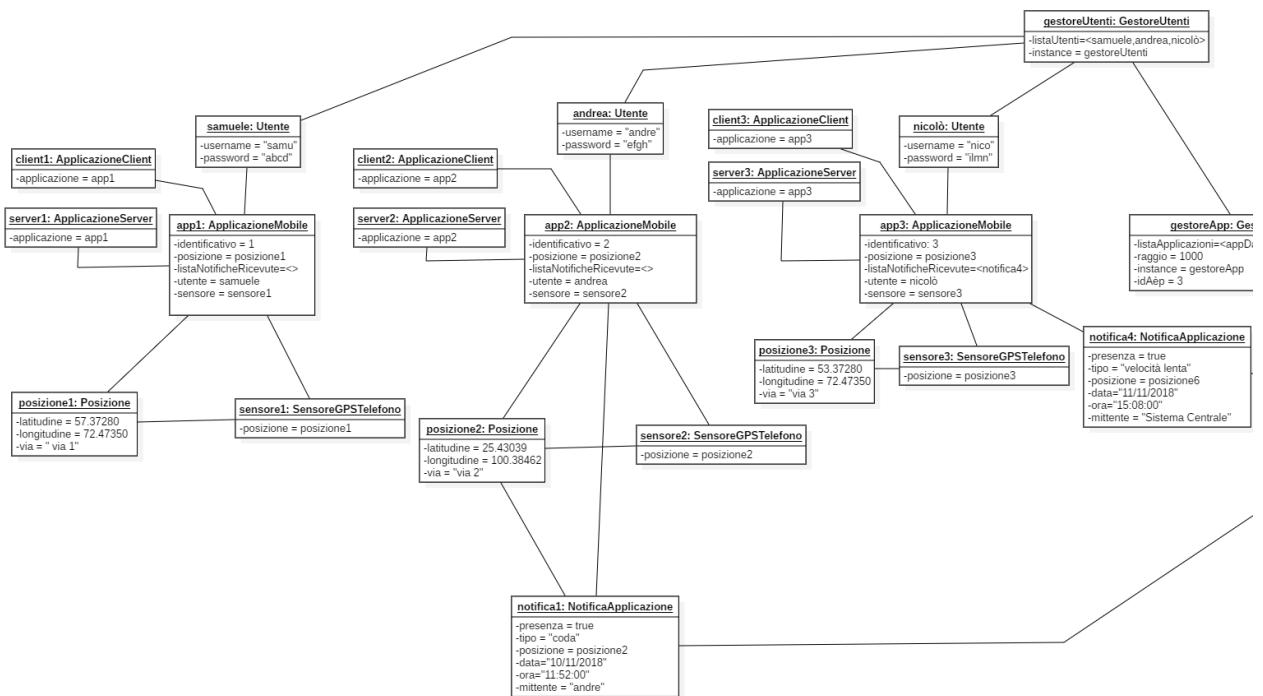


Figura 15: Object diagram - applicazioni

Al fine di mostrare tutto ciò che può accadere con il sistema complessivo in funzione, viene rappresentato anche il caso in cui sia un'applicazione a segnalare la presenza di coda nel punto in cui si trova. Essa crea quindi una notifica di tipo NotificaApplicazioni che, successivamente, tramite il GestoreApplicazioni viene inviata al GestoreDatabase, il quale la salva e la elabora.

## 2.3 STRUCTURAL MODELING – IMPLEMENTATION DIAGRAMS

### 2.3.1 Deployment diagram

Il deployment diagram documenta la distribuzione fisica di un sistema. Gli elementi principali si chiamano nodi e sono collegati da path di comunicazione.

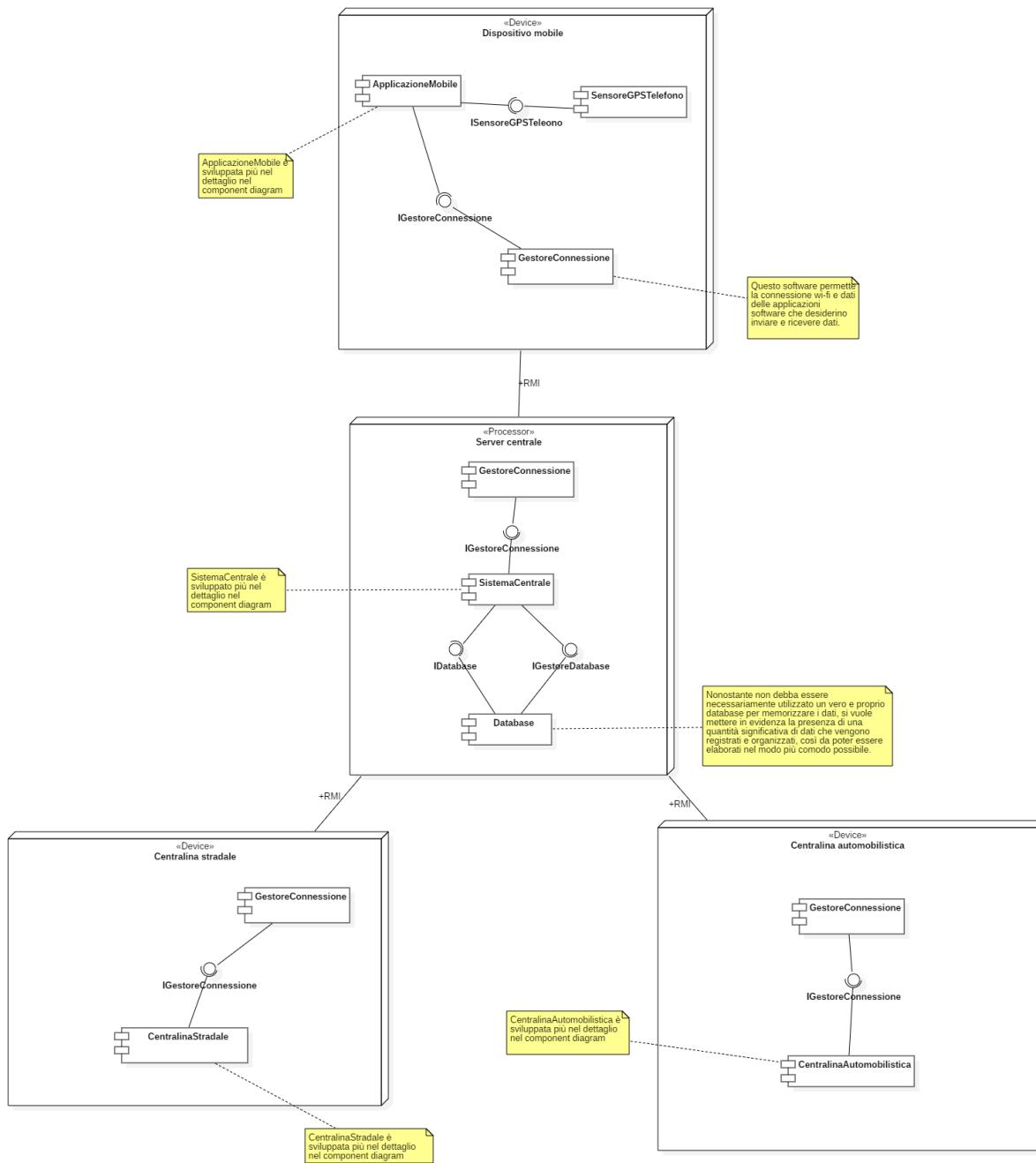


Figura 16: Deployment diagram

In questo deployment diagram vengono mostrati i componenti fisici che compongono il sistema complessivo e come comunicano tra loro. L'utente e l'amministratore, di conseguenza, non vengono rappresentati in quanto non si trattano di componenti fisici. La comunicazione tra il sistema centrale, le centraline e i dispositivi mobili avviene attraverso una connessione remota tramite RMI Java.

Il nodo server centrale rappresenta il centro di elaborazione di tutti i dati che vengono raccolti dai vari sottosistemi e, allo stesso tempo colui che manda informazioni alle applicazioni mobili. Esso è quindi costituito da un componente adibito alla gestione della connessione tramite RMI, un componente DataBase che costituisce il centro di memorizzazione dei dati e un componente SistemaCentrale (spiegato più nel dettaglio con il component diagram) che gestisce ed elabora le informazioni ricevute.

Il nodo Dispositivo mobile rappresenta appunto il dispositivo su cui viene installata l'applicazione mobile, rappresentata dall'omonimo componente (spiegato più nel dettaglio con il component diagram). Vi è poi anche qui un componente che si occupa della gestione della connessione tramite RMI, e infine un componente che gestisce il sensore GPS necessario per la rilevazione della posizione.

I due nodi rimanenti rappresentano invece i due differenti tipi di centralina (stradale e automobilistica) e contengono, rispettando la struttura degli altri nodi, un componente che gestisce le connessioni RMI e un componente rappresentante la centralina dal punto di vista software.

### 2.3.2 Component diagram

Il component diagram rappresenta le componenti software che caratterizzano il sistema complessivo e le interfacce utilizzate dai sottosistemi per comunicare tra loro.

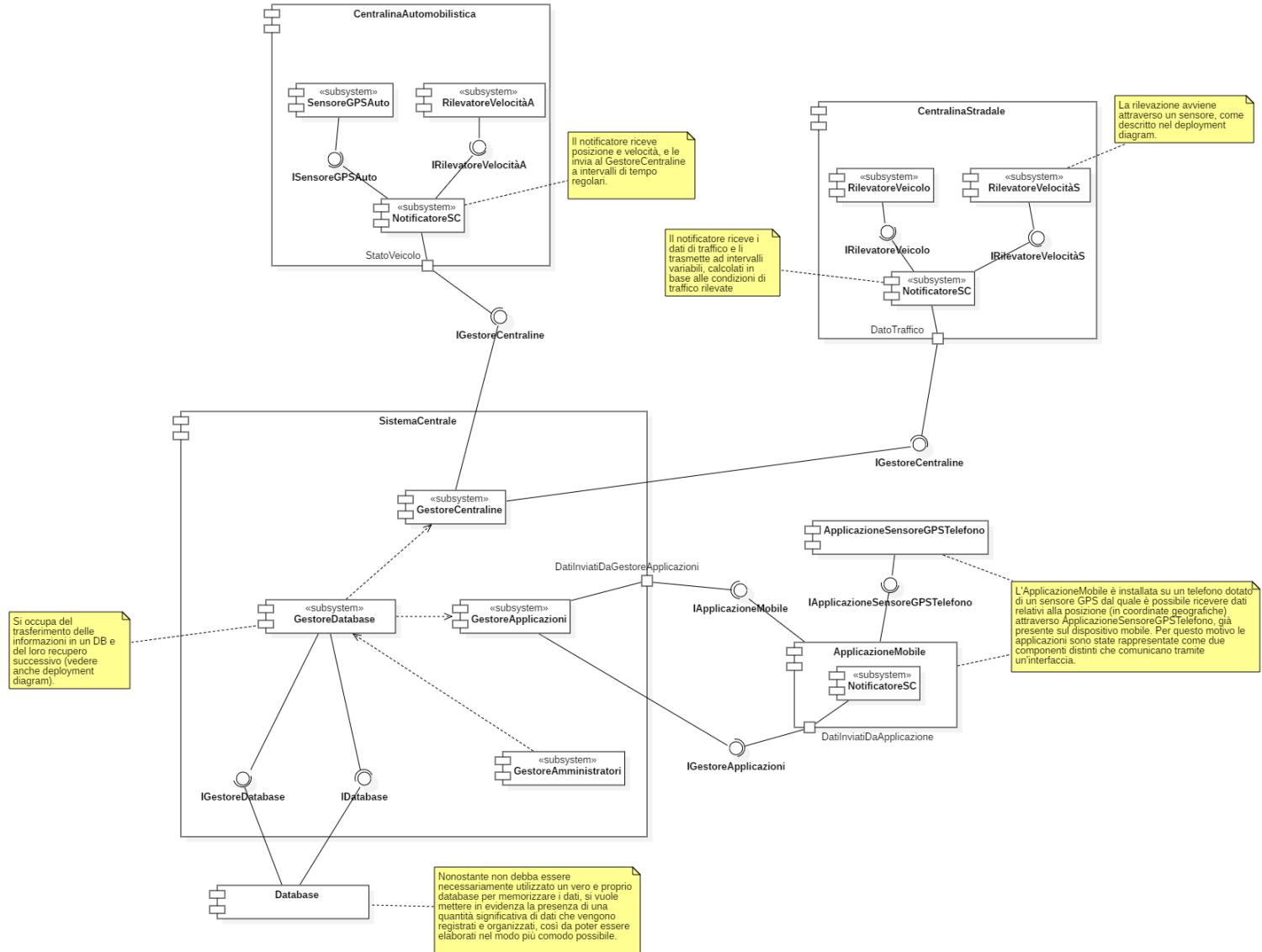


Figura 17: Component diagram

Il componente SistemaCentrale si occupa della gestione dei sottosistemi e della gestione delle informazioni raccolte. Per fare ciò, esso è suddiviso in componenti minori, ciascuno che si occupa di una specifica funzione. In particolare, il GestoreApplicazioni e il GestoreCentraline si occupano rispettivamente di tutto ciò che riguarda le applicazioni mobili e le centraline. Vi è poi un componente, GestoreDatabase, che si occupa della gestione delle informazioni ricevute,

elaborandole e interfacciandosi con il DataBase in modo da salvarle e recuperarle nel momento in cui sono necessarie.

Il componente ApplicazioneMobile contiene un componente NotificatoreSC che si occupa di gestire l'invio della notifica di presenza di coda dall'applicazione mobile al sistema centrale. Si interfaccia poi con il componente SensoreGPS per rilevare la propria posizione e poterla fornire quindi al sistema centrale, il quale, in base ad essa, sa quali notifiche deve inviarle e quali no.

Le centraline sono anche esse costituite da un componente NotificatoreSc per la comunicazione con il sistema centrale e dai componenti necessari alla rilevazione dei dati per i quali le centraline sono state installate. In particolare, la centralina stradale possiede un componente per la rilevazione del passaggio di un veicolo e uno per la rilevazione della velocità di tale veicolo, mentre la centralina automobilistica contiene un sensore per la rilevazione della posizione e uno per la misurazione della velocità del veicolo su cui è installata.

## 2.4 DYNAMIC MODELING – INTERACTION DIAGRAMS

### 2.4.1 Sequence diagrams

Il sequence diagram documenta tipicamente il comportamento di un singolo scenario. Il diagramma include un certo numero di oggetti e i messaggi scambiati tra essi durante l'esecuzione in ordine temporale dello scenario.

#### *Centralina stradale*

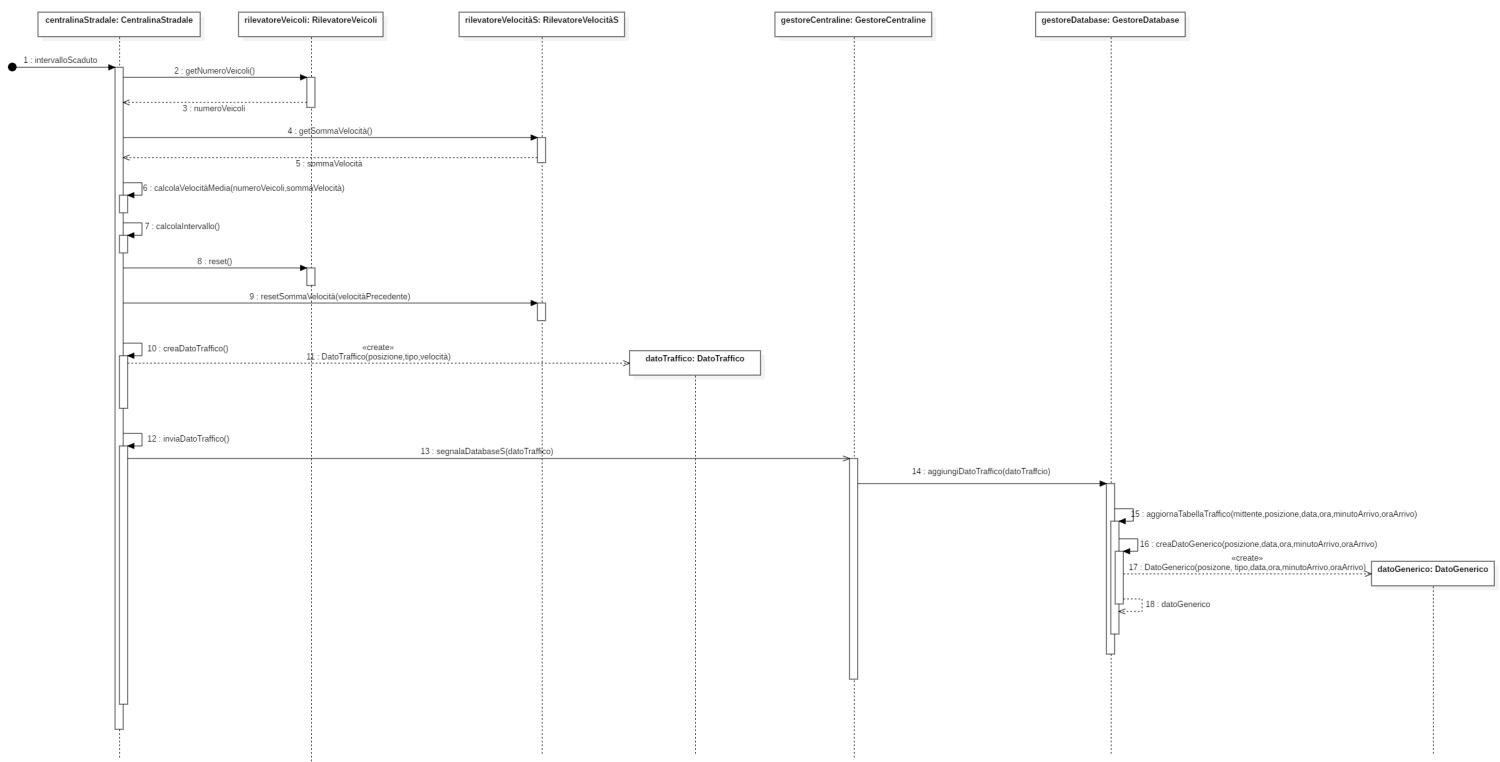


Figura 18: Sequence diagram CentralinaStradale

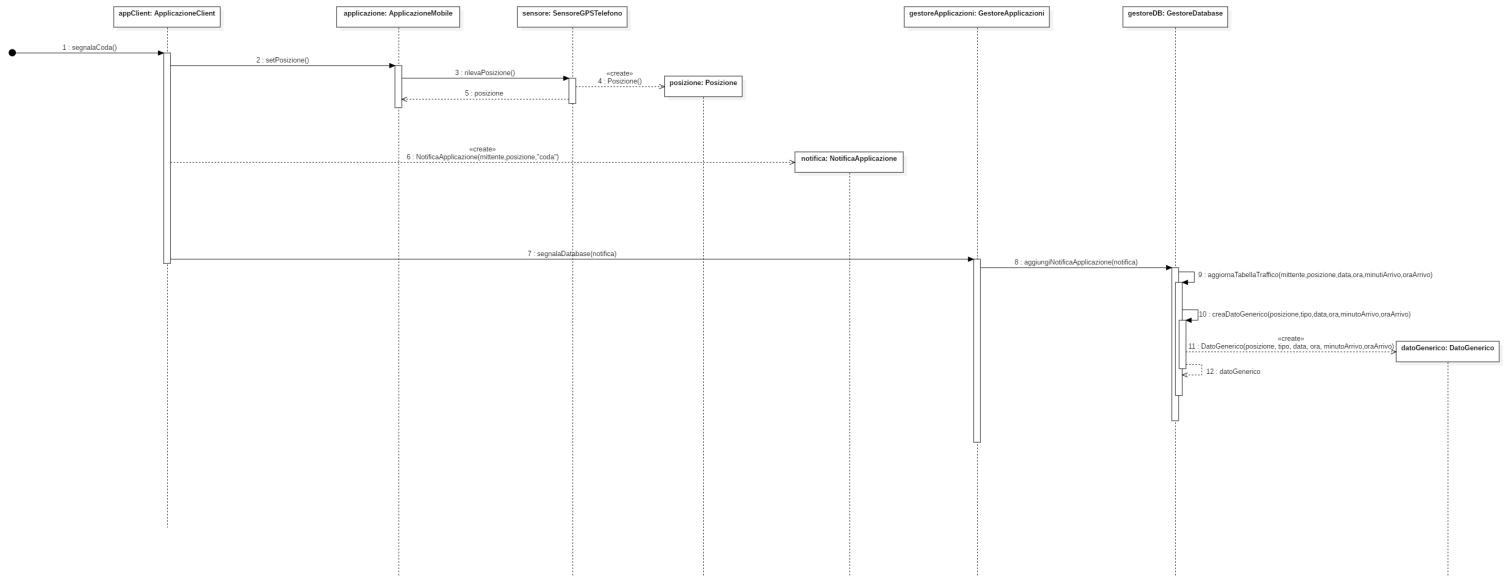
In questo sequence diagram è rappresentato ciò che comporta lo scadere dell'intervallo di tempo di una centralina stradale.

Terminato l'intervallo di tempo, la centralina stradale richiede il numero di veicoli che sono stati rilevati dall'apposito sensore rappresentato dalla classe `RilevatoreVeicoli` e la somma delle loro velocità al `RilevatoreVelocitàS`. In base alla quantità di vetture transitate davanti alla centralina, viene calcolato il nuovo intervallo di tempo e la velocità media.

In seguito, la centralina stradale crea un dato di traffico configurando i suoi parametri. La notifica sulla situazione del traffico viene inviata al `GestoreCentraline`, il quale la inoltra al `GestoreDatabase`.

La sequenza di azioni termina con la creazione di un DatoGenerico che viene inserito nella TabellaTraffico.

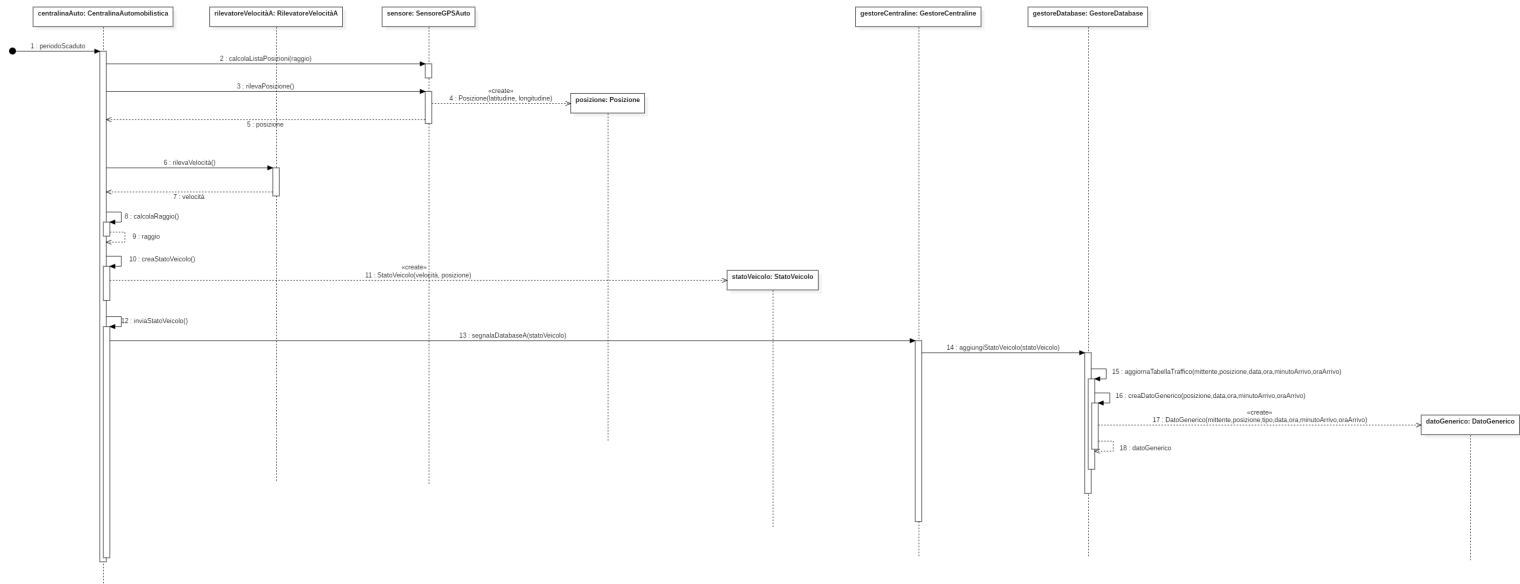
### *Applicazione mobile segnala la coda*



**Figura 19:** Sequence diagram ApplicazioneMobile segnala un evento di coda

In questo sequence diagram è rappresentato il meccanismo che gestisce la segnalazione della presenza di coda da parte di un utente. Nel momento in cui viene premuto il bottone dell’interfaccia grafica offerto dall’applicazione, viene subito avviato il procedimento di segnalazione: l’applicazione richiede la posizione attuale al SensoreGPS a lei associato, successivamente crea una notifica del tipo NotificaApplicazione e la invia immediatamente al GestoreApplicazioni. Quest’ultimo, non appena ricevuto il dato, lo inoltra al GestoreDatabase il quale crea un DatoGenerico, lo salva e aggiorna la TabellaTraffico contenente tutti i dati fino ad ora ricevuti.

### Centralina automobilistica



**Figura 20:** Sequence diagram CentralinaAutomobilistica

In questo sequence diagram viene mostrato cosa avviene quando scade il periodo di tempo della centralina stradale.

La centralina stradale al termine del periodo rileva la velocità del veicolo (grazie al RilevatoreVelocitàA) e la posizione corrente (con il SensoreGPSAuto). Successivamente crea uno StatoVeicolo e lo invia al GestoreCentraline, il quale lo inoltra al GestoreDatabase. Quest'ultimo aggiorna la TabellaTraffico creando e inserendo un DatoGenerico.

### 2.4.2 Collaboration diagram

Il collaboration o communication diagram mostra le comunicazioni che intercorrono tra diversi oggetti in una situazione particolare, enfatizzando lo scambio dei dati tra gli oggetti. La sequenza temporale è rappresentata enumerando i messaggi scambiati durante l'esecuzione dello scenario.

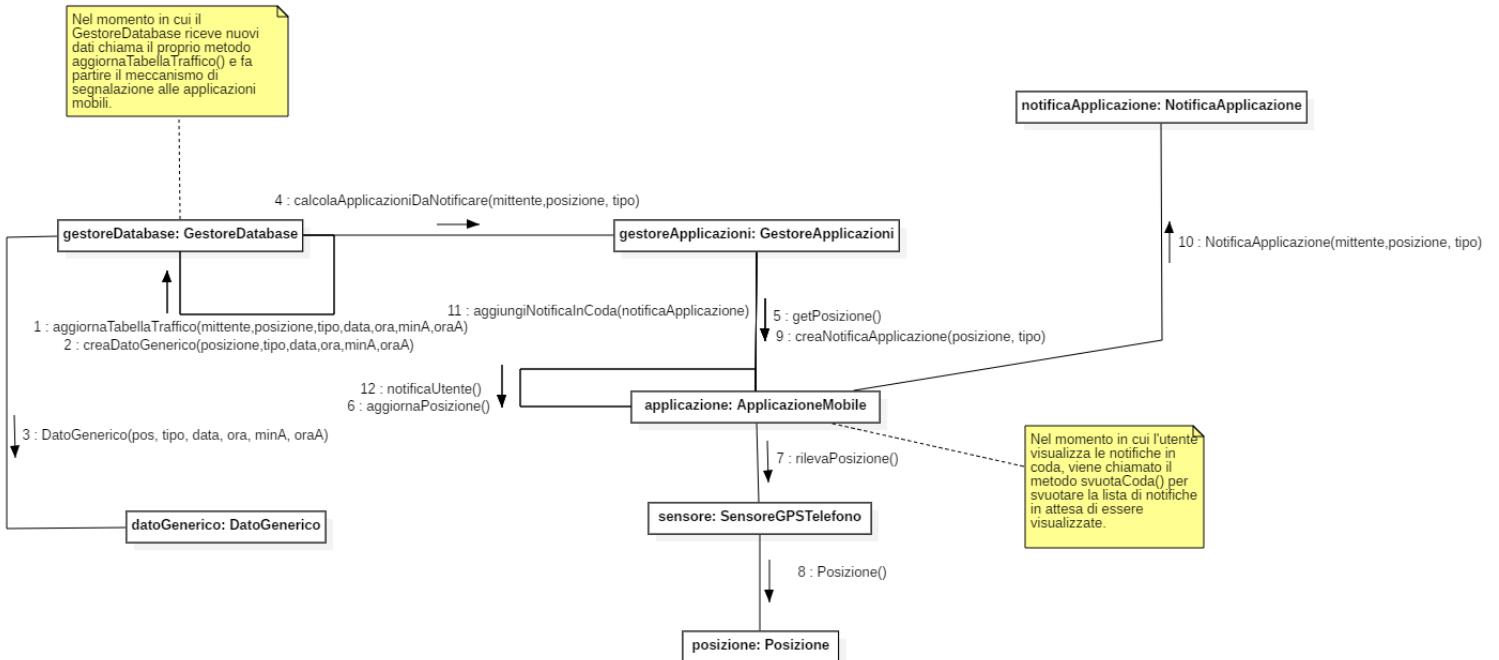


Figura 21: Collaboration diagram

In questo collaboration diagram, si analizzano gli oggetti e i metodi coinvolti quando il GestoreDatabase vuole notificare le applicazioni riguardo a un nuovo evento di traffico appena ricevuto. Questo evento di traffico può essere uno StatoVeicolo, un DatoTraffico o una NotificaApplicazione.

In questa situazione, il GestoreDatabase aggiorna la tabella di traffico creando un DatoGenerico e chiama il metodo di GestoreApplicazioni per calcolare quali sono le applicazioni da notificare in base alla posizione dell'evento di traffico ricevuto.

Il GestoreApplicazioni chiede a ogni applicazione la posizione (che viene calcolata grazie al sensoreGPS) e valuta se è all'interno del cerchio coinvolto nell'evento di traffico. Nel caso lo fosse, viene creata una NotificaApplicazione, aggiunta in coda all'applicazione e infine segnalato l'utente.

## 2.5 DYNAMIC MODELING – STATECHART DIAGRAMS

Lo statechart diagram descrive il comportamento di un oggetto per la durata del suo ciclo di vita, mostrando gli stati che assume, gli eventi a cui risponde, le risposte che fornisce e le transizioni tra gli stati.

### *Centralina stradale*

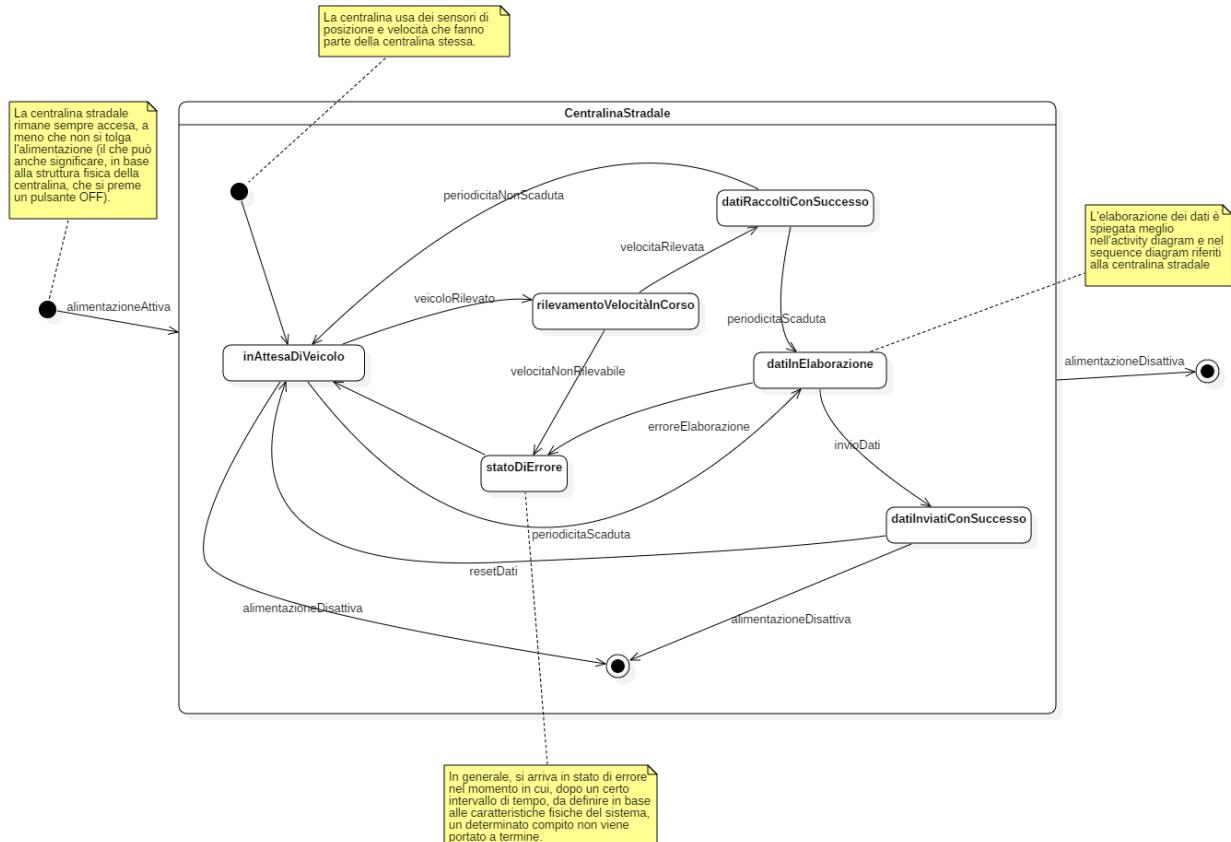


Figura 22: State diagram CentralinaStradale

La centralina stradale si occupa del rilevamento dei dati di traffico, che devono essere inviati al sistema centrale ad intervalli di tempo (periodicità) variabili a seconda delle condizioni del traffico rilevate.

La centralina può essere accesa o spenta. In particolare, potrà essere spenta scollegando l'alimentazione oppure utilizzando un tasto apposito. Questi dettagli, però, vengono definiti sulla base della struttura fisica della centralina e, di conseguenza, non sono parte del progetto.

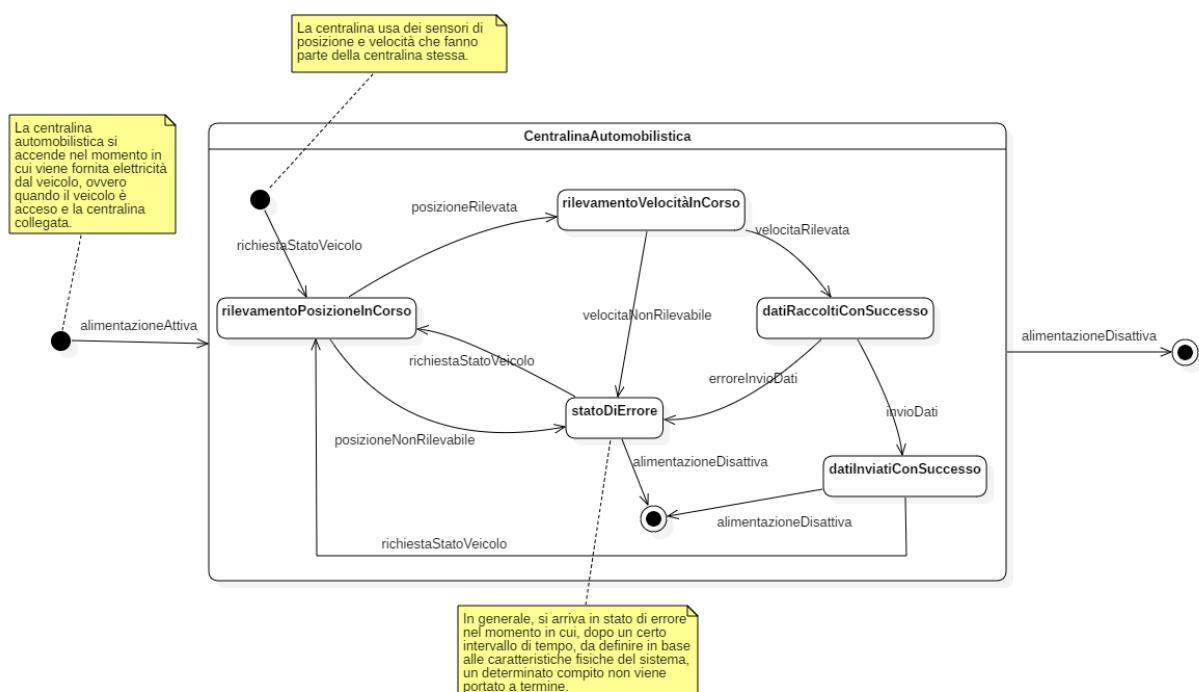
Il sistema centrale richiede i dati di traffico alla centralina, la quale, in seguito alla richiesta, attende che transiti un veicolo da rilevare, continuando ad aggiornare un clock interno per la misurazione del tempo trascorso.

Nel momento in cui un veicolo viene rilevato, ne viene rilevata anche la velocità e viene valutato il tempo rimanente prima che i dati debbano essere inviati al sistema centrale. Se il tempo non è ancora scaduto, si attende un nuovo veicolo e si ripetono i passaggi già descritti. In caso contrario i dati raccolti vengono elaborati e inviati al sistema centrale, resettando il conteggio del tempo.

Se, invece, la periodicità scade mentre la centralina sta attendendo il veicolo, invia i dati raccolti fino a quel momento.

Nel caso in cui una qualsiasi operazione non venga portata a termine, la centralina torna in attesa di un veicolo da rilevare.

Centralina automobilistica



**Figura 23:** State diagram CentralinaAutomobilistica

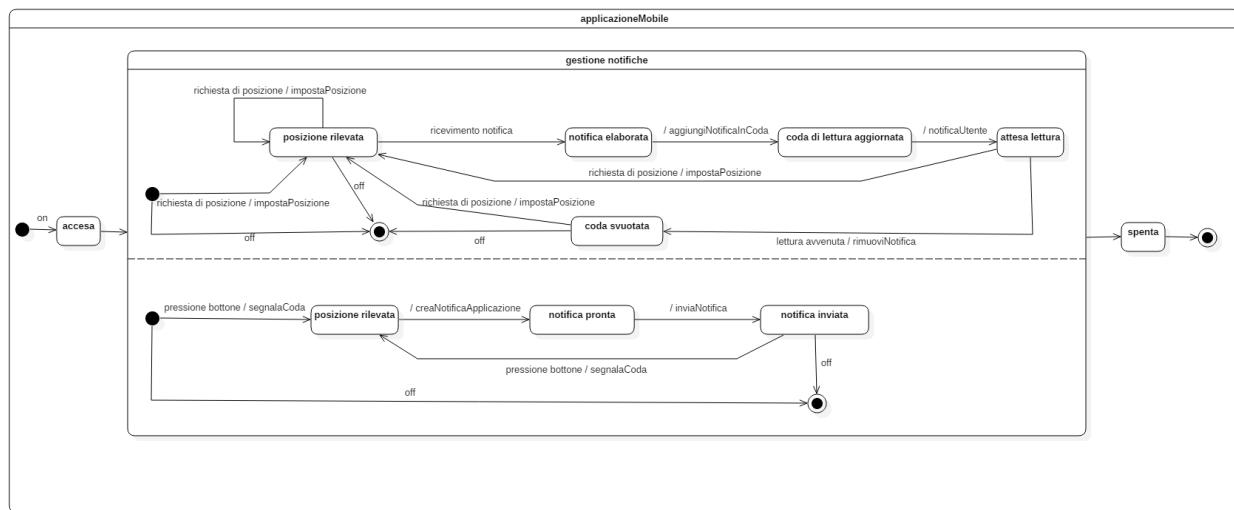
La centralina automobilistica si occupa del rilevamento dello stato del veicolo, che deve essere inviato al sistema centrale ad intervalli di tempo che, a differenza di quanto accade nel caso della centralina stradale, sono regolari.

La centralina può essere accesa o spenta. In particolare, potrà essere spenta scollegando l'alimentazione (questa situazione si verifica, per esempio, quando viene spenta l'automobile).

Il sistema centrale richiede lo stato del veicolo alla centralina, la quale, in seguito alla richiesta, rileva la posizione del veicolo in quel momento e la sua velocità. Successivamente i dati raccolti vengono inviati al sistema centrale e si inizia nuovamente il processo.

Nel caso in cui un'operazione non venga portata a termine correttamente (ed entro un certo intervallo di tempo), la centralina inizia nuovamente a rilevare posizione e velocità del veicolo. Il motivo per cui non vengono recuperati i dati da inviare è che questi diventano presto obsoleti e, di conseguenza, poco utili.

## *Applicazione mobile*



**Figura 24:** State diagram ApplicazioneMobile

L'applicazione mobile si può trovare in tre stati: accesa, spenta, oppure gestione delle notifiche. Per entrare nello stato "accesa" è necessario un evento "on", in "spenta" è necessario ricevere "off", mentre nello stato "gestione notifiche" si accede immediatamente dopo l'accensione.

L'applicazione può ricevere notifiche dal GestoreApplicazioni oppure può inviare a sua volta notifiche al GestoreApplicazioni. Per questo motivo la gestione delle notifiche è rappresentato come un sottosistema che esegue entrambi i compiti in parallelo.

Nella parte superiore viene rappresentata la ricezione di una possibile nuova notifica di traffico da mostrare all'utente, che inizia con una richiesta di impostazione della posizione, se l'applicazione è all'interno del raggio di notifica prosegue con la ricezione della notifica e termina con la gestione della coda di notifiche da leggere, altrimenti rimane in rilevamento posizione aspettando una nuova richiesta.

Nella parte inferiore invece è mostrato come si comporta l'applicazione quando l'utente preme il bottone, cioè dopo la pressione viene rilevata la posizione, scritta la notifica e inviata.

## 2.6 DYNAMIC MODELING – ACTIVITY DIAGRAMS

Un activity diagram illustra i flussi tra le attività associate a un determinato scenario mostrando gli oggetti, le transizioni, i branch, i merge, le fork e le join.

*Visualizzazione stato traffico da parte dell'amministratore*

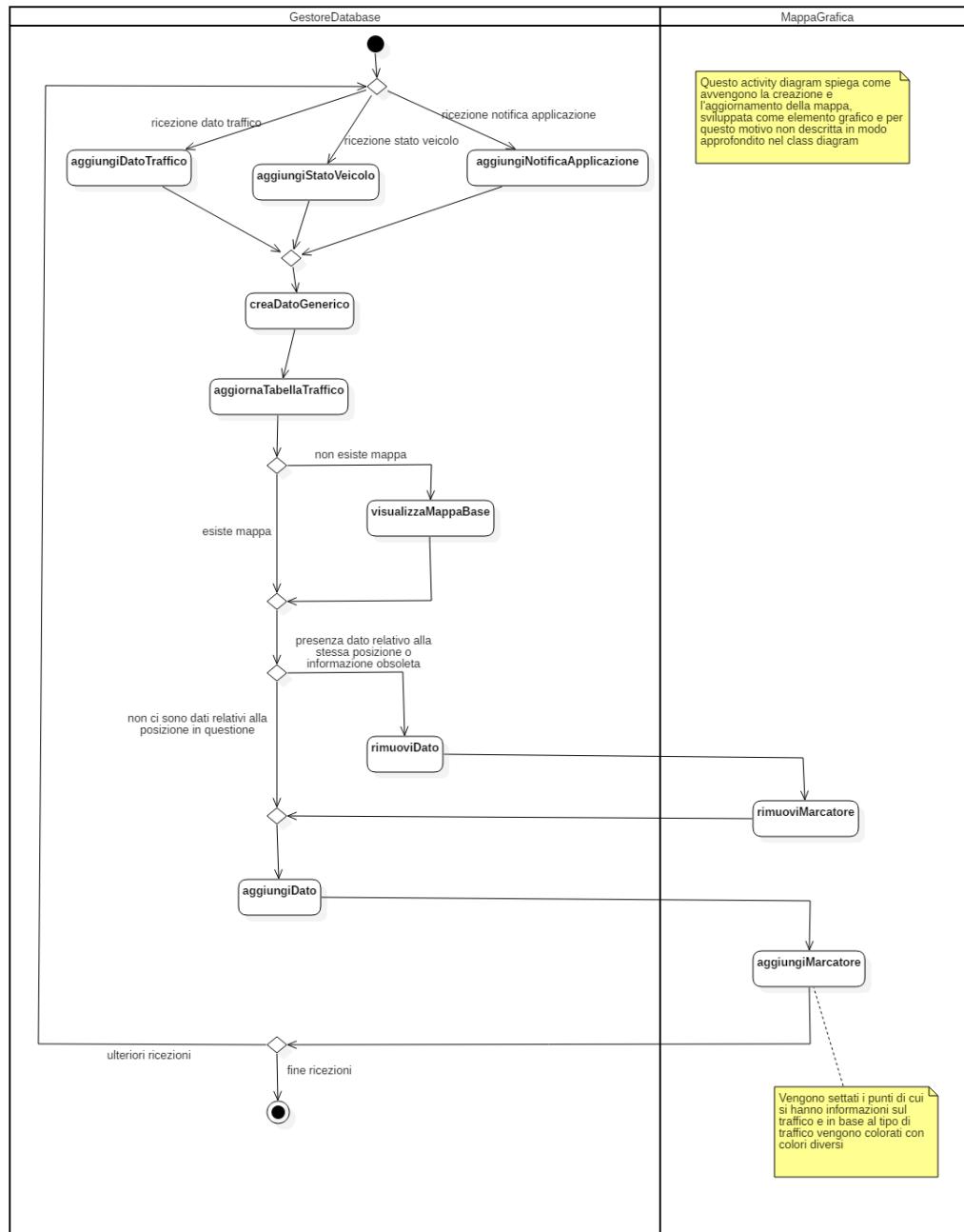


Figura 25: Activity diagram aggiornamento tools amministratore

La situazione che si vuole rappresentare in questo activity diagram è l'aggiornamento dei tools a disposizione dell'amministratore per la consultazione del traffico.

Il GestoreDatabase riceve uno statoVeicolo o un DatoTrafico o una Notifica applicazione, lo aggiunge alla lista corrispondente, lo converte in un DatoGenerico e aggiorna la TabellaTrafico e la MappaGrafica. In particolare, se quest'ultima non era ancora stata aperta, essa viene creata e successivamente viene aggiunto un marcitore alla mappa. Tale marcitore viene inserito nella posizione indicata all'interno della notifica ricevuta e viene colorato in modo diverso a seconda del tipo di evento di traffico notificato.

Nel caso in cui sia già presente un altro marcitore nella stessa posizione in cui deve essere inserito quello relativo alla notifica appena ricevuta, esso viene rimosso e quindi sostituito da quello più recente. Allo stesso tempo, tutti i marcatori associati ad informazioni obsolete (cioè presenti nella mappa da oltre tre minuti) vengono rimossi, così da mantenere la MappaGrafica pulita e aggiornata.

### Monitoraggio del traffico stradale e notificare le applicazioni

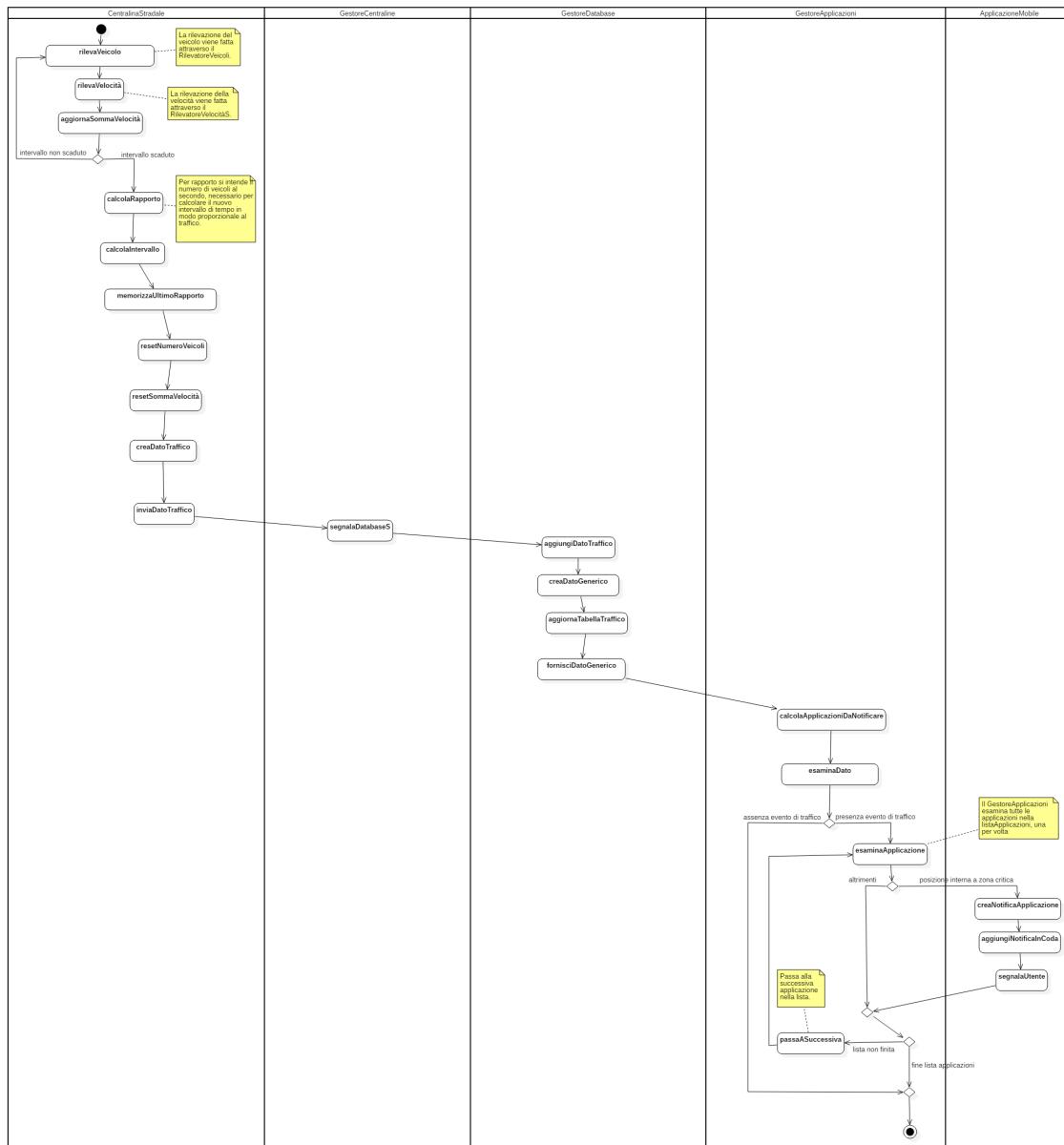


Figura 26: Activity diagram centralina stradale e applicazioni

In questo activity diagram si vuole descrivere il procedimento con cui la centralina stradale monitora il traffico nel segmento di strada in cui è installata. In particolare, attraverso i sensori di rilevazione dei veicoli e di rilevazione della velocità essa tiene traccia del numero di veicoli che le passano davanti, rilevandone per ognuno la velocità.

Nel momento in cui scade l'intervallo di tempo, viene calcolato il nuovo intervallo, resettati i conteggi e creato un DatoTraffico. Il Dato-Traffico creato viene inviato al GestoreCentraline, il quale, a sua volta

lo manda al GestoreDatabase che memorizza il dato convertendolo in DatoGenerico.

Successivamente il controllo passa al GestoreApplicazioni che calcola le applicazioni da notificare, esaminando prima il dato ricevuto e poi chiedendo a ogni applicazione la posizione. Per ogni applicazione da notificare viene creata una NotificaApplicazione e viene segnalato l'utente.

# 3

## IMPLEMENTAZIONE IN JAVA

Il progetto complessivo è costituito da quattro progetti, collegati tramite RMI. I progetti realizzati sono **Sistema centrale**, **CentralinaStradale**, **CentralinaAuto** e **Applicazione mobile**.

### 3.1 SISTEMA CENTRALE

Il sistema centrale ha il compito di raccogliere tutti i dati che arrivano dalle applicazioni e dalle centraline, disporli in una mappa geografica (sotto forma di marcatori di colori diversi) e renderla accessibile agli amministratori, rimuovere le segnalazioni meno recenti (dopo tre minuti dalla segnalazione, il punto sulla mappa viene rimosso) e calcolare le applicazioni che devono essere notificate (vengono notificate le applicazioni all'interno di un certo raggio intorno alla segnalazione) e inviare a queste una notifica.

Il sistema centrale è composto da cinque gestori: `GestoreDatabase`, `GestoreApplicazioni`, `GestoreAmministratori`, `GestoreUtenti`, `GestoreCentraline`. Il `GestoreDatabase` si occupa della gestione di tutte le notifiche delle applicazioni e delle centraline che vengono inviate al sistema centrale. Per uniformare il tipo di notifica esso le trasforma in `DatoGenerico` e le inserisce in una tabella di traffico. Inoltre quando avviene una nuova ricezione, il `GestoreDatabase` fa partire il calcolo delle applicazioni da notificare. Il `GestoreApplicazioni` si occupa di tenere traccia di tutte le applicazioni mobili creando una lista di `App`, di raccogliere le notifiche in arrivo dalle applicazioni e di inviare a queste le notifiche mandate dal `GestoreDatabase`. Il `GestoreAmministratore` gestisce le informazioni relative a tutti gli amministratori, mentre il `GestoreUtenti` quelle relative agli utenti delle applicazioni mobili. Il `GestoreCentraline` possiede la lista delle `Cent` (stradali) e delle `Auto` e riceve i dati di traffico che arrivano dalle centraline stradali e automobilistiche.

La classe `FunzionamentoSistemaCentrale` apre una finestra di Login/Registrazione di un amministratore. Quando viene effettuato il login viene aperta la mappa e vengono messi online i server che raccolgono informazioni dalle applicazioni e dalle centraline.

### 3.2 CENTRALINA STRADALE

La centralina stradale è composta da un **RilevatoreVeicoliS** e da un **RilevatoreVelocita**. Quando la centralina è in funzione, il rilevatore di veicoli ogni tre secondi sceglie casualmente tra 0 e 1, zero indica che non è passato nessun veicolo nei tre secondi trascorsi, mentre uno che ne è passato uno nel medesimo intervallo. Quando un veicolo viene rilevato, il **RilevatoreVelocita** calcola la sua velocità (in un intorno di  $\pm 15$  km/h rispetto alla velocità media) e la somma alle velocità dei veicoli precedenti, memorizzando quindi tale valore in un attributo **sommaVelocita**. Allo scadere di un intervallo di tempo, inizialmente fissato e poi calcolato in base al numero di veicoli transitati, viene generato un **DatoTrafico** che si basa sulla velocità media e sul tipo di strada in cui è posizionata la centralina. Il dato di traffico viene poi inviato al **GestoreCentraline** e vengono azzerati i conteggi di veicoli e velocità.

La classe **FunzionamentoCentralinaS** fa partire i run della **CentralinaStradale** e del **RilevatoreVeicoli** e così facendo si stabilisce anche la connessione RMI (lato client) con il **GestoreCentraline**.

La classe astratta **Centralina** è stata implementata perché nel momento in cui si volesse installare altri tipi di centralina, sarebbe possibile ereditare da questa gli attributi e i metodi fondamentali per una generica centralina.

Anche la centralina è dotata di un'interfaccia grafica per la sua configurazione (è possibile impostare i seguenti parametri: via o piazza, tipo di strada, velocità iniziale impostata manualmente o casuale) e il suo utilizzo. In particolare premendo il bottone "Imponi una velocità" si può cambiare la velocità media del tratto stradale su cui è installata la centralina.

### 3.3 CENTRALINA AUTOMOBILISTICA

La centralina automobilistica è stata realizzata in modo simile alla centralina stradale. Le principali differenze consistono nella gestione dell'intervallo di tempo, della velocità e della posizione. Per questo motivo la centralina stradale è composta da un **SensoreGPSAuto** e da un **RilevatoreVelocitaA**, mentre non possiede il **RilevatoreVeicoli**.

La centralina stradale possiede un intervallo di tempo fisso pari a 90 secondi. Allo scadere di tale intervallo la centralina calcola tramite il sensore GPS una lista di posizioni raggiungibili in base alla distanza in km e alla velocità, seleziona una di queste posizioni e si sposta rilevando la velocità della nuova strada. La velocità varia in un intorno di -20 e +15 rispetto alla velocità precedente (in questo modo è più probabile ci sia traffico).

Se la velocità dovesse essere pari a zero, allo scadere dell'intervallo l'auto non si sposta dalla posizione in cui si trova. Se invece la velocità è diversa da zero ma la posizione successiva è uguale a quella attuale, il raggio (che misura la distanza in km che può percorrere in un intervallo di tempo) raddoppia. In tutti gli altri casi, il raggio è calcolato in base alla velocità del tratto stradale (considerando un moto rettilineo uniforme).

Scaduto l'intervallo viene inoltre creato uno StatoVeicolo, che contiene la velocità e la posizione dell'auto, e viene inviato al Sistema Centrale tramite la connessione RMI con il GestoreCentraline.

La centralina automobilistica presenta un'interfaccia grafica molto semplice in cui viene mostrata l'ultima posizione e l'ultima velocità rilevate.

### 3.4 APPLICAZIONE

L'applicazione svolge due ruoli: ricevere le notifiche dal sistema centrale mostrandole all'utente, e inviare a sua volta notifiche al GestoreApplicazioni. Per questo motivo oltre alla classe ApplicazioneMobile, sono state implementate le classi ApplicazioneClient e ApplicazioneServer, per gestire al meglio la comunicazione in entrambi i lati.

La classe FunzionamentoApplicazione crea l'applicazione mobile e permette di eseguire i run dell'applicazione client e dell'applicazione server in contemporanea.

Il lato server dell'applicazione riceve le notifiche dal sistema centrale e le mostra all'utente mettendo in alto la più recente. L'utente può decidere di svuotare la lista delle notifiche in qualsiasi momento. Il lato client invece crea le notifiche applicazione e le invia al sistema centrale.

La scelta della posizione dell'applicazione avviene casualmente tra una lista di posizioni presente in un foglio di calcolo (queste posizioni sono le uniche valide, anche per la configurazione delle centraline). Si può anche decidere di fissare la posizione dell'applicazione nell'ultima posizione rilevata. L'applicazione gestisce la sua posizione anche grazie al sensore GPS del telefono, che qui viene gestito come un rilevatore di posizioni casuali.

L'applicazione mobile si presenta inizialmente con una grafica per il Login/Registrazione. Avvenuto il login vengono mostrate le notifiche e i pulsanti per l'utilizzo dell'applicazione.

## 3.5 ALGORITMI SIGNIFICATIVI

### 3.5.1 Calcolo delle applicazioni da notificare

---

```

if (!(this.listaApplicazioni.isEmpty())) {
    for (App var: this.listaApplicazioni) {
        if ((var.getUtente().getUsername() != null) && (! (var.
            getUtente().getUsername().equals(mittente))) {
            IApplicazioneMobile appServer=this.
                getInterfacciaApp(var.getId());
            double R = 6378.137;
            double lat1=posizione.getLatitude();
            double lat2=appServer.getPosizione().
                getLatitude();
            double lon1=posizione.getLongitude();
            double lon2=appServer.getPosizione().
                getLongitude();
            double dLat = lat1 * Math.PI/180 - lat2 * Math.
                PI/180;
            double dLon = lon1 * Math.PI/180 - lon2 * Math.
                PI/180;
            double a = Math.sin(dLat/2) * Math.sin(dLat/2) +
                Math.cos(lat1 * Math.PI/180) * Math.cos(
                lat2 * Math.PI/180) * Math.sin(dLon/2) *
                Math.sin(dLon/2);
            double c = 2 * Math.atan2(Math.sqrt(a), Math.
                sqrt(1-a));
            double d = R * c;

            if (d*1000<this.raggio) {
                NotificaApplicazione notifica;    notifica=
                    appServer.creaNotificaApplicazione(
                    posizione, tipo);
                appServer.aggiungiNotificaInCoda(notifica);
            }
        }
    }
}

```

---

L'algoritmo mostrato rappresenta il procedimento seguito dal GestoreApplicazioni per determinare quali applicazioni notificare in seguito alla segnalazione di un evento di traffico.

In particolare, se la lista non è vuota, viene eseguito un ciclo su tutti gli elementi dell'array, il quale contiene i dati delle applicazioni notificabili, in modo da trovare quelle a cui si è connesso un utente, il quale quindi ha fatto il login o si è registrato, verificando però che non sia lo stesso utente che ha inviato quella segnalazione.

Per ogni applicazione che rispetti questi criteri viene chiamato il metodo che trova nel registry RMI l'interfaccia di tale applicazione e viene calcolata la distanza in metri tra la posizione dell'evento di traffico e quella dell'applicazione; nel caso in cui tale distanza sia minore del raggio predefinito, e quindi nel caso in cui l'applicazione rientri nella zona circolare critica, questa riceve la notifica appena creata.

### 3.5.2 Aggiornamento della tabella di traffico

```
DatoGenerico datoGenerico=creaDatoGenerico(pos, tipo, data,
    ora, minA, oraA);
GregorianCalendar dat = new GregorianCalendar();
for (int i=this.tabellaTraffico.size()-1;i>=0;--i) {

    int oraAttuale=dat.get(Calendar.HOUR);
    int minAttuale=dat.get(Calendar.MINUTE);
    int minArr=this.tabellaTraffico.get(i).getMinA();
    int oraArr=this.tabellaTraffico.get(i).getOraA();

    if (this.tabellaTraffico.get(i).getPosizione().equals(
        pos)||((oraAttuale==oraArr&&(minAttuale-minArr>2))
        ||(oraAttuale>oraArr&&minAttuale>0)||((oraAttuale
        ==0&&oraArr==11&&minAttuale>0))) {
        try {
            FunzionamentoSistemaCentrale.getMappa().
                rimuoviMarcatore(this.tabellaTraffico.get(i)
                    .getPosizione().getLatitudine(), this.
                    tabellaTraffico.get(i).getPosizione().
                    getLongitudine());
            this.tabellaTraffico.remove(i);
        }catch(Exception e) {
            JOptionPane.showMessageDialog(null, "La mappa
                non e' disponibile.");
        }
    }
}

this.tabellaTraffico.add(datoGenerico);

try {
    FunzionamentoSistemaCentrale.getMappa().aggiungiPunto(
        datoGenerico);
}
catch(Exception e) {
```

```

        JOptionPane.showMessageDialog(null, "La mappa non e' disponibile.");
    }

    if (!(datoGenerico.getTipo()).endsWith("Traffico nella norma"))
    ){
        GestoreApplicazioni.getInstance().
            calcolaApplicazioniDaNotificare(mittente, pos,
                tipo);
    }

```

---

L'algoritmo mostrato rappresenta il metodo con cui viene aggiornata la tabellaTrafico (array contenente tutti i dati ricevuti) e di conseguenza la mappa.

In particolare, una volta creato il dato di tipo `DatoGenerico`, si scorre la tabella di traffico con un ciclo `for` al fine di verificare per prima cosa se sono presenti dati obsoleti<sup>1</sup>, e quindi cancellarli, e in secondo luogo per verificare se sono già presenti dati relativi alla posizione della nuova notifica ricevuta e nel caso sostituirli. Dopodiché viene inserito il dato nella mappa sotto forma di marcatore con un colore dipendente dal tipo di evento e, successivamente, nel caso in cui il dato descriva un evento di traffico critico, viene chiamata la funzione `calcolaApplicazioniDaNotificare()` (descritta nell'algoritmo precedentemente mostrato) al fine di notificare le applicazioni nella zona critica.

### 3.5.3 Rilevazione della posizione

---

```

double latitudine=0;
double longitudine=0;
String via;
String percorsoCorrente = System.getProperty("user.dir");

Workbook wb= Workbook.getWorkbook(new File(percorsoCorrente
+ "/vie3.xls"));

Sheet sheet = wb.getSheet(0);
Random random = new Random();
int min = 0;
int max = 543;
int viaCasuale = ((max-min) + 1);
int miavar = random.nextInt(viaCasuale) + min;

Cell cella = sheet.getCell(0,miavar);
via = cella.getContents();

```

---

<sup>1</sup> Un dato è obsoleto se è stato ricevuto da più di tre minuti

```

cella=sheet.getCell(1,miavar);
latitudine=Double.valueOf(cella.getContents());
cella=sheet.getCell(2,miavar);
longitudine=Double.valueOf(cella.getContents());

this.posizione=new Posizione(via,latitudine,longitudine);
return this.posizione;

```

---

L'algoritmo mostrato indica il metodo con cui viene rilevata la posizione dell'applicazione.

In particolare, viene fatto riferimento ad un foglio di calcolo contenente le vie della città di Como prese in considerazione. Dopo aver generato un numero casuale indicante il numero della riga da prelevare, vengono a loro volta estratti il nome della via (nella prima colonna), la latitudine (nella seconda colonna) e la longitudine (nella terza colonna), necessari per creare un nuovo oggetto posizione che viene poi assegnato all'applicazione in esame.

#### 3.5.4 Connessione con RMI e messaggio di errore

```

Registry registry = null;
try {

    registry = LocateRegistry.createRegistry(12345);
    registry.rebind("gestApp", GestoreApplicazioni.
        getInstance());

} catch (Exception e) {
    JOptionPane.showMessageDialog(null, "Il sistema e'
        gia' in uso");
}

```

---

La porzione di codice mostrata rappresenta il caricamento dell'interfaccia server del gestore applicazioni nel registry RMI. Il motivo principale per il quale viene mostrato è però un altro: come si può notare nella `catch`, in caso di eccezione, viene mostrato un pannello di errore che indica che il sistema è già in uso. Ciò è importante in quanto impedisce di poter aprire due o più volte il sistema centrale, che è unico.

### 3.5.5 Calcolo dell'intervallo di aggiornamento della centralina stradale

---

```

float temp=((float)numeroVeicoli)/((float)this.
    intervalloDiTempo);

if (temp > 0.0) {
    this.intervalloDiTempo=(int) (this.intervalloDiTempo
        *this.rapporto/temp);

    if (this.intervalloDiTempo < this.intervalloMinimo)
    {
        this.intervalloDiTempo = this.
            intervalloMinimo;
    }

    this.rapporto=temp;
}
else {
    this.intervalloDiTempo=this.intervalloDiTempo*2;
}

```

---

L'algoritmo, come da specifica, varia l'intervallo di tempo di segnalazione del traffico da parte della centralina in base alla quantità di veicoli rilevati durante l'intervallo appena scaduto. In particolare, se vengono rilevati veicoli, temp sarà maggiore di zero, di conseguenza il nuovo intervallo di tempo dipenderà dall'intervallo di tempo precedente (impostato dall'interfaccia della centraline), dal rapporto (inizialmente a zero) e da temp stesso.

Più nello specifico, rapporto indica il numero di veicoli al secondo e, in particolare, nella formula usata per aggiornare l'intervallo di tempo della centralina esso indica il valore relativo all'intervallo precedente. La variabile temp ha lo stesso significato di rapporto, tuttavia indica il numero di veicoli al secondo relativo all'intervallo appena scaduto. In questo modo, se passano più vetture, l'intervallo diminuisce, così che vengano inviate più spesso le notifiche relative alla situazione nel tratto di strada considerato, mentre se ne passano meno l'intervallo aumenta.

L'intervallo di tempo ha un valore minimo, sotto il quale non è possibile scendere. Se non vengono rilevati veicoli, invece, viene raddoppiato l'intervallo di tempo considerato.

All'aumentare del numero di veicoli per unità di tempo rilevati, quindi, diminuisce l'intervallo di tempo della rilevazione successiva.

### 3.5.6 Variazione di velocità rilevata dalla centralina stradale

---

```

Random random = new Random();
int min;
int max;
if (this.velocita>15) {
    min = this.velocita-15;
}
else {
    min =0;
}
if(this.velocita<95) {
    max = this.velocita+15;
}
else {
    max=110;
}
int intorno = ((max-min) + 1);
int vel = random.nextInt(intorno) + min;
this.sommaVelocita=this.sommaVelocita+vel;

```

---

Una centralina stradale deve rilevare la velocità dei veicoli che transitano sul tratto di strada a cui si riferisce.

Dal momento che risulterebbe poco realistico se la centralina rilevasse una velocità molto diversa rispetto a quella della misurazione precedente, la simulazione impone che questa possa variare fino ad un massimo di 15 km/h (in più o in meno) da una rilevazione a quella successiva.

Nel caso in cui la velocità dovesse essere inferiore a 15 km/h, il valore minimo consentito per la rilevazione successiva sarà pari a 0 km/h. Invece se la velocità media dovesse essere superiore ai 95 km/h si utilizza una velocità massima pari a 110 km/h (non essendoci autostrade nel progetto realizzato).

Il valore minimo, il valore massimo e la differenza tra i valori minimo e massimo vengono indicati dalle variabili `min`, `max` e `intorno`.

Una volta generata la velocità casuale, questa viene aggiunta a `somaVelocita`, che viene utilizzata per il calcolo della velocità media (infatti viene divisa per il numero di automobili rilevate).

### 3.5.7 Calcolo della nuova posizione della centralina automobilistica

---

```

this.listaPosizioni.clear();
if (raggio!=0) {
    double R = 6378.137;
    double lat1=this.posizione.getLatitudine();
    double lat2;

```

---

```

double lon1=this.posizione.getLongitudine();
double lon2;
double dLat;
double dLon;
double a;
double c;
double d;
String via;
String percorsoCorrente=System.getProperty("user.dir");
Workbook wb=Workbook.getWorkbook(new File(percorsoCorrente
+ "/vie3.xls"));
Sheet sheet = wb.getSheet(0);
int miavar;
for (miavar=0; miavar<543;miavar++) {
    Cell cella = sheet.getCell(0,miavar);
    via = cella.getContents();
    cella=sheet.getCell(1,miavar);
    lat2=Double.valueOf(cella.getContents());
    cella=sheet.getCell(2,miavar);
    lon2=Double.valueOf(cella.getContents());
    dLat = lat1 * Math.PI/180 - lat2 * Math.PI/180;
    dLon = lon1 * Math.PI/180 - lon2 * Math.PI/180;
    a = Math.sin(dLat/2) * Math.sin(dLat/2) + Math.cos(lat1
        * Math.PI/180) * Math.cos(lat2 * Math.PI/180) * Math
        .sin(dLon/2) * Math.sin(dLon/2);
    c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
    d = R * c;
    if (d<raggio) {
        this.listaPosizioni.add(new Posizione(via.toLowerCase
            () ,lat2,lon2));
    }
}
}

```

---

Questo algoritmo viene utilizzato per il calcolo delle possibili posizioni in cui si può trovare la centralina automobilistica allo scadere dell'intervallo di tempo.

In particolare, questo metodo riceve in ingresso il raggio in km che indica di quanto può spostarsi l'auto dalla posizione in cui si trova. Successivamente viene aperto il foglio di calcolo contenente tutte le vie di Como e tramite un ciclo for si calcola la distanza di ogni via dalla posizione attuale. Se la distanza è minore del raggio, la via viene aggiunta alla lista di posizioni.

Il calcolo della lista di posizioni verrà utilizzato successivamente per scegliere una di queste in modo casuale, nel momento in cui dovrà essere effettivamente calcolata la nuova posizione.

## 3.6 INTERFACCIA GRAFICA

L’interfaccia grafica è stata realizzata per agevolare l’interazione con gli utenti e con gli amministratori. In particolare, per ogni progetto (CentralinaStradale, CentralinaAuto, Sistema Centrale e Applicazione) è stata realizzata un’interfaccia diversa.

L’interfaccia grafica è stata realizzata utilizzando Swing [VV<sub>e</sub>] ed è stata integrata con le API fornite da OpenStreetMap [VV<sub>b</sub>], distribuito con licenza GPL.

### 3.6.1 Login e registrazione



**Figura 27:** Schermata di scelta tra login e registrazione

Nonostante le interfacce grafiche utilizzate per il login e la registrazione di utenti e di amministratori siano uguali, i dati a cui si fa riferimento sono diversi a seconda della tipologia di accesso.

La schermata permette di scegliere il login, nel caso in cui l’utente (o l’amministratore) si sia già registrato durante la sessione in corso<sup>2</sup>, oppure la registrazione, nel caso in cui la registrazione non sia già stata effettuata con il nome utente desiderato.

Nelle schermate successive sarà possibile fare il logout usando l’apposito tasto o, più semplicemente, chiudendo la finestra (per l’utente dell’applicazione mobile, per l’amministratore della mappa).

### 3.6.2 Sistema centrale

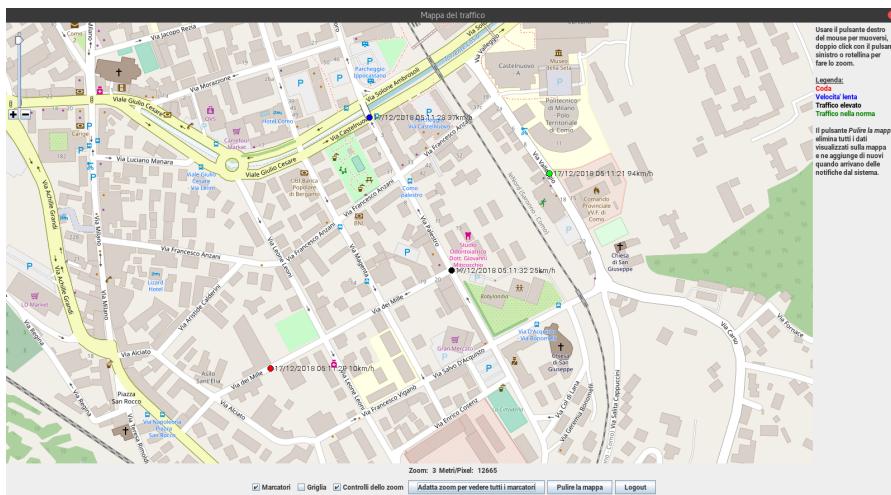
L’accesso al sistema centrale viene permesso in seguito al login o alla registrazione di un nuovo amministratore. La prima schermata visualizzata, quindi, richiede di effettuare una di queste due operazioni.

Se il login o la registrazione vanno a buon fine viene visualizzata l’interfaccia grafica vera e propria del sistema centrale, costituita dalla mappa (i dati provengono da [VV<sub>c</sub>]), da alcuni controlli per spostarsi sulla mappa e da un pulsante che permette di effettuare il logout dal sistema centrale.

Quando la mappa riceve dei dati dagli altri sistemi (collegati grazie a RMI [VV<sub>d</sub>]), li visualizza utilizzando marcatori di colore diverso in base all’evento di traffico (la legenda<sup>3</sup> sulla destra dell’interfaccia

<sup>2</sup> La sessione inizia quando viene aperto il sistema centrale

<sup>3</sup> **Rosso:** coda; **Nero:** traffico elevato; **Blu:** velocità lenta; **Verde:** Traffico nella norma.



**Figura 28:** La mappa con alcuni marcatori posizionati durante una simulazione

grafica spiega il significato dei diversi colori), riportando, per ogni marcitore, informazioni relative alla data e all'ora di rilevamento e alla velocità media calcolata (se disponibile). I marcatori vengono automaticamente posizionati in corrispondenza della posizione della segnalazione.

Nel caso in cui più marcatori si riferiscano alla stessa posizione viene mantenuta l'ultima informazione ricevuta dal database e i marcatori relativi alla stessa posizione già presenti vengono eliminati dalla mappa.

### 3.6.3 Centralina stradale

**Centralina stradale (Via Palestro)**

Via o piazza:	<input type="text" value="via palestro"/>
Tipo di strada:	<input type="text" value="urbana"/>
Intervallo di tempo iniziale [s]:	<input type="text" value="10"/>
Velocita' [km/h]:	<input type="text" value="20"/>
<input type="checkbox"/> Selezionare per impostare una velocita' iniziale casuale	
Ultima velocita' rilevata: 34 km/h	
<input type="button" value="OK"/> <input type="button" value="Imporre una velocita'"/>	

**Figura 29:** Schermata iniziale della centralina stradale

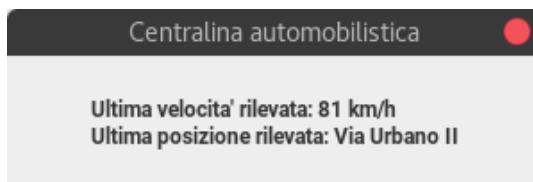
La centralina stradale viene impostata senza effettuare il login e la registrazione, infatti immaginando che il sistema venga effettivamente utilizzato da un utente.

te realizzato è ragionevole pensare che la centralina venga impostata, per esempio da un operatore, nel momento in cui viene installata e che la configurazione rimanga immutata fino ad un suo eventuale riavvio.

L'interfaccia grafica della centralina permette anche di imporre la velocità registrata in modo da testare il funzionamento dei sistemi anche in assenza di dati reali. In alternativa è possibile utilizzare un simulatore di velocità che genera dei valori casuali che oscillano intorno all'ultimo valore registrato.

Una volta effettuata l'impostazione iniziale, la posizione della centralina (che sarà necessariamente fissa) verrà indicata nella barra del titolo della finestra di impostazione. In ogni caso, sarà possibile osservare un marcitore, di colore variabile in funzione dell'evento di traffico registrato, in corrispondenza della posizione di installazione.

#### 3.6.4 Centralina automobilistica



**Figura 30:** La centralina automobilistica

L'interfaccia grafica per la centralina automobilistica non permette di modificare parametri, ma solo di visualizzare la velocità registrata e la posizione. Per questo motivo vengono utilizzate due etichette, modificate nel momento in cui viene registrato un nuovo dato.

Il motivo per cui non viene data la possibilità di modificare i parametri significativi (periodicità e posizione iniziale) è che la periodicità, secondo le specifiche, deve essere fissa, mentre la posizione in una situazione reale non viene impostata dall'utente.

Anche in questo caso vengono usati marcatori di colore diverso a seconda della velocità e, di conseguenza, dell'evento di traffico che si suppone si stia verificando.

#### 3.6.5 Applicazione mobile

L'applicazione mobile consente all'utente, che deve registrarsi o effettuare il login, di segnalare una coda nella posizione in cui si trova (questa apparirà sotto forma di marcitore sulla mappa, se in esecuzione) e di visualizzare le notifiche relative ad un'area circolare<sup>4</sup> con centro nella posizione rilevata dall'applicazione e raggio di 5 km.

---

<sup>4</sup> A cui è possibile riferirsi come **area critica**.



Figura 31: L'applicazione mobile con alcune notifiche.

Le notifiche più recenti appariranno nella parte superiore dell'area di notifica e saranno di colori diversi in base alla tipologia, seguendo le stesse convenzioni dei marcatori visualizzati sulla mappa. Non vengono visualizzate le segnalazioni di assenza di traffico perché non rilevanti.

Al fine di testare il sistema complessivo, è possibile richiedere che l'applicazione rimanga nella stessa posizione tra una rilevazione e l'altra.

### 3.6.6 Utilizzo di Swing

Le interfacce grafiche sono state realizzate in modo simile. In particolare, si definisce **frame** il contenitore di tutti gli elementi visualizzati nell'interfaccia grafica.

All'interno di ogni frame possono essere posizionati più pannelli (**JPanel**), che vengono utilizzati come dei contenitori di altri elementi. Ad esempio, la GUI della centralina stradale ha tre pannelli, che contengono i dati di impostazione della centralina, l'impostazione (e visualizzazione) della velocità rilevata (per controllare il funzionamento del sistema complessivo) e i bottoni per confermare le scelte effettuate sulla schermata.

Gli elementi che possono essere utilizzati sono vari. Nel caso di questo progetto sono risultati molto utili gli **spinner** (per selezionare numeri in un dato intervallo), le **label**, i **textfield** (per brevi testi), **textpane** (per testi più lunghi), le **combobox** (per scegliere tra più opzioni, sotto forma di stringa), i **button** e gli **optionpane** (per visualizzare messaggi di errore e informazioni).

## 3.7 FORMATO DELLE NOTIFICHE

Le notifiche sono fondamentali per garantire la comunicazione tra i diversi programmi. Per questa ragione le notifiche devono avere un formato standard che sia comprensibile da tutti i programmi.

In particolare, ogni notifica è caratterizzata da un tipo, che è una stringa. Il formato di questo attributo è il seguente: **M**, **S** oppure **V<sup>5</sup>** + **velocità** + **carattere** '' + **Tipo di evento**.

Nel caso della segnalazione della coda da parte delle applicazioni mobili, la velocità non viene rilevata, quindi può essere inserita una qualsiasi stringa prima del carattere '' e questa verrà ignorata.

Un esempio di notifica può essere 'S10 Coda' e significa che una centralina stradale invia una notifica di traffico elevato registrando una velocità di 10 km/h.

Le notifiche verranno visualizzate sull'applicazione mobile se sono riferite a segnalazioni provenienti da altre applicazioni o centraline nel raggio di 5 km.

Il formato di queste ultime notifiche, però, è più comprensibile per un utente umano. La notifica 'S10 Coda' proveniente da via Valleggio il giorno 17/12/2018 alle ore 11:11, per esempio, verrà visualizzata come 17/12/2018, 11:11 | Coda in via Valleggio ad una velocità media di 10 km/h e sarà di colore rosso (vedere anche figura 31). Le notifiche dell'applicazione mobile sono realizzate con un codice HTML, grazie alla libreria **HTMLDocument** [VVA].

---

<sup>5</sup> M nel caso in cui il dato si origini dall'applicazione mobile, S nel caso in cui provenga da una centralina stradale, V per la centralina automobilistica

## 3.8 PROTOCOLLO DI COMUNICAZIONE

Il sistema centrale, la centralina e l'applicazione vengono eseguiti su macchine differenti e quindi è necessario configurare una connessione tra questi. Il protocollo di comunicazione che è stato utilizzato è RMI. Sono state programmate tre differenti connessioni RMI.

Le centraline sono state sviluppate come client del GestoreCentraline che lavora da server. Il GestoreCentraline implementa l'interfaccia `IGestoreCentraline` attraverso la quale riceve i dati di traffico inviati dalle centraline.

Le applicazioni e il GestoreApplicazioni si comportano sia da server sia da client. Le applicazioni sono dei server quando devono ricevere delle notifiche dal GestoreApplicazioni, e a questo proposito implementano l'interfaccia `IApplicazione`. Il GestoreApplicazioni invece implementa l'interfaccia `IGestoreApplicazioni` per ricevere notifiche da parte delle applicazioni.

Le porte che sono state utilizzate sono:

- 12344 per la connessione tra centraline e GestoreCentraline;
- 12345 per la connessione tra applicazioni e GestoreApplicazioni in cui è quest'ultimo a fare da server;
- da 12346 in poi, per la connessione in cui le applicazioni fanno da server e il numero di porta è ottenuto sommando a 12346 il numero di identificativo dell'applicazione.

## 3.9 CASI DI TEST JUNIT

### *Aggiunta e rimozione di utenti - AggiuntaUtentiTest*

Prima del test vengono aggiunti tre utenti tramite il GestoreUtenti gest, quindi questi vengono inseriti all'interno della lista `listaUtenti`.

Successivamente, durante il test vero e proprio, gli utenti vengono rimossi da `listaUtenti` grazie alla funzione `rimuoviUtente()` di gest. `ListaUtenti` alla fine del test deve risultare vuota.

Il test fallirebbe nel caso in cui si dovesse chiamare `rimuoviUtente()` senza passare tutti gli username degli utenti aggiunti.

### *Calcolo dell'intervallo di aggiornamento delle centraline - CalcoloIntervalloTest*

Il test serve per verificare che, in assenza di veicoli registrati da una centralina, l'intervallo di rilevazione raddoppi.

Viene creata una centralina stradale, la cui posizione è ininfluente ai fini del test, ha un intervallo iniziale di 10 s ed è posizionata su una strada urbana (anche questo dettaglio è ininfluente).

Quando viene chiamata la funzione `calcolaIntervallo()` si passa come argomento `o`, che è il numero di veicoli rilevati.

In particolare, il test consiste nel verificare che il nuovo intervallo (`centralina.getIntervallo()`) sia pari a 20, ovvero il doppio dell'intervallo iniziale.

*Verificare che la mappa, inizialmente vuota, venga mostrata - FunzionamentoSistemaCentraleTest*

Il test viene utilizzato per verificare che la mappa venga effettivamente visualizzata nel momento in cui viene chiamata la funzione `visualizzazioneMappaBase()`.

Si tratta di un test molto compatto dal momento che la funzione `visualizzazioneMappaBase()` ritorna `mappa` e che la funzione `isVisible()` è contenuta all'interno delle API di `JMapView`. Allo stesso tempo è fondamentale che abbia successo per garantire che l'amministratore visualizzi le informazioni sul traffico.

*Riconoscimento di un amministratore - GestoreAmministratoriTest*

Il test serve per verificare che, una volta aggiunto un amministratore alla lista degli amministratori, questo venga riconosciuto nel momento in cui viene cercato.

La funzione centrale per il riconoscimento è `riconosciAmministratore()`, che richiede il nome utente e la password dell'amministratore e restituisce `true` o `false`.

*Funzionamento di creaDatoGenerico - GestoreDatabaseTest*

`DatoTrafico` è una specializzazione di `DatoGenerico`, di conseguenza alcune delle informazioni ottenute a partire da `DatoTrafico` potranno essere applicabili a `DatoGenerico` e le informazioni ottenute a partire da `DatoGenerico` saranno sicuramente applicabili a `DatoTrafico`.

Il test verifica che il `DatoGenerico` costruito a partire da alcune delle informazioni di `DatoTrafico` ed elaborato dal `GestoreDatabase` sia uguale al `DatoGenerico` che dovrebbe contenere gli stessi dati.

Il test, quindi, verifica indirettamente che `DatoTrafico` sia una specializzazione di `DatoGenerico`, anche se il suo scopo è la verifica del funzionamento corretto di `creaDatoGenerico()` in `GestoreDatabase`.

*Test sulla rilevazione della posizione - PosizioneApplicazioneTest*

Il test è suddiviso in due test e vuole verificare il corretto funzionamento dell'aggiornamento della posizione di un'applicazione.

Il primo test crea un'applicazione e aggiorna la sua posizione con `getPosizione()`. Successivamente viene imposto all'applicazione di

fissare la sua posizione, e quindi si prova a aggiornare la posizione. L'asserzione verifica che la posizione non sia cambiata.

Il secondo test invece svolge gli stessi passaggi del test precedente. A differenza del primo però, dopo aver aggiornato la posizione, toglie il vincolo che fissa la posizione e chiama nuovamente `getPosizione()`. In questo test si verifica quindi che la posizione è cambiata rispetto alla posizione iniziale.

#### *Test sulla gestione delle notifiche ricevute dall'applicazione - SegnalaApplicazioneTest*

Anche questo test è diviso in due parti e vuole verificare la corretta gestione della lista delle notifiche che riceve l'applicazione.

Nel primo test l'applicazione riceve tre notifiche e le aggiunge alla lista notifiche. Per verificare che ciò venga fatto, si fa un controllo sul numero di notifiche all'interno dell'`ArrayList`.

Il secondo test aggiunge al test precedente l'istruzione che svuota completamente la lista notifiche. In questo caso la dimensione dell'`ArrayList` deve tornare a zero perché tutte le notifiche vengono eliminate.

#### *Test sulla ricezione di notifiche applicazione da parte del sistema centrale - SegnalaDatabaseTest*

Il test ancora una volta è diviso in due. In questo caso si vuole verificare il corretto funzionamento della gestione dei dati che riceve in ingresso il sistema centrale.

Nel primo test vengono create due notifiche applicazione in posizioni differenti e si vuole verificare che il `GestoreDatabase` le tramuti correttamente in `DatoGenerico` e le aggiunga alla tabella di traffico.

Nel secondo test si prova a inviare una notifica nella stessa posizione di una notifica precedente e si verifica che il sistema confronti nel modo corretto le posizioni e elimini la notifica meno recente per aggiungere quella nuova.

#### *Test sulla ricezione di dati di traffico - SegnalaDatabaseSTest*

Questo test è molto simile al precedente, l'unica differenza è che al posto delle notifiche applicazione, il test viene effettuato su dati di traffico in arrivo dalle centraline.

#### *Verifica dell'accesso e dell'esistenza di un utente usando la classe GestoreApplicazioni - RegistrazioneUtenteTest*

Nel momento in cui un utente tenta di registrarsi, è necessario che l'username scelto sia diverso rispetto a quello di tutti gli altri utenti. Inoltre, nel momento in cui un utente vuole fare il login, in seguito ad

una precedente registrazione, bisogna controllare che l'utente esista e, in questo caso, che la password inserita sia corretta.

Per questa ragione sono stati realizzati due metodi `verificaAccesso` ed entrambi vengono testati. Viene testato, quindi, sia il riconoscimento di un utente effettivamente esistente, sia il mancato riconoscimento di un utente non ancora registrato.

Entrambe le versioni (riconoscimento di utenti esistenti e blocco di utenti non ancora registrati) sono molto importanti, infatti verificano che non si possa accedere al sistema se non autorizzati.

#### *Verifica dell'accesso e dell'esistenza di un utente usando la classe GestoreUtenti - RiconoscimentoUtenteTest*

Questo test è molto simile al precedente, ma sfrutta la classe `GestoreUtenti` piuttosto di `GestoreApplicazioni`.

Lo scopo del test è quindi quello di verificare che il sistema funzioni anche chiamando `riconosciUtente` al posto di `verificaAccesso`.

#### *Test sulla creazione di DatoTraffico - CreaDatoTrafficoTest*

Il test in analisi è suddiviso in tre verifiche distinte che controllano la correttezza della creazione di dati di traffico da parte delle centraline stradali.

Nella sezione `BeforeEach` vengono create tre centraline, ognuna con un tipo di strada differente. In tutte e tre le centraline viene settata la velocità media pari a 45 km/h, e viene creato un dato di traffico per ciascuna.

I tre test verificano che il dato creato abbia il tipo di evento di traffico corretto, che varia a seconda della velocità media per ogni tipologia di strada.

#### *Test sulla creazione di StatoVeicolo - CreaStatoVeicoloTest*

Questo test è molto simile al precedente, ma riguarda la creazione di `StatoTraffico` da parte di `CentralinaAuto`.

Nella sezione `BeforeEach` vengono create tre centraline con posizioni uguali e velocità differenti e poi vengono creati tre `StatoVeicolo`, uno per ogni centralina.

I tre test verificano che il dato creato abbia il tipo di evento di traffico corretto, che varia a seconda della velocità media.

#### *Verifica del funzionamento di calcolaRaggio - CalcolaRaggioTest*

Il test in questione verifica il corretto funzionamento del metodo `calcolaRaggio` della centralina automobilistica. Dato che questo metodo presenta un if, un else if e un else, il test è suddiviso in tre test.

Nella sezione BeforeEach vengono create tre centraline auto con posizioni e velocità differenti e vengono settate le ultime posizioni rilevate dalle centraline.

Il primo test verifica che il raggio venga calcolato essendoci posizioni differenti e velocità diversa da zero.

Il secondo test verifica che il raggio venga posto a zero quando la velocità rilevata è pari a zero.

L'ultimo test verifica che il raggio venga raddoppiato nel momento in cui la posizione rilevata non è cambiata rispetto all'ultima rilevata.

### 3.10 SIMULAZIONE

Per simulare al meglio il progetto è consigliato seguire i seguenti passi:

1. Nel progetto "SistemaCentrale" aprire il package "prog" e eseguire "FunzionamentoSistemaCentrale".
2. Completare la registrazione dell'amministratore (username e password non possono essere vuoti altrimenti viene visualizzato un messaggio di errore). Se la registrazione andrà a buon fine verrà aperta la mappa (verificare di essere connessi a una rete internet).
3. Può essere eseguito solamente un "FunzionamentoSistemaCentrale" alla volta, se un altro venisse eseguito verrebbe mostrato un messaggio di errore e terminerebbe.
4. Nel progetto "CentralinaStradale" aprire il package "prog" e eseguire "FunzionamentoCentralinaS". Se "FunzionamentoCentralinaS" venisse eseguito prima di "FunzionamentoSistemaCentrale" verrebbe mostrato un messaggio di errore e terminerebbe.
5. Scrivere il nome di una via di Como (anteponendo "via"), se non dovesse essere presente nella lista delle vie possibili verrà chiesto di inserirne un'altra. Per essere sicuri di inserire vie esistenti, **aprire il file vie3.xls presente in tutti e tre i progetti e selezionare una tra le vie presenti** (in alternativa è possibile consultare l'appendice B in fondo al documento).
6. Compilare gli altri campi e nel caso si volesse scegliere una velocità iniziale casuale selezionare la casella apposita.
7. Nel progetto "CentralinaAuto" aprire il package "prog" e eseguire "FunzionamentoCentralinaA". Se "FunzionamentoCentralinaA" venisse eseguito prima di "FunzionamentoSistemaCentrale" verrebbe mostrato un messaggio di errore e terminerebbe.

8. Nel progetto "Applicazione" aprire il package "prog" e eseguire "FunzionamentoApplicazione". Se "FunzionamentoApplicazione" venisse eseguito prima di "FunzionamentoSistemaCentrale" verrebbe mostrato un messaggio di errore e terminerebbe. Non è invece obbligatorio eseguire prima la centralina dell'applicazione.
9. Completare la registrazione dell'applicazione (username e password non possono essere vuoti altrimenti viene visualizzato un messaggio di errore). Se la registrazione andrà a buon fine verrà aperta l'interfaccia grafica dell'applicazione.
10. Possono essere eseguiti contemporaneamente quante applicazioni e centraline si voglia.

A questo punto si può scegliere di testare tutti i comandi a disposizione; alcuni esempi sono:

- Effettuare il login e il logout dell'applicazione.
- Segnalare la coda con l'apposito pulsante dell'applicazione.
- Pulire la lista delle notifiche.
- Fissare la posizione di qualsiasi applicazione.
- Impostare una velocità media alla centralina stradale per impostare un evento di traffico.
- Pulire manualmente la mappa o attendere che venga pulita in automatico (ogni punto rimane sulla mappa per tre minuti, poi viene eliminato).
- Adattare lo zoom in modo che tutti i punti vengano visualizzati al meglio sulla mappa premendo l'apposito pulsante sulla mappa.

# 4

## MODIFICHE

Durante la realizzazione del progetto si sono rese necessarie alcune modifiche rispetto alle supposizioni fatte nelle fasi precedenti. Lo scopo di questo capitolo è quello di raccogliere i cambiamenti più significativi.

La documentazione è comunque aggiornata tenendo conto di tutti i cambiamenti e propone la versione finale dell'elaborato.

### 4.1 PROTOCOLLO DI COMUNICAZIONE

Le **connessioni** tra i diversi programmi sono state realizzate utilizzando RMI, e non implementando una connessione TCP/IP come supposto durante la fase di analisi dei requisiti. Il motivo di questa scelta è che RMI è supportato nativamente in Java e permette di connettere le parti di un sistema distribuito utilizzando una programmazione ad alto livello e ad oggetti, coerentemente con il resto del progetto.

### 4.2 AGGIUNTA DI NUOVI METODI, ATTRIBUTI E CLASSI

Rispetto al **class diagram**, all'interno del quale erano stati riportati soltanto i metodi e gli attributi più significativi per il funzionamento del sistema, sono stati aggiunti alcuni getter e setter e sono state apportate alcune modifiche ai costruttori, in modo da garantire il funzionamento del sistema ed una maggiore leggibilità del codice anche in presenza di RMI.

Inoltre sono state aggiunte alcune classi necessarie per il corretto funzionamento del sistema con RMI. Di conseguenza è stato modificato anche l'**object diagram**.

### 4.3 INTERFACCIA GRAFICA

L'**interfaccia grafica**, non considerata durante la realizzazione del class diagram, ha richiesto l'implementazione di alcune classi aggiuntive, tra cui `LoginDialog`, `RegistrazioneDlg` e `MappaGrafica`. Inoltre è stato necessario implementare alcuni metodi per poter utilizzare le API di `JMapView`.

Avendo scelto di realizzare una mappa, e non ad esempio un diagramma, per permettere all'amministratore di consultare i dati, sono stati rimossi i riferimenti al diagramma nei diagrammi UML che ne facevano uso.

#### 4.4 SCELTA DELLA MODALITÀ DI RAPPRESENTAZIONE DEI DATI

La scelta del metodo di rappresentazione dei dati per l'amministratore è ricaduta sulla mappa, che, rispetto al diagramma, rende più semplice il controllo dello stato del traffico complessivo.

Per quanto riguarda gli altri sistemi, i dati vengono rappresentati tutti ricorrendo a interfacce grafiche, più facili da consultare rispetto ad un terminale.

#### 4.5 STIMA DEI DATI TRASMESSI

La stima dei dati trasmessi è stata rivista, in modo da tenere in considerazione margini di sicurezza più ampi. In questo modo il sistema potrà funzionare anche in condizioni estreme (ad esempio quando tutte le macchine di cittadini del Comune di Como stanno circolando contemporaneamente, continuando ad inviare e ricevere segnali grazie alle applicazioni mobili).

#### 4.6 PROPOSTE DI SOLUZIONE AI POTENZIALI PROBLEMI

Mentre nella versione originale di questa documentazione la risoluzione di alcuni problemi non era stata affrontata in modo esplicito, ora è presente una tabella che, per ogni potenziale rischio o ostacolo propone una soluzione.

#### 4.7 INTRODUZIONE CENTRALINA AUTOMOBILISTICA

In un primo momento si era deciso di implementare solamente la centralina stradale. Successivamente si è deciso di implementare anche la centralina automobilistica per una maggiore completezza dell'analisi e della gestione del traffico cittadino.

# A | APPENDICE

## A.1 CALCOLO DEI COSTI

Siano SC il sistema centrale, CS una centralina stradale, CA una centralina automobilistica, EL la corrente elettrica e t il tempo in ore. Si indichi con C un costo, con P una potenza, con N un numero e con M la manutenzione. Il costo complessivo può essere espresso come somma di una parte fissa (il costo dei componenti) e di una parte variabile (che dipende dal consumo di corrente e dal costo periodico di manutenzione). Il costo complessivo, di conseguenza, è dato dalla formula seguente:

$$C(t) = C_{CS}N_{CS} + C_{CA}N_{CA} + C_{SC} + C_{EL}t(P_{CS}N_{CS} + P_{SC}) + \frac{t}{8760}(M_{CS} + M_{SC})$$

Supponendo che:

- il numero di centraline stradali ( $N_{CS}$ ) sia 100;
- il numero di centraline delle automobili ( $N_{CA}$ ) sia 8000 (pari al 15% delle automobili presenti nel Comune di Como);
- il costo della corrente elettrica ( $C_{EL}$ ) sia pari a 0.20 €/kWh [ARE];
- il costo del sistema centrale ( $C_{SC}$ ) sia di 20000 €;
- il costo di una centralina stradale ( $C_{CS}$ ) sia di 200 €;
- il costo di una centralina auto ( $C_{CA}$ ) sia di 50 €;
- la potenza di una centralina stradale ( $P_{CS}$ ) sia di 3 W (dato stimato sulla base di [Pat15]);
- la potenza del sistema centrale ( $P_{SC}$ ) sia di 1 kW [Pat15];
- il costo di manutenzione di una centralina stradale ( $M_{CS}$ ) sia di 100 €/anno;
- il costo di manutenzione del sistema centrale ( $M_{SC}$ ) sia di 1000 €/anno.

si ottiene quindi un costo complessivo pari a:

$$C(t) = 200 * 100 + 50 * 8000 + 20000 + 0.20 * t * (0.003 * 100 + 1)$$

$$+ \frac{t}{8760}(100 * 100 + 1000) = 440000 + 1.52t$$

dove il primo numero è il costo di realizzazione mentre il secondo è il costo dell'elettricità e della manutenzione in funzione del tempo espresso in ore. Annualmente il sistema necessita di una spesa pari a circa 14000 €(ottenuto ponendo t=8760h).

## A.2 CALCOLO DEI DATI TRASMESSI

Si può quindi stimare la quantità di dati trasmessi nel caso peggiore. Le centraline automobilistiche devono inviare al sistema centrale velocità e posizione. 9 bit risultano sufficienti per la trasmissione del dato di velocità. Per esprimere la posizione in termini di latitudine e longitudine, supponendo che ogni valore abbia una precisione di 10 cifre decimali, sono necessari  $\log_2(10^{20}) = 67$  bit.

Complessivamente, di conseguenza, devono essere trasmessi 76 bit. Quando i dati devono essere trasmessi, è necessario aggiungere un certo overhead, variabile in base al protocollo utilizzato. Supponendo che avvenga un invio per ogni secondo e considerando che è buona norma considerare anche un margine di sicurezza, la velocità di trasmissione richiesta è pari a 256 bit/s.

Per quanto riguarda le centraline stradali, queste devono inviare la velocità (9 bit) ed, eventualmente, il numero identificativo della centralina (10 bit). Si può supporre che venga utilizzato un protocollo analogo a quello delle centraline automobilistiche e, ancora una volta, un ampio margine di sicurezza, per un volume di dati complessivo pari a 128 bit/s.

Supponendo di installare le centraline su 8000 auto, nel caso peggiore trasmetteranno  $256 * 8000 = 2.048 * 10^6$  bit/s  $\sim 10^7$  bit/s. Le centraline stradali, assumendo che siano posizionate tutte in strade con un alto flusso di traffico, trasmettono il proprio identificativo e la velocità di ogni macchina che passa. Si può ipotizzare che tutte le macchine viaggino a 180 km/h (50 m/s), che ogni macchina sia distanziata di 5 metri dalla precedente e sia lunga 4 metri e che invii i dati ogni secondo, di conseguenza passano 6 macchine al secondo, quindi si ottiene  $6 * 128 * 100 = 7.68 * 10^4$  bit/s  $\sim 10^5$  bit/s.

Infine è possibile stimare la quantità di dati inviati e ricevuti dall'applicazione mobile. Si supponga che a Como circolino contemporaneamente 60000 auto, circa pari alle macchine presenti sul territorio [IST], ognuna delle quali riceve i dati dal sistema centrale (ad esempio una notifica da 16 caratteri, pari a 128 bit ogni due secondi), e invii altrettanti bit nello stesso tempo. Il traffico di bit relativi all'app mobile risulta essere  $60000 * 128 * \frac{2}{2} = 7.68 * 10^6$  bit/s  $\sim 10^7$  bit/s.

Il traffico di dati massimo, quindi, sarebbe nell'ordine di grandezza di  $10^7$  bit/s, un valore più basso di un fattore 100 rispetto alle velocità di connessione raggiungibili con le tecnologie attualmente disponibili per la banda ultralarga [Svi].

I valori ottenuti sono nettamente superiori a quelli che si ottrebbero mediamente, infatti la circolazione contemporanea di tutte le automobili presenti in città è un evento improbabile e il coefficiente di sicurezza utilizzato è volutamente molto elevato. Il motivo di questa scelta è che le stime devono valere anche in caso di imprevisti o di sviluppi futuri del sistema.

# B

## LISTA DI INDIRIZZI POSSIBILI

Quello che segue è l'elenco degli indirizzi che è possibile inserire nel campo "Via o piazza" della centralina stradale, presenti anche nel foglio di calcolo. Il campo non è case sensitive ed è necessario inserire il nome completo della via (ad es. via Valleggio, e non Valleggio).

Via Abba	Via Fiammenghino	Via Piave
Via Acquanera	Via Fiume	Via Picasso
Via Adamello	Via Florio	Via Picchi
Via Agliati	Via Fogazzaro	Via della Pila
Via Airoldi	Via Fontana	Piazzetta Pinchetti
Via Alberina	Via Fontanella	Via Pio XI
Via Albertolli	Via Fornace	Via Plinio
Via Albricci	Via Foscolo	Via Pola
Via Alciato	Via Fossati	Via Polano
Via Alebbio	Via Fra Silvestro da Siena	Via Ponte Nuovo
Via Ambrosoli	Via Franscini	Piazza del Popolo
Piazza Amendola	Via Frigerio	Via Porro
Via Amoretti	Via Frisia	Via Porta
Via Annunciata	Via Friuli	Via Prato Pasquee
Via Antelami	Via Fulda	Via Prestino
Via Appiani	Via Fumagalli	Via Pretorio
Via Arcioni	Via Gaggi	Via Primo Maggio
Via Artaria	Via Gagini	Via Prinetti
Via Asiago	Via Galilei	Via Priva
Via Auguadri	Via Galli	Via Privata Perlasca
Via Badone	Via Gallio	Via Prudenziiana
Via Bainsizza	Via Galvani	Viale Puecher
Via Balbiani	Via Garibaldi	Via Quadrio
Via Balestra	Via Campo Garibaldi	Via Quaglio
Via Ballarini	Via Garovaglio	Via Quarcino
Via Baracca	Via Gasparotto	Via Quasimodo
Via Baraggia	Via Gattoni	Via Raffaello
Via Baragiola	Viale Geno	Via Raimondi
Via Baravalle	Piazzale Gerbetto	Via Rampoldi
Via Barberini	Via Ghislanzoni	Via Raschi
Via Barelli	Via Giacosa	Via Recchi
Via Bari	Via Gioa	Via Regazzoni
Via Barsanti	Via Giorgione	Via Regina Teodolinda
Via Barzaghi	Piazzale Giotto	Via Repubblica Romana
Via Baserga	Via Giovio	Via Rezzonico
Via al Bassone	Via Giudici	Via Rho
Via della Bastiglia	Via Giulio Cesare	Via Ricci
Viale Battisti	Via Giustizia e Liberta	Via Rienti
Via Bazzoni	Via Gobbi	Via Rienza
Via Beccaria	Via Gobetti	Via Righi
Via Bellini	Via Golasecca	Viale Rimembranza
Via Bellinzona	Via Gorio	Via Rimoldi

Via Belvedere	Via Gorizia	Via Ripamonti
Via Benzi	Via Gramsci	Via Rismondo
Via Bernasconi	Via Grandi	Via Risorgimento
Via Bernina	Via Grassi	Via Ristori
Via Bertacchi	Via Grilloni	Via Riviera
Via Bertinelli	Via Guaita	Via Roasio
Via Bertolone	Via Guanella	Via al Roccolo
Via Bianchi Giovini	Via Imbonati	Via Rodari
Via Bignanico	Via Indipendenza	Piazza Roma
Via Binda	Viale Innocenzo XI	Via Roncate
Via Bixio	Via Interna	Via Ronchetto
Via Boccioni	Via Isonzo	Via al Ronco
Via Boldoni	Via Italia Libera	Via Rosales
Via Bonanomi	Piazza IV Novembre	Via Rosati
Via Bonardi	Via Juvara	Via Roscio
Via Bonola	Via Kolbe	Viale F.lli Rosselli
Via Bonomelli	Via Lambertenghi	Via Rossini
Via Bontempelli	Via Landriani	Via Rota
Via Borgo Vico	Via del Lavoro	Via Rovelli
Via Borsieri	Via Lazzago	Via Rubini
Via Bosatta	Viale Lecco	Via Rusconi
Via Boselli	Strada Statale per Lecco	Via Rutschi
Via Bossi	Via Lega Insurrezionale	Via Sabotino
Via Botticelli	Largo Leopardi	Via Sacchi
Via Brambilla	Via Liberta	Via Sacco
Via Brenna	Via Linati	Via Sagnino
Via Brennero	Via Lissi	Via Sala
Via Brenta	Via Longhena	Via Salvadonica
Via Breva	Via Longoni	Piazza San Fedele
Via Briantea	Via di Lora	Via San Felice
Via Brogeda	Via Konrad Lorenz	Viale San Fermo della Battaglia
Via per Bronno	Via Lucini	Via San Francesco d'Assisi
Via Bronzetti	Via Luini	Via San Giacomo
Via per Brunate	Via Macchi	Via San Giovanni da Meda
Via B. Buozzi	Via Maderno	Piazzale San Gottardo
Via Buschi	Via Madruzza	Via San Martino
Piazza Cacciatori delle Alpi	Via Maestri Campionesi	Via San Michele del Carso
Via Cadorna	Via Maestri Comacini	Piazzale San Rocchetto
Via Caduti Albatesi	Via Magistretti	Piazzale San Rocco
Via Cairoli	Via Majnoni	Via San Zenone
Via Calderini	Via Maloja	Via Sant'Abbondio
Piazza Camerlata	Via Malvito	Via Sant'Antonino
Via Camozzi	Via Manara	Via Sant'Arialdo da Cucciago
Via Campagna	Via Mantegna	Via Sant'Elia
Via Campari	Via Marchesi	Via Sant'Eutichio
Via Campora	Viale Marconi	Via Santa Caterina
Via Canonica	Via Mariani	Via Santa Chiara
Via Canova	Via Martinelli	Via Santa Maria in Cristino
Via Cantoni	Via Masaccio	Piazzale Santa Teresa
Via Cantoniga	Via Mascherpa	Via Saporiti
Via Cantore	Viale Masia	Via N. Sauro
Via Cantu	Via Massardi	Via Scalabrinii
Via Canturina	Piazza Matteotti	Via Scalini
Via Capiaghi	Viale G. Matteotti	Via Scarabota
Via Cappelletti	Piazza Mazzini	Via Schiavio
Via Caprani	Piazza Medaglie d'Oro	Via Scolari
Via Caracciolo	Via Mentana	Via Sebenico

Largo Caradocco	Via Meroni	Via Sebregondi
Via Carcano	Via Merzario	Via Selva Fiorita
Via Cardano	Via Meucci	Via Silva
Via Cardina	Via Miani	Viale Sinigaglia
Via Carducci	Via Michelangelo	Via Sirtori
Via Carloni	Via Milano	Via Soave
Via Carluccio	Via dei Mille	Via Solari
Via Caronti	Via Mincio	Piazzale Somaini
Via Carpani	Via Mirabello	Via Somigliana
Via Carso	Via Mocchetti	Via Spalato
Via Casartelli	Via Mognano	Via Spallanzani
Via Casati	Via Mola	Via Spallino
Via Cascina Viola	Via Monte Bianco	Via Spartaco
Via Castel Carnasino	Via Monte Caprino	Via Spazzi
Via Castellini	Via Monte Croce	Largo Spluga
Via Castelnuovo	Via Monte Goi	Viale Sport
Via Catenazzi	Via Montegrappa	Via Sportivi Comaschi
Via Catone	Via Montello	Via Stampa
Viale Cattaneo	Via Montelungo	Via Stazzi
Via Catullo	Via Montenero	Via Stoppani
Via alla Cava	Via Monterotondo	Via Strabone
Viale Cavallotti	Via Monticelli	Via Tagliamento
Piazza Cavour	Via Moraglia	Via Tatti
Via Cecilio	Via Morazzone	Via Tentorio
Largo Ceresio	Via Aldo Moro	Via Terlizza
Via Ceresola	Via Muggio	Via Tettamanti
Via Cermenati	Via Mugiasca	Via Tibaldi
Via per Cernobbio	Via dei Mulini	Via Ticozzi
Via Ceruti	Via Muralto	Via Timavo
Via Cetti	Via Musa	Via Tintoretto
Via Ciapparelli	Via Museo Giovio	Via Tiziano
Via Cigalini	Via Muttoni	Via Tobagi
Via Cinque Giornate	Via Napoleona	Via Tofane
Via Clemente XIII	Via Natta	Viale Tokamachi
Via Clerici	Via Navedano	Via Torno
Via Col di Lana	Via Negretti	Via Torriani
Via Colli	Via Nicolodi	Via Trau
Via Collina	Via I. Nievo	Via Trecallo
Via C. Colombo	Via Ninguarda	Via Trento
Via Colonna	Via Nulli	Via Lungo Lario Trento
Via Comerio	Via Odescalchi	Via Tridi
Via Comolli	Via Oldelli	Via Lungo Lario Trieste
Via Conciliazione	Via Olgati	Via Urbano II
Via Conconi	Via Olginati	Via Valbasca
Piazza Concordia	Via Oltrecolle	Via Valera
Via Confalonieri	Via Orazio	Via Valeria
Via Coni Zugna	Via Orelli	Via Valleggio
Via Coretta	Via Oriani	Via Valorsa
Via Corridoni	Via Ortelli	Via Varese
Via dei Cortili	Via Ortigara	Via Varesina
Via Cossoni	Via dell' Orto	Via Vassena
Via Crespi	Via Osjava	Via Vela
Via Cressoni	Via Ostinelli	Via Velzi
Via Crispi	Via Ovidio	Via Venini
Via Croce Rossa	Via Pacinotti	Viale Venturino
Via Crotto del Sergente	Via Pagani	Via Venusti
Via Cuzzi	Via Palestro	Piazza Verdi

Via D'Acquisto	Via Paluda	Via Verga
Via D'Azeglio	Via Pannilani	Via Vigano
Via D'Oggiono	Via Pantera	Via Villa Giovio
Via Pietro Da Breggia	Via Paradiso	Villa Olmo
Via Leonardo Da Vinci	Via Parini	Via dei Villini
Via Dante	Via Parrocchiale	Via Virgilio
Via De Cristoforis	Via Partigiani	Via Vitani
Via G. Di Vittorio	Via Pascoli	Piazza Vittoria
Via A. Diaz	Via Passeri	Via Vittorio Emanuele II
Via Don Bosco	Via Pastonchi	Via Vittorio Veneto
Via Don Gnocchi	Via Pastrengo	Via Vodice
Via Don Minzoni	Via Pasubio	Via Volpati
Via Don Monza	Via Pedemonte	Via Volpi
Via Donatori di Sangue	Salita Peltlera	Piazza Volta
Via del Dos	Via Perego	Via A. Volta
Via Dottesio	Via Perlasca	Passeggiata Voltiana
Via Dotti	Via del Pero	Via XX Settembre
Via Druso	Via Perpenti	Piazza XXIV Maggio
Piazza Duomo	Via Pertì	Via XXVII Maggio
Vicolo Duomo	Via Perugino	Via Zamenhof
Via Durini	Via Pessina	Via Zampiero
Via El Alamein	Via Petrarca	Via Zanella
Via Falciola	Via Petrolo	Via Zara
Via G. Ferrari	Via Piadeni	Via Zezio
Via Ferrata	Via al Piano	Via alla Zocca

## BIBLIOGRAFIA

- [ARE] ARERA. *ARERA - Prezzi e tariffe*. URL: <https://www.arera.it/it/prezzi.htm>.
- [DD05] Harvey M. Deitel e Paul J. Deitel. *Java: tecniche avanzate di programmazione*. 2005.
- [Fow] Martin Fowler. *UML Distilled*. Pearson. ISBN: 9788871925981.
- [IST] ISTAT. *Tavole di dati - ambiente urbano*. URL: <https://www.istat.it/it/archivio/217887>.
- [Pat15] David A Patterson. *Struttura e progetto dei calcolatori / David A. Patterson, John L. Hennessy*. 4. ed. italiana condotta sulla 5. ed. americana. Bologna: Zanichelli, 2015. ISBN: 978-88-08-35202-6.
- [Sco] Kendal Scott. *UML explained*. Addison-Wesley. ISBN: 9788871921204.
- [sof] Docente ed esercitatori del corso di ingegneria del software.  
«Esercitazioni di ingegneria del software».
- [Svi] Ministero dello Sviluppo Economico. *Piano strategico banda ultralarga - documentazione - glossario*. URL: <http://bandaultralarga.italia.it/documentazione/glossario/>.
- [VVA] AA. VV. *Class HTMLDocument - javax.swing.text.html.HTMLDocument*. URL: <https://docs.oracle.com/javase/10/docs/api/javax/swing/text/html/HTMLDocument.html>.
- [VVb] AA. VV. *JMapView*. URL: <https://wiki.openstreetmap.org/wiki/JMapView>.
- [VVc] AA. VV. *OpenStreetMap*. URL: <https://www.openstreetmap.org/>.
- [VVd] AA. VV. *Package java.rmi*. URL: <https://docs.oracle.com/javase/10/docs/api/java/rmi/package-summary.html>.
- [VVe] AA. VV. *Package javax.swing*. URL: <https://docs.oracle.com/javase/10/docs/api/javax/swing/package-summary.html>.