



POLITECNICO
MILANO 1863

Corso di Ingegneria del Software
Prof. Marco Brambilla

Prova

Finale

Documento finale

Gruppo 2
Cappelletti Andrea | Primerano Matteo | Maglione Sandro

Anno accademico 2018-2019

Note conclusive

Durante gli incontri con il Prof. responsabile del progetto, dopo la consegna delle tre parti (requisiti, design, java), abbiamo preso nota delle osservazioni di quest'ultimo e aggiornato il progetto per sistemare i suoi punti deboli.

Di seguito viene riportata una lista delle modifiche apportate, riferite alle varie parti, in vista della consegna finale:

- Requisiti
 - Sistemato l'i* diagram
 - Aggiunte soluzioni ai possibili rischi ("come possono essere minimizzati?")
- Design
 - Sistemato Component Diagram
 - Sistemato Deployment Diagram
 - Sistematate immagini dei diagrammi sgranate
 - Diagrammi aggiornati dopo aver scritto il codice

INDICE DEI CONTENUTI

- [Requisiti](#)
- [Design](#)
- [Java e testing](#)

Requisiti

1 | Tema

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

- Sistema centrale: incaricato di memorizzare tutte le informazioni di stato, inviare notifiche a sistemi esterni in caso di specifici eventi, mostrare lo stato dell'intero sistema e sottosistemi. Il sistema quindi include una interfaccia utente che consente di esplorare le varie informazioni attuali.
Opzionale: è possibile decidere di mostrare i dati anche in un qualche tipo di forma grafica (diagrammi, mappe. ecc.).
- Centraline stradali: incaricate di monitorare il flusso di traffico del segmento stradale in cui collocate e inviarlo al sistema centrale con periodicità proporzionale all'ammontare di traffico
- Centraline automobilistiche: incaricate di inviare con periodicità fissa il dato di velocità (e posizione) del veicolo su cui sono installate
- Applicazioni mobili: installate su telefono cellulare e incaricate di inviare al sistema centrale esplicite segnalazioni di traffico (coda, con posizione GPS) da parte degli utenti / guidatori. Le applicazioni inoltre ricevono notifiche dal sistema centrale per qualsiasi evento di traffico (coda, velocità lenta, traffico elevato) in un raggio fisso dalla posizione (ultima registrata) del telefono

Specificare, progettare e implementare il sistema distribuito necessario, coprendo: sistema centrale, applicazione mobile, e una a scelta tra centralina stradale e centralina automobilistica. Definire esplicitamente tutti i formati dei dati scambiati e le modalità di scambio (protocollo). E' possibile raffinare i requisiti ed aggiungere ipotesi e assunzioni sul contesto, sensate e in linea con quanto indicato nei requisiti. Tali estensioni devono essere esplicitamente riportate nella documentazione di progetto (sezione specifica requisiti).

2 | Studio di fattibilità

Intro

Lo scopo di questo documento è quello di determinare la fattibilità dell'aggiornamento software per il sistema di monitoraggio e controllo integrato del traffico cittadino dell'azienda committente. Il software di gestione del sistema attualmente in uso, risalente all'anno 2001, risulta obsoleto, incompleto e inefficace. Sorge la necessità di un rinnovo del sistema in linea con le tecnologie e i metodi di sviluppo moderni.

Why / Scope

L'obiettivo è di sviluppare un software interamente nuovo, composto da un sistema centrale che interagisce con dei sottosistemi distribuiti: centraline stradali e applicazioni mobile. L'azienda committente è già munita di centraline stradali installate, le quali monitorano il flusso di traffico dell'intera città di Milano e sono in funzione 24/7. Il sistema centrale in oggetto riceverà informazioni da queste ultime, andando poi a elaborare e memorizzare i dati ricevuti. Ogni automobilista potrà in seguito interagire con il sistema tramite l'applicazione mobile, inviando esplicite segnalazioni e ricevendo notifiche in caso di eventi specifici, quali code, velocità lenta e traffico elevato.

Benefits

- Maggiore semplicità di gestione e utilizzo del sistema
- Potenzialità di diffusione e guadagno elevate
- Sistema completamente automatizzato, facilmente scalabile e espandibile
- Sfruttamento delle centraline stradali preesistenti e conseguente risparmio economico
- Possibilità di analizzare i dati sul traffico registrati per il miglioramento e l'ottimizzazione costante del software
- Servizio offerto più efficiente e in linea con le moderne tecnologie

Obstacles

- Progettazione e realizzazione dell'intero sistema in soli tre mesi
- Necessità di un sistema centrale abbastanza potente da poter gestire l'interazione continua con i sottosistemi connessi
- Tempo necessario per familiarizzare con le tecnologie e gli strumenti di sviluppo richiesti per la creazione del sistema
- Piattaforma di Version Control (GitLab) mai utilizzata fino ad ora

Risks

- Disinteresse da parte dell'utente finale all'utilizzo dell'applicazione
Problema di marketing risolvibile tramite un'opportuna campagna pubblicitaria.
- Informazioni non attendibili provenienti dalle segnalazioni dell'utente
Filtro delle segnalazioni in strade monitorate tramite le centraline, segnalazioni in strade non monitorate da parte degli utenti.
- L'hardware attualmente in uso non supporta le nuove tecnologie
Rimpiazzare le vecchie centraline con hardware di nuova generazione.
- Software inefficiente per la gestione di un numero di dati eccessivamente elevato o non previsto
Rendere il software versatile ed espandibile nel tempo.
- Necessità di una formazione del personale per l'utilizzo del nuovo sistema, con possibile malcontento e avversione all'aggiornamento
Interfaccia grafica del software semplice ed intuitiva. Fornire un manuale di funzionamento.
- Sistema basato su un unico sistema centrale, totalmente inutilizzabile in caso di guasti o malfunzionamenti
Suddivisione del sistema centrale in più sottosistemi che lavorano in modo indipendente l'uno dall'altro.

Resources needed

Il sistema verrà prodotto da un gruppo di lavoro composto da tre ingegneri informatici. Ognuno avrà a disposizione un laptop personale e la collaborazione avverrà tramite GitLab, piattaforma di Version Control. I software necessari sono:

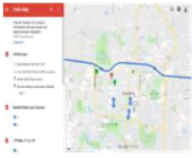

- **OpenOME**: per lo studio dei requisiti e la produzione di un diagramma i*;
- **StarUML**: per il design UML del sistema;
- **IntelliJ**: IDE scelto per la produzione effettiva del codice in linguaggio Java.

Il tempo a disposizione per la produzione completa del progetto è di tre mesi, con consegna fissata in data 14 Gennaio 2019.

3 | Data Dictionary

Si riporta di seguito il data dictionary del progetto con la definizione dei termini utilizzati nelle specifiche e nel diagramma i*.

Nome	Definizione	Sinonimi	Tipologia di dato	Esempi	Derivato?
Sistema centrale	Centro di controllo e gestione delle funzionalità del servizio	---	Sottosistema	—	No
Monitoraggio	Acquisizione ed elaborazione informazioni di stato	Raccogliere dati	Procedura	—	No
Opera in modo distribuito	Processi interconnessi in cui le comunicazioni avvengono solo tramite lo scambio di messaggi	---	Concetto astratto	—	No
Controllo integrato	Gestione processi indipendente per singolo dispositivo	---	Concetto astratto	—	No
Traffico cittadino	Quantità di veicoli nell'unità di tempo	Flusso di traffico	Dato numerico	"50 veicoli al minuto", "3000 veicoli all'ora"	No
Memorizzare	Procedura di salvataggio delle informazioni di stato all'interno del Sistema Centrale	---	Procedura	—	No

Informazioni di stato	Insieme delle informazioni memorizzate nel sistema centrale	Evento di traffico, stato, stato complessivo, informazioni di traffico	List	{coda, intensità elevata, via Anzani}	Segnalazion e di traffico, traffico cittadino
Notifica	Informazione scambiata fra il sistema centrale, applicazione mobile e utente	---	List	<i>Vedi informazioni di stato</i>	Informazion e di stato
Inviare notifiche	Scambio di dati tra il sistema centrale, l'applicazione mobile e l'utente	---	Procedura	---	No
Sottosistema	Componenti di cui è composto il sistema	---	Dipende dall'implementazione	Centralina stradale, applicazione mobile, sistema centrale	No
Interfaccia utente	Insieme dei diagrammi grafici addetti a visualizzare i dati sul traffico	---	GUI		Diagrammi grafici
Diagrammi grafici	Rappresentazione visuale dei dati memorizzati nel Sistema Centrale	---	GUI		Informazion e di stato
Centralina stradale	Dispositivo atto a rilevare e segnalare la presenza di traffico al Sistema Centrale	---	Sottosistema	---	No
Informazione periodica	Informazione che viene inviata periodicamente dalla centralina stradale al sistema centrale	---	List	<i>Vedi informazioni di stato</i>	No

Applicazione mobile	Software installato sui telefoni cellulari degli utenti atti a ricevere e inviare segnalazioni	---	Sottosistema	---	No
Telefono cellulare	Dispositivo fisico utilizzato dall'utente per ricevere e inviare segnalazioni di traffico	---	Dispositivo hardware	---	No
Segnalazione di traffico	Dato scambiato tra Sistema Centrale e Applicazioni Mobili e tra Sistema Centrale e Centraline Stradali	---	List	Incidente, code, velocità lenta	No
GPS	Dispositivo embedded, presente all'interno del mobile, fornisce la posizione dell'automobilista	---	Dispositivo hardware	---	No
Automobilista	Utente finale del servizio. Fornisce inoltre segnalazioni su eventi al Sistema Centrale	Utente, guidatore	Persona fisica	---	No
Raggio fisso dalla posizione	Perimetro circolare che identifica l'area di interesse di un telefono cellulare	Raggio di posizione	Dato numerico	"500 metri", "250 metri"	No
Posizione	Dato rilevato tramite GPS	Posizione GPS	Vettore	"45.520664, 9.139561"	GPS

4 | Commento al Goal Diagram

Il progetto i* è composto da due diagrammi: un diagramma SDM e un diagramma SRM. Il primo, che ha lo scopo di mettere in evidenza le risorse, ha permesso, nell'SRM, di relazionarle ai goal di ogni attore e definire poi i task necessari per raggiungere questi ultimi.

I due diagrammi si trovano in appendice.

Progetto i* del software

L'SDM mette in evidenza le risorse scambiate tra i quattro attori, mentre nell'SRM vengono riportati gli hard-goal e i vari task. Gli obiettivi di cui si fa carico il sistema non ammettono, invece, più livelli possibili di soddisfazione, e, pertanto, non sono stati inseriti soft-goal; di fatto questi ultimi avrebbero reso il modello più confusionario e dispersivo. Si è stabilito di definire quattro attori che insieme interagiscono per soddisfare i requisiti.

I goal principali, ricavati direttamente dal tema, sono stati formattati in grassetto.

Le assunzioni effettuate in fase di progetto vengono riportate di seguito.

Generali

- Le informazioni di stato vengono memorizzate nel sistema centrale, il quale mette a disposizione un'interfaccia grafica per il loro consulto. Si è deciso di omettere un eventuale attore "Amministratore di sistema" in quanto la sua interazione con il sistema non ha particolare rilevanza nel contesto.

Sistema centrale

- Nella voce "informazioni in arrivo", si intendono sia le fonti di dati provenienti dalle centraline stradali, sia le quelle proveniente dagli utenti. Nelle fasi successive verranno specificati i metodi di integrazione delle due risorse.

Centralina stradali

- Il periodo di invio delle informazioni sul traffico viene calcolato in base all'ammontare di traffico (come richiesto nel tema). Si stabilirà in fase di design come avverranno le modalità di calcolo specifiche.

Utente

- L'attore "Utente" è predisposto per l'utilizzo dell'applicazione mobile; di fatto non ha veri e propri goal, ma solo il compito di condividere la risorsa "Segnalazione" e ricevere la risorsa "Notifica".

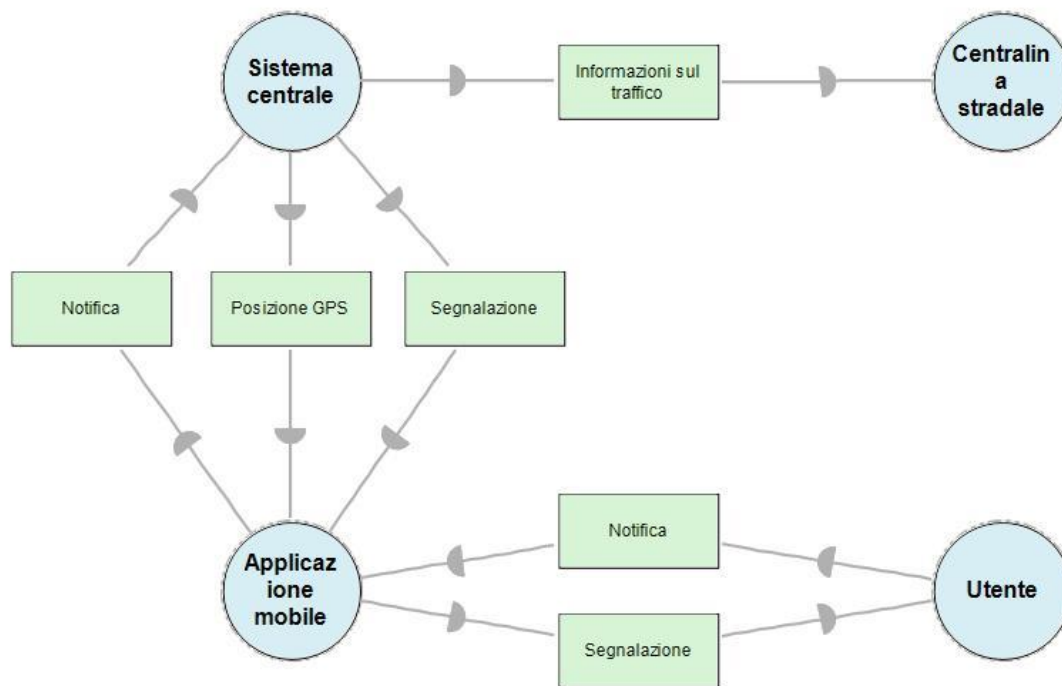
Applicazione mobile

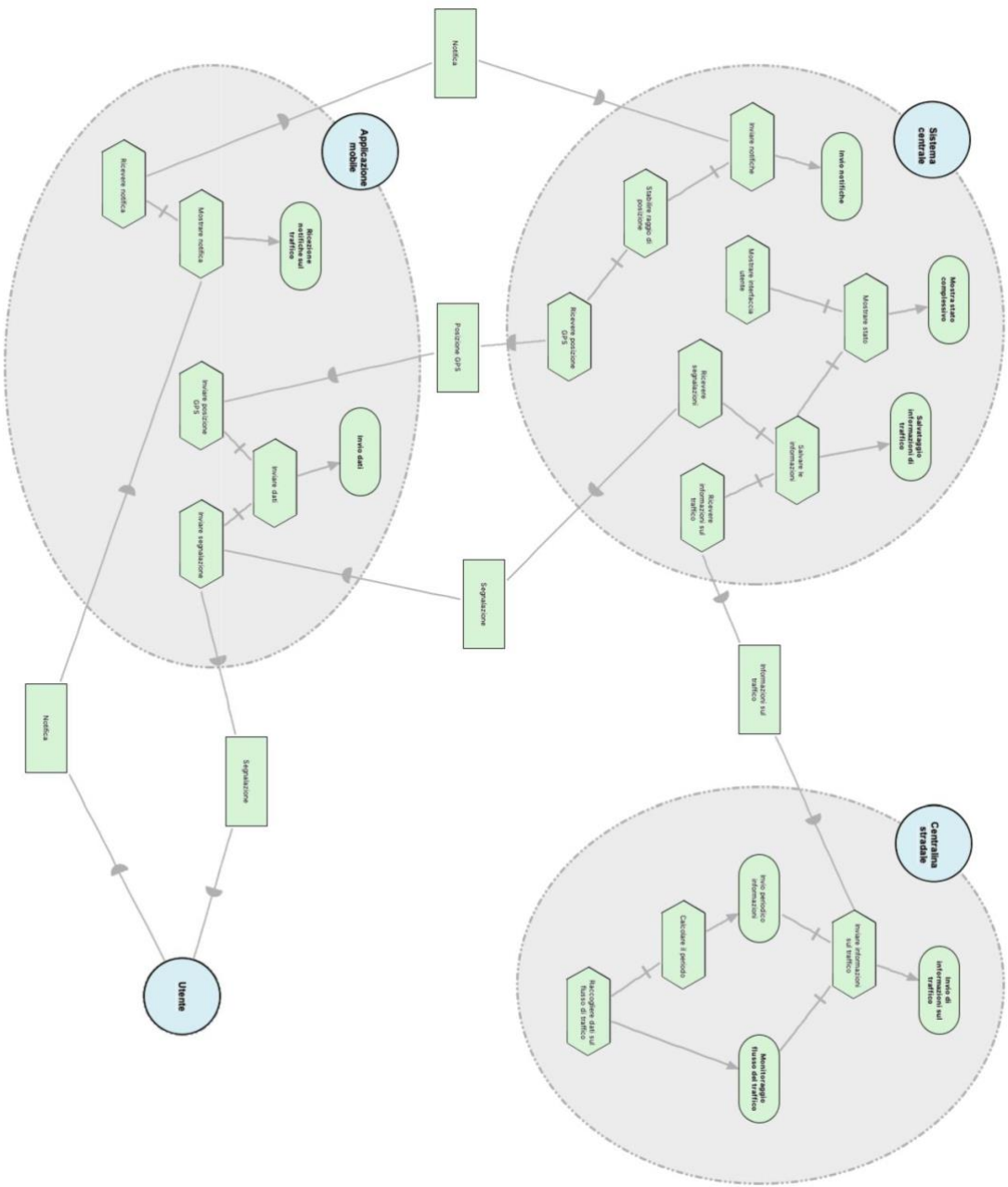
- L'applicazione mobile fa da tramite tra il sistema centrale e l'utente per quanto riguarda lo scambio delle risorse "Segnalazione" e "Notifica".
- Si assume che la posizione GPS del telefono cellulare venga inviata al sistema centrale ad intervalli regolari. Quest'ultimo fa uso della risorsa per calcolare l'area di interesse prima di procedere con l'invio della notifica relativa all'evento di traffico.
- Nel momento in cui un utente effettua una segnalazione tramite l'applicazione mobile, si sottintende che avvenga anche l'invio della posizione GPS in concomitanza.

5 | Appendice

Di seguito vengono riportati i diagrammi i*, realizzati con il tool OpenOME.

1 | SDM





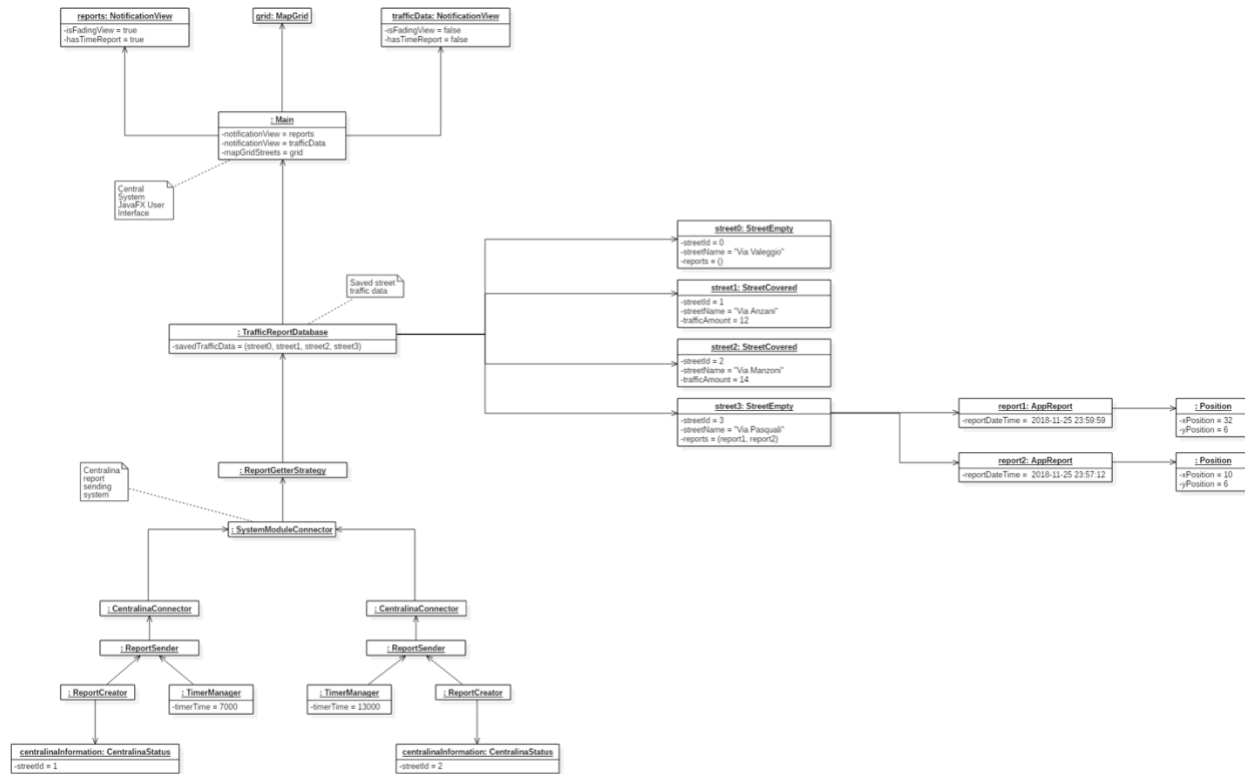
Design

In questo documento vengono presentati i diagrammi UML relativi alla fase di Design del sistema per il controllo e monitoraggio del traffico cittadino. Per ogni diagramma segue un commento esplicativo.

INDICE DEI CONTENUTI

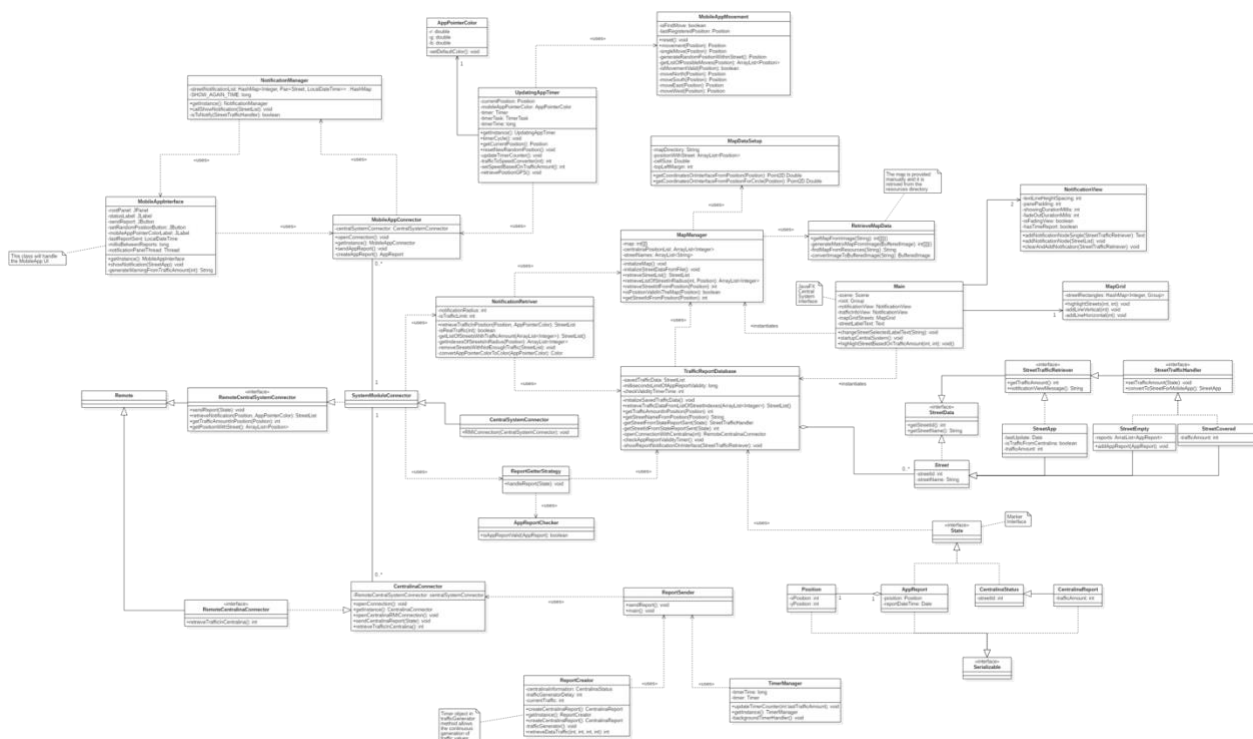
- [Object Diagram](#)
- [Class Diagram](#)
- [Sequence Diagram \(centralina\)](#)
- [Sequence Diagram 2 \(Sistema centrale\)](#)
- [Collaboration Diagram](#)
- [Use case Diagram](#)
- [Statechart Diagram 1 \(centralina\)](#)
- [Statechart Diagram 2 \(generale\)](#)
- [Activity Diagram 1 \(AppMobile – invio segnalazione\)](#)
- [Activity Diagram 2 \(AppMobile – ricezione notifica\)](#)
- [Component Diagram](#)
- [Deployment Diagram](#)

Object Diagram



Questo *Object Diagram* mostra gli oggetti coinvolti nel processo che porta il dato di traffico ad essere visualizzato all'interno dell'interfaccia utente nel sistema centrale. Due centraline con le loro rispettive streetId inviano il dato che viene salvato all'interno della variabile savedTrafficData come mostrato. Questo dato verrà poi inviato alla classe **Main** che si occupa di aggiornare l'interfaccia grafica, mostrando i dati ricevuti all'interno delle views (**NotificationView** e **MapGrid**) ad essa associate.

Class Diagram



Il sistema è diviso in quattro aree che svolgono compiti diversi tra di loro e che interagiscono per scambiarsi informazioni e dati.

- L'applicazione mobile è composta dalle classi **NotificationManager**, **MobileAppInterface** e **MobileAppMovement**.
- La centralina è composta dalle classi **CentralinaConnector**, **ReportSender**, **ReportCreator** e **TimerManager**.
- Il sistema centrale è invece diviso in varie classi che gestiscono la logica di salvataggio dei dati interi e dall'interfaccia grafica.

Il sistema presenta tre **façade** differenti (classi-Connector), che sono responsabili dell'interazione delle centraline e delle applicazioni mobile con il sistema centrale.

L'invio dei dati delle centraline è gestito da un timer presente nel **TimerManager**. Allo scadere del timer viene richiesta la creazione di un oggetto CentralinaReport che viene inviato al sistema centrale. La dinamica del traffico sulla strada di responsabilità della centralina viene simulata all'interno della classe **ReportCreator**. Anche l'applicazione mobile implementa un timer, che gestisce l'invio dell'aggiornamento della posizione GPS al sistema centrale. In particolare, la durata del timer è direttamente proporzionale alla velocità di movimento

dell'applicazione mobile, calcolata coerentemente alla quantità di traffico attraverso la classe **MobileAppMovement**.

Il sistema gestisce la ricezione del dato di traffico attraverso la classe **SystemModuleConnector**. Quando il sistema centrale riceve una segnalazione dall'utente, l'oggetto inviato AppReport passa attraverso la classe **AppReportChecker** la quale controlla la validità del dato prima di salvarlo all'interno del sistema (Proxy Pattern.)

Tutti i dati sul flusso di traffico vengono memorizzati all'interno della classe **TrafficReportDatabase** come oggetti di tipo **Street**, a loro volta contenuti in un'ArrayList in un oggetto di tipo **StreetList**. In particolare, viene salvata un'istanza della classe StreetEmpty nel caso la relativa strada non sia coperta da una centralina, e un'istanza della classe StreetCovered nel caso lo sia.

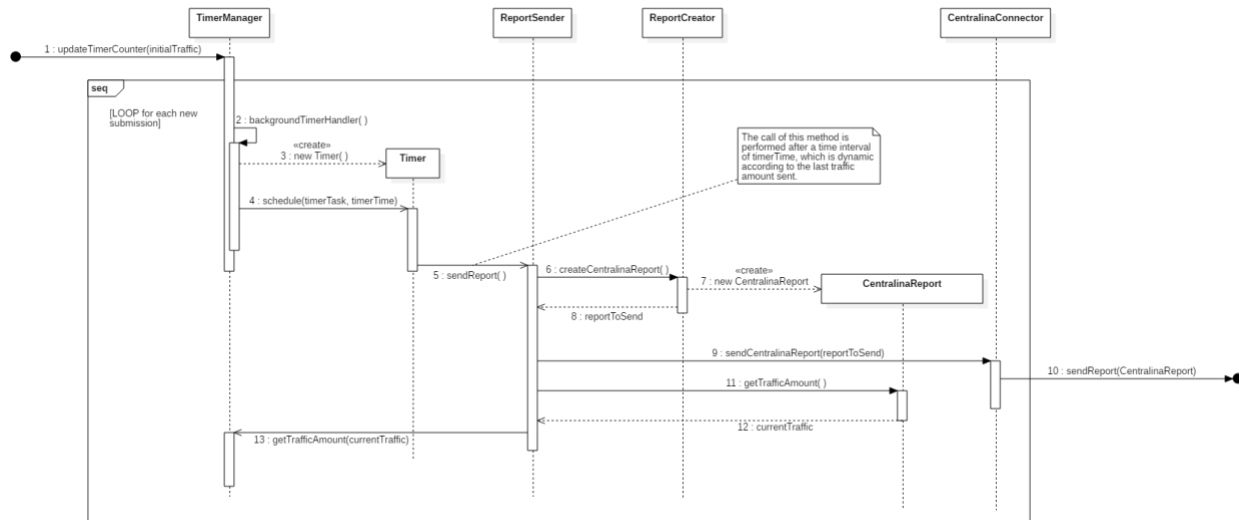
Il processo di notifica avviene nel momento in cui l'applicazione mobile invia la propria posizione al sistema centrale. Vengono recuperate e ritornate all'applicazione mobile tutte le strade trafficate in un raggio fisso dalla posizione segnalata. L'applicazione mobile controlla di non aver già ricevuto lo stesso dato di recente (per evitare lo "spamming" di notifiche) e procede a visualizzare la notifica sull'interfaccia.

L'interfaccia utente all'interno del sistema centrale è gestita dalla classe **Main**. All'avvio del sistema vengono create due liste di tipo **NotificationView**: una per mostrare lo stato attuale del traffico in tutte le strade registrate e l'altra per notificare l'arrivo di un dato di traffico da uno dei sotto-moduli. Viene inoltre creato un oggetto di tipo **MapGrid** il quale si occupa di mostrare e aggiornare una mappa che mostra l'evolversi del traffico in tempo reale. Ogni qual volta il sistema centrale riceve un nuovo dato di traffico, viene notificata la classe Main che si occuperà di aggiornare conseguentemente l'interfaccia grafica.

Gli oggetti di tipo State implementano il **Marker Interface Pattern**.

Le classi che svolgono funzioni in comune (di tipo Manager) sono state rese **Singleton**.

1 | Sequence Diagram



Questo *Sequence Diagram* descrive il processo di rilevamento del traffico e generazione del report da inviare al sistema centrale, eseguito da una centralina stradale.

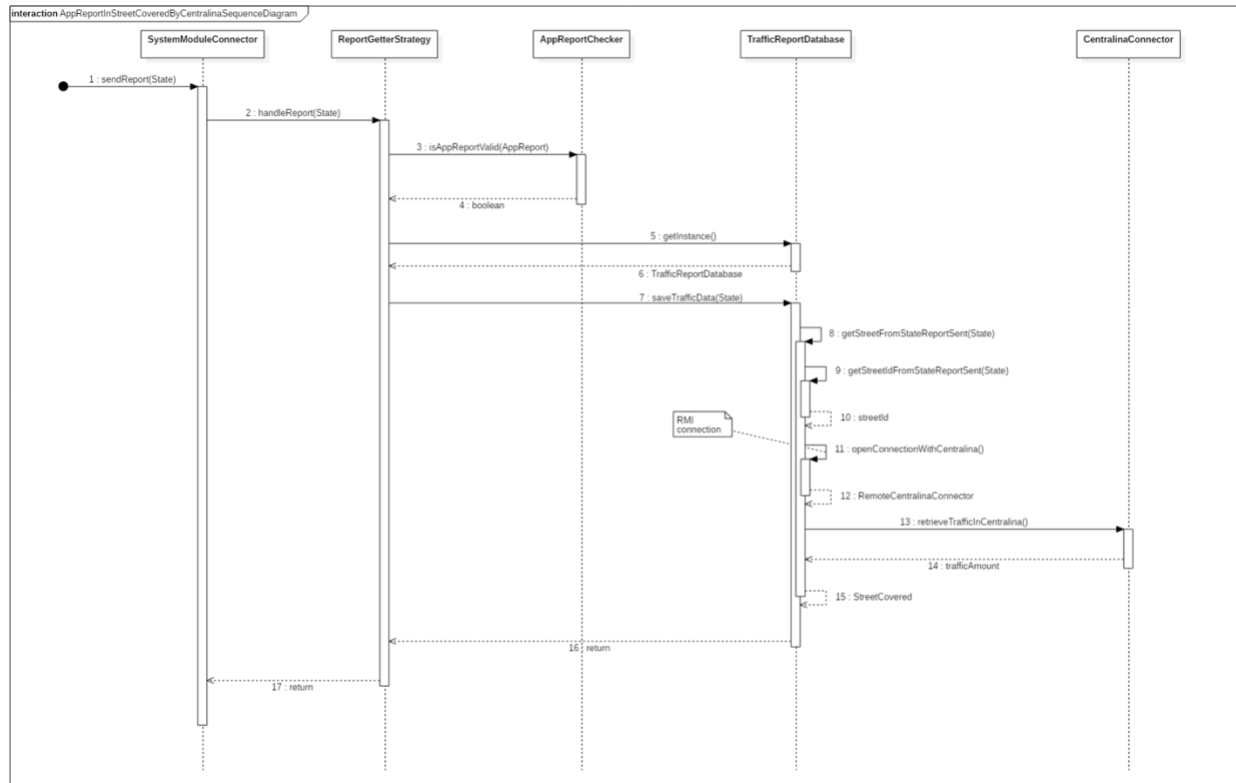
Sono state omesse la creazione degli oggetti principali della centralina, dopo la quale parte la chiamata iniziale `updateTimerCounter(initialTraffic)` e il comportamento della classe **CentralinaConnector** nel gestire la connessione tramite RMI al sistema centrale. Inoltre essendo le classi della centralina per lo più singleton vengono sottintese le chiamate ai metodi `getInstance`.

Dopo la chiamata iniziale possiamo distinguere 3 momenti principali:

- 1) Il **TimerManager** è responsabile di generare un nuovo timer al termine del quale viene incaricato il **ReportSender** di inviare un nuovo report (`sendReport()`). Il timer “scatta” dopo un tempo proporzionale all’intensità di traffico dell’ultima rilevazione.
- 2) Il **ReportSender** si occupa di far generare al **ReportCreator** il nuovo report basato sui dati attuali.
- 3) Il **ReportSender** fornisce il report a **CentralinaConnector** (`sendReport(reportToSend)`) che provvede ad inviarlo al sistema centrale e fornisce anche il valore di traffico di quest’ultimo report al **TimeManager**.

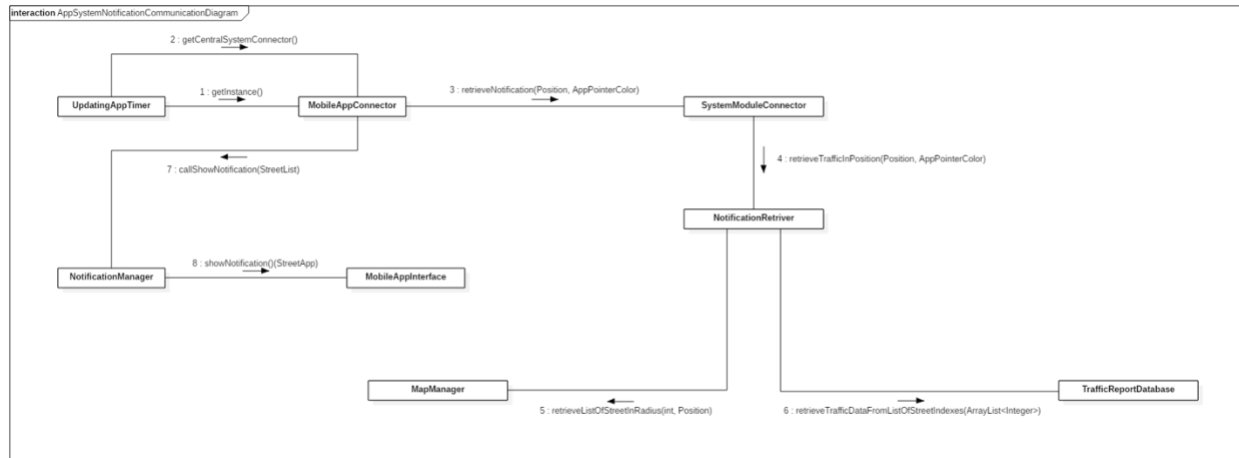
Questi passaggi avvengono in loop per tutto il funzionamento della centralina.

2 | Sequence Diagram



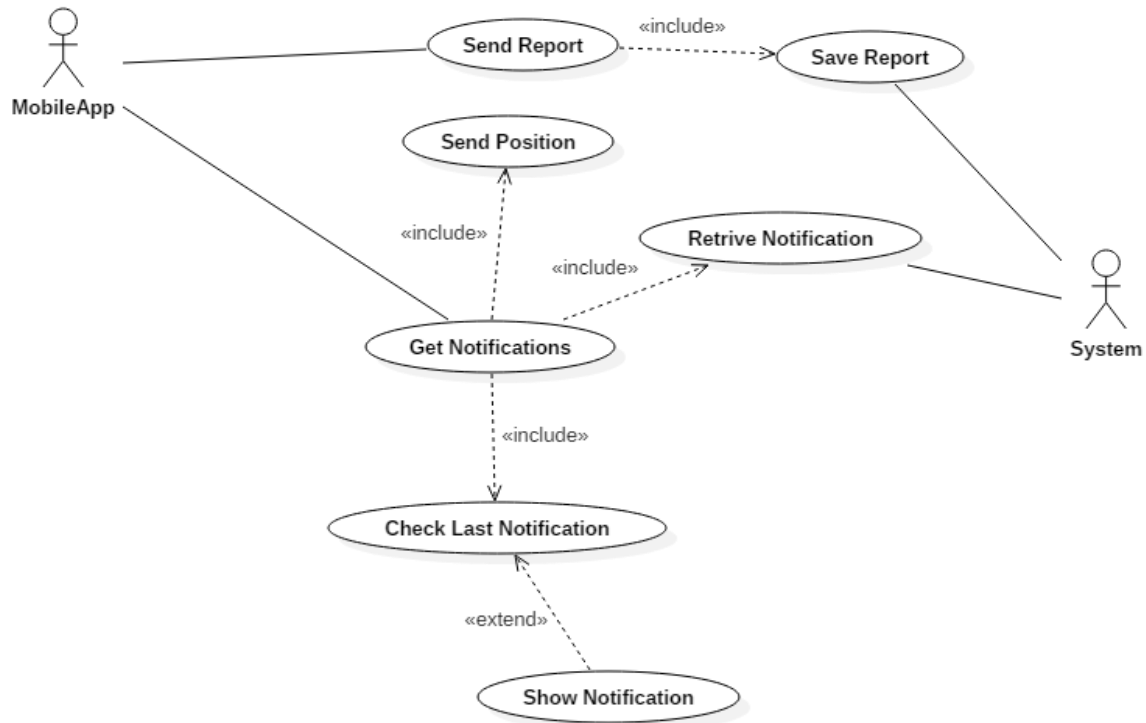
Questo *Sequence Diagram* descrive il processo di salvataggio di una segnalazione ricevuta da un'applicazione mobile nel caso in cui questa si trovi in una strada coperta da una centralina. Inizialmente viene controllata la validità della segnalazione attraverso la classe **AppReportChecker**, la quale controlla che la posizione inviata sia effettivamente associata ad una strada nella mappa (Proxy Pattern.) Il dato viene poi inviato alla classe **TrafficReportDatabase**. Quest'ultima riconosce la presenza di una centralina nella strada di interesse e provvede ad attivare una connessione remota per prelevare ed aggiornare il dato sul traffico reale.

Collaboration Diagram



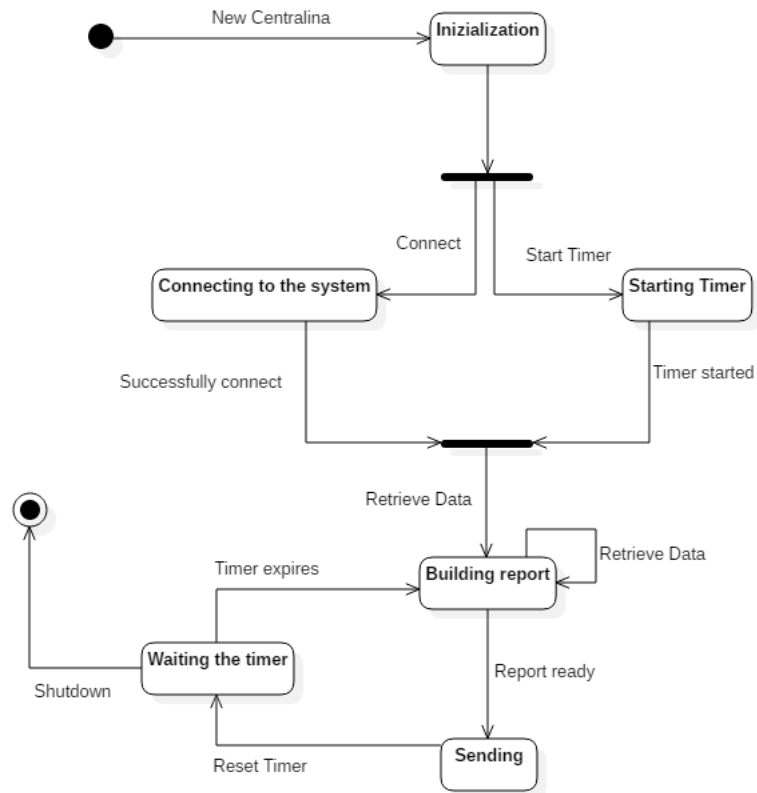
Questo *Collaboration Diagram* mostra l'interazione tra le classi dell'applicazione mobile e del sistema centrale nel contesto dell'invio di una notifica. L'applicazione mobile invia la propria posizione GPS al sistema centrale. Il sistema si occupa di recuperare il dato sul traffico in un raggio fisso dalla posizione e inviare un'eventuale notifica all'applicazione mobile.

Use case Diagram



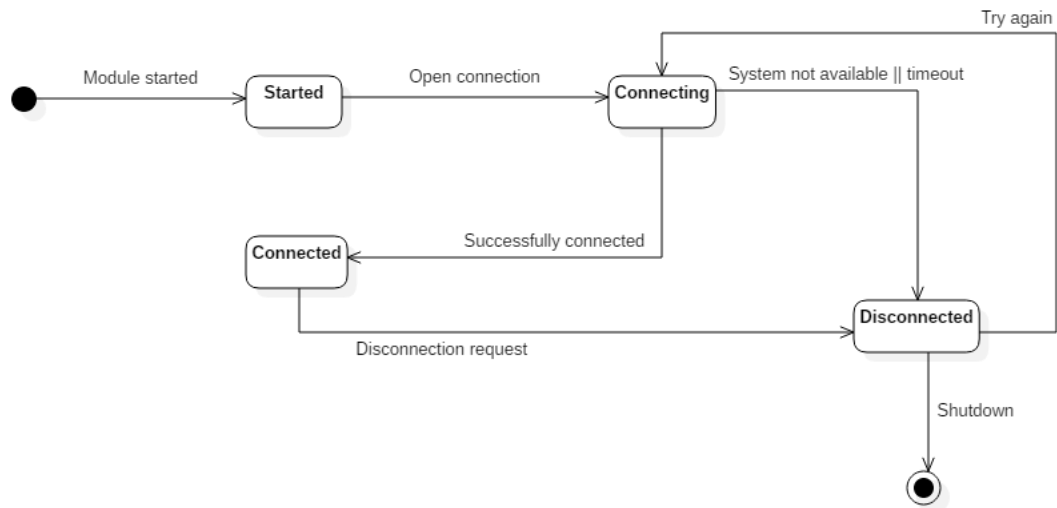
Questo *Use Case* propone i casi d'uso dell'applicazione mobile, ricavabili direttamente dal diagramma i* dei requisiti con ulteriori estensioni. In particolare, l'applicazione mobile può inviare una segnalazione al sistema e ricevere notifiche tramite un algoritmo che controlla quando è stata ricevuta l'ultima notifica in quella strada e decide se è il caso di visualizzarla.

1 | Statechart Diagram



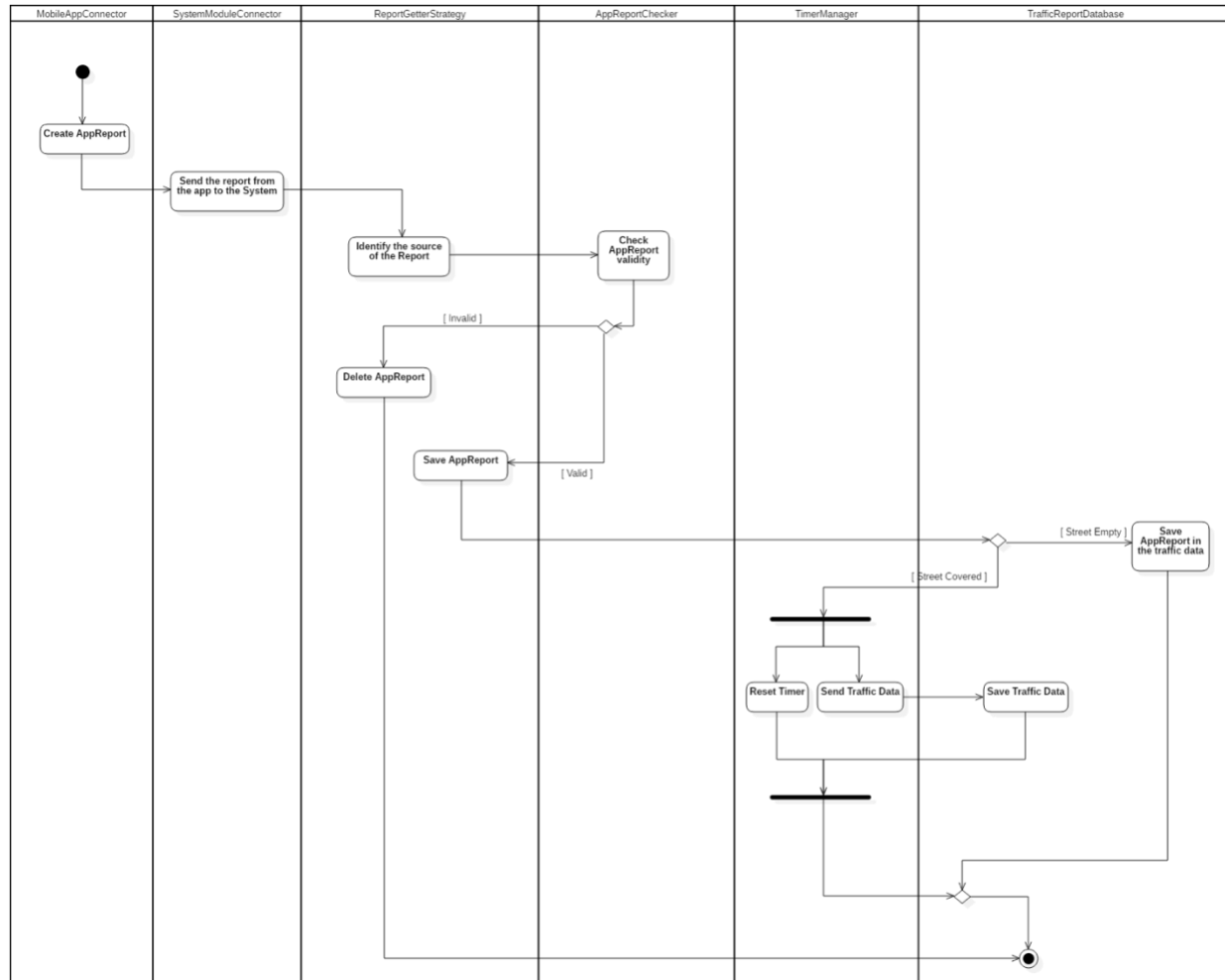
Questo *Statechart Diagram* modella le varie fasi di una centralina (dall'inizializzazione al suo utilizzo costante).

2 | Statechart Diagram



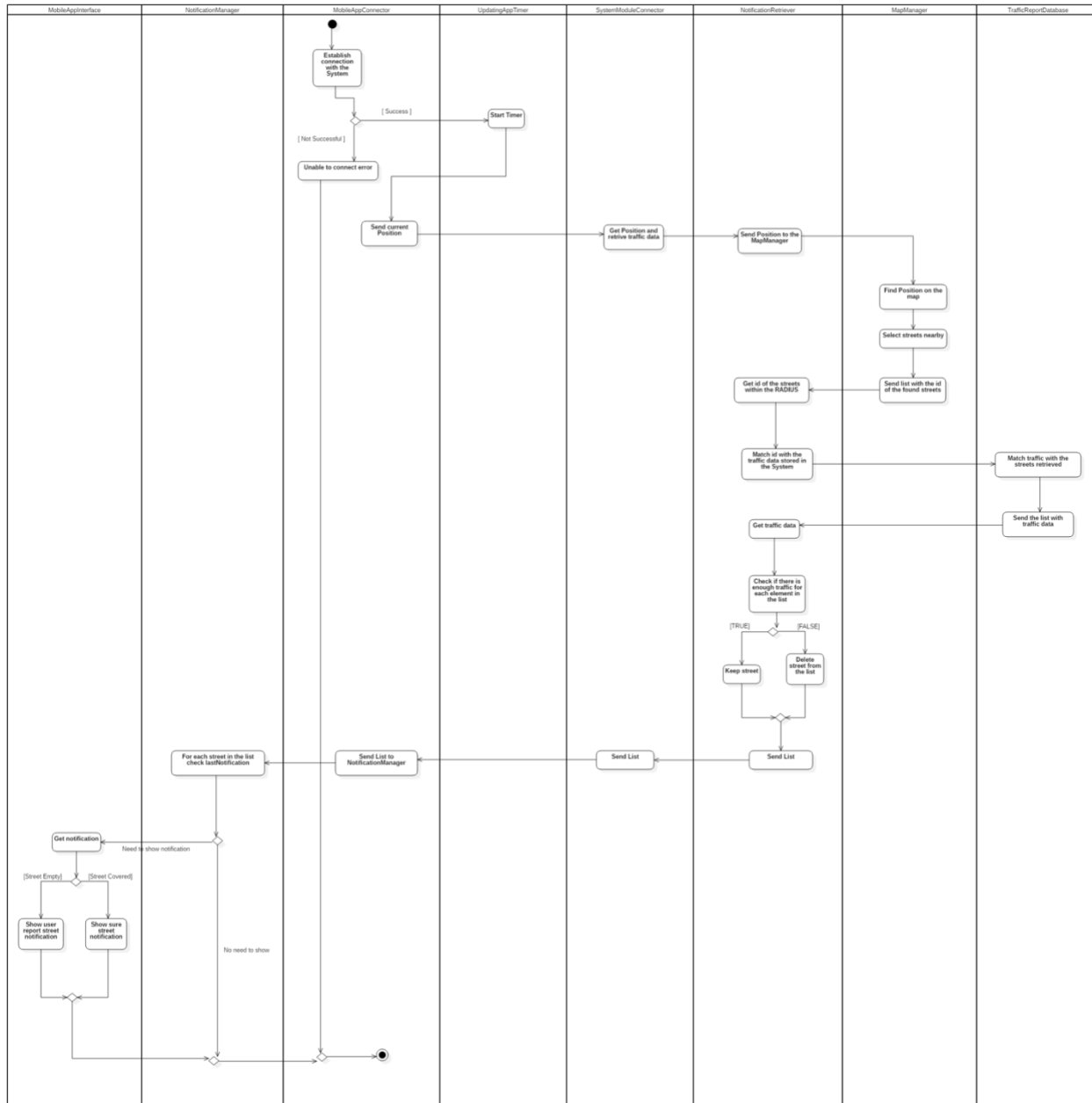
Questo *Statechart Diagram* modella la fase di connessione di tutti i moduli (Centralina e Applicazione Mobile) al sistema centrale tramite RMI.

1 | Activity Diagram



Questo *Activity Diagram* descrive la l'invio di una segnalazione di un'applicazione mobile al sistema centrale. L'applicazione mobile segnala la presenza di traffico nella propria posizione attuale, inviando tali informazioni al sistema centrale. Il sistema centrale controlla che la posizione inviata sia valida all'interno della mappa attraverso la classe **AppReportChecker**. Una volta confermata la validità del dato, l'informazione viene inviata alla classe **TrafficReportDatabase**. Nel caso in cui sia presente una centralina nella strada da cui è stata inviata la segnalazione (*Street Covered*), il sistema si collega con la centralina interessata per recuperare il dato reale sul traffico e resetta il timer interno alla classe **TimerManager**. Se invece la strada non è coperta da una centralina (*Street Empty*), il sistema salva le informazioni sulle segnalazioni (posizione, data e ora).

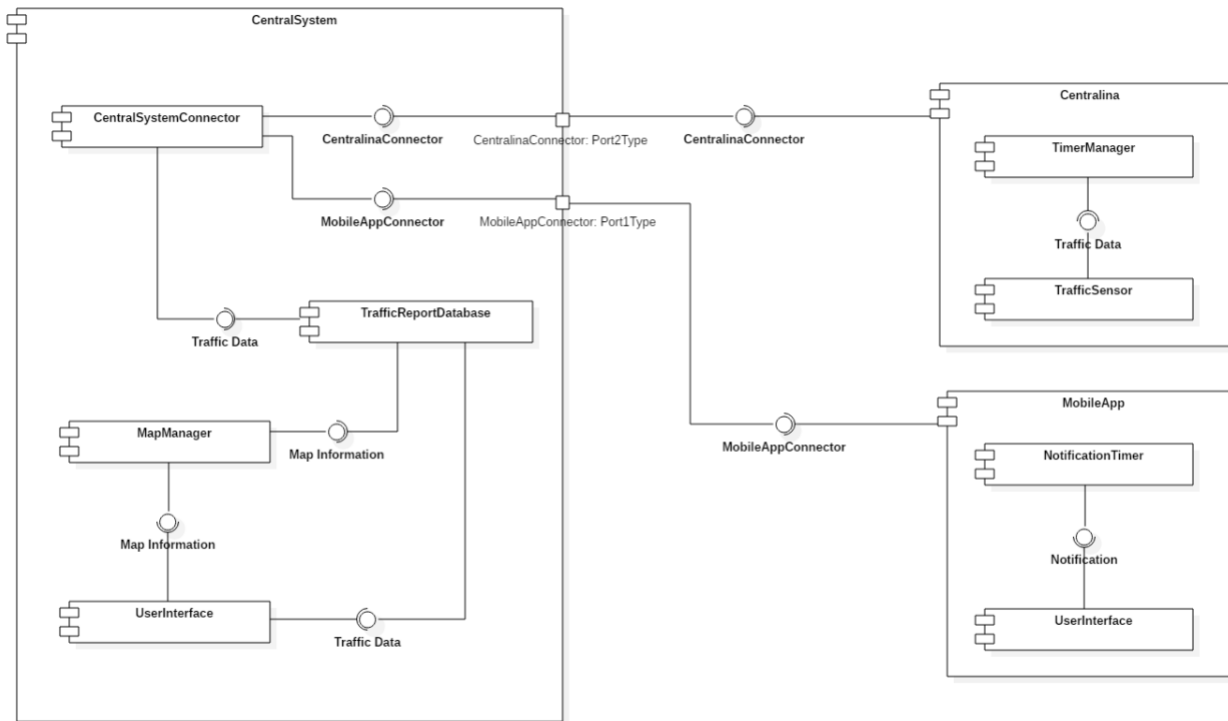
2 | Activity Diagram



Questo *Activity Diagram* il processo che intercorre un'applicazione mobile per la visualizzazione di una notifica. L'applicazione si connette al sistema centrale per richiede le notifiche sul traffico in un intorno fisso dalla propria posizione. Il sistema centrale controlla il traffico attuale in tutte le strade presenti nell'intorno della posizione ricevuta dall'applicazione mobile. Queste informazioni vengono recuperate dalle classi **MapManager** e **TrafficReportDatabase**. La classe **NotificationRetriever** filtra la lista di strade trovate, eliminando le strade poco trafficate. La lista risultante viene ritornata all'applicazione mobile. Qui il dato ricevuto viene elaborato dalla

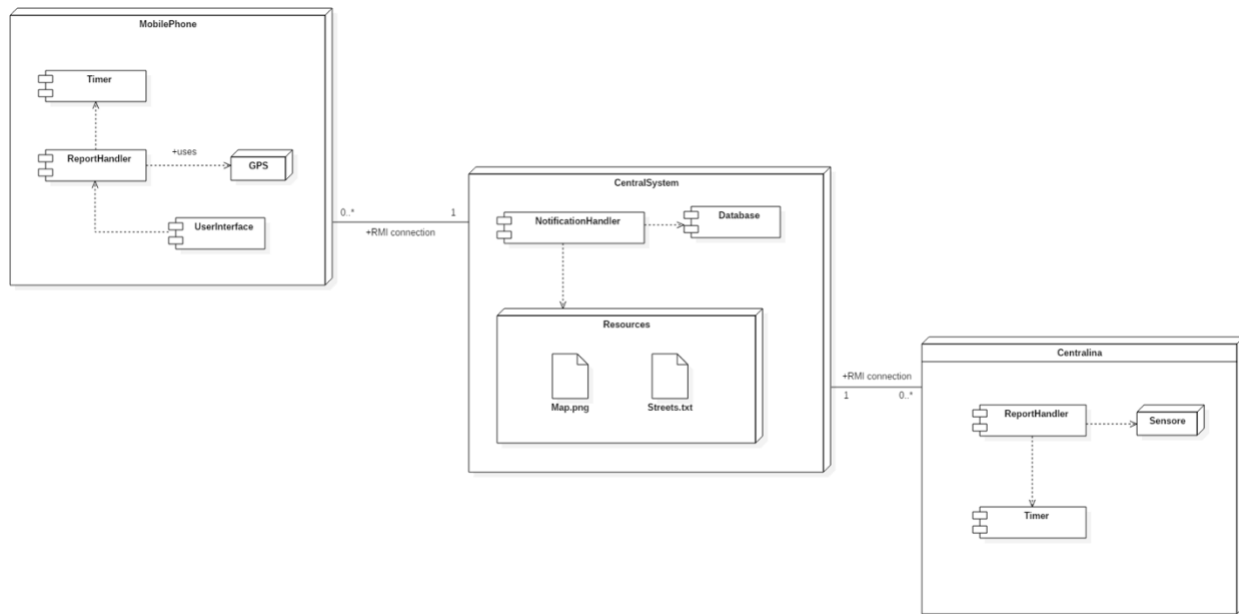
classe **NotificationManager** e inviato alla classe **MobileAppInterface** per essere mostrato sotto forma di notifica.

Component Diagram



Questo *Component Diagram* presenta la struttura dell'intero sistema, composto dal sistema centrale che si interfaccia separatamente con le varie centraline e applicazioni mobile connesse. Vengono mostrati inoltre i vari componenti interni ai vari moduli e i dati scambiati tra di loro.

Deployment Diagram



Il Deployment Diagram mostra la suddivisione dei vari nodi e dei moduli principali del progetto.

Il MobilePhone è il dispositivo dell'utente finale che si presuppone dotato di modulo GPS, quest'ultimo nella nostra implementazione è in realtà un generatore di posizioni, parte integrante della Mobile App.

Anche nel nodo Centralina il sensore è in realtà una simulazione del traffico sulla strada associata ed è quindi parte integrante della centralina stradale.

Nel CentralSystem abbiamo evidenziato un nodo interno Resource che contiene i 2 file relativi alla mappa e alle strade. Questi sono adattabili alle esigenze.

Tutti e 3 tre i nodi principali comunicano tramite protocollo RMI.

Java e Testing

In questo documento viene presentata la realizzazione del sistema per il controllo e monitoraggio del traffico cittadino in Java.

INDICE DEI CONTENUTI

- [Considerazioni e scelte](#)
- [Configurazione ed avvio](#)
- [Sistema Centrale](#)
- [Applicazione Mobile](#)
- [Centralina](#)
- [Testing](#)

Considerazioni e scelte

Inizialmente si pensava di integrare delle api che fornissero una mappa con i dati di traffico (come Google Maps), ma si è scelto di procedere senza l'utilizzo di queste ultime in quanto si è ritenuto opportuno mettersi in gioco e provare a realizzare l'intero sistema tramite gli strumenti a disposizione. Questa scelta è risultata molto costruttiva, sia a livello didattico che a livello di esperienza. Si è deciso di realizzare la mappa tramite una matrice bidimensionale e di mostrarla graficamente con JavaFx. Maggiori dettagli vengono discussi in seguito.

Configurazione e avvio

Parametri preimpostati

Di seguito la lista di tutti i parametri di configurazione impostati di default all'interno dell'intero sistema:

- Valore traffico rilevato da centraline
Valore massimo = **20**
Valore minimo = **1**
- Velocità AppMobile i.e. numero di step (pixel) che l'indicatore può percorrere in un ciclo
Valore massimo = **5**
Valore minimo = **1**
- Tempo di scadenza di una segnalazione prima di essere rimossa = **60000** (1 minuto)
- Tempo tra la visualizzazione di due notifiche uguali sulla stessa AppMobile = **30000** (30 secondi)
- Tempo tra un invio e il successivo dei dati sul traffico delle centraline al sistema centrale (valore reale proporzionale al flusso di traffico)
Tempo massimo = **20000** (20 secondi)
Tempo minimo = **3000** (3 secondi)

Istruzioni per l'uso

Il progetto richiede JavaFx per l'avvio dell'interfaccia grafica.

L'utilizzo di quest'ultimo prevede l'SDK versione 8 e Language level "Lambdas".

The screenshot shows the 'Project name' field set to 'traffic-monitor'. Under 'Project SDK', it shows '1.8 (java version "1.8.0_191")' with 'New...' and 'Edit' buttons. Under 'Project language level', it shows '8 - Lambdas, type annotations etc.' with a dropdown arrow. Under 'Project compiler output', it shows the path '/Users/andreacappelletti/IdeaProjects/traffic-monitor/out' with a folder icon button.

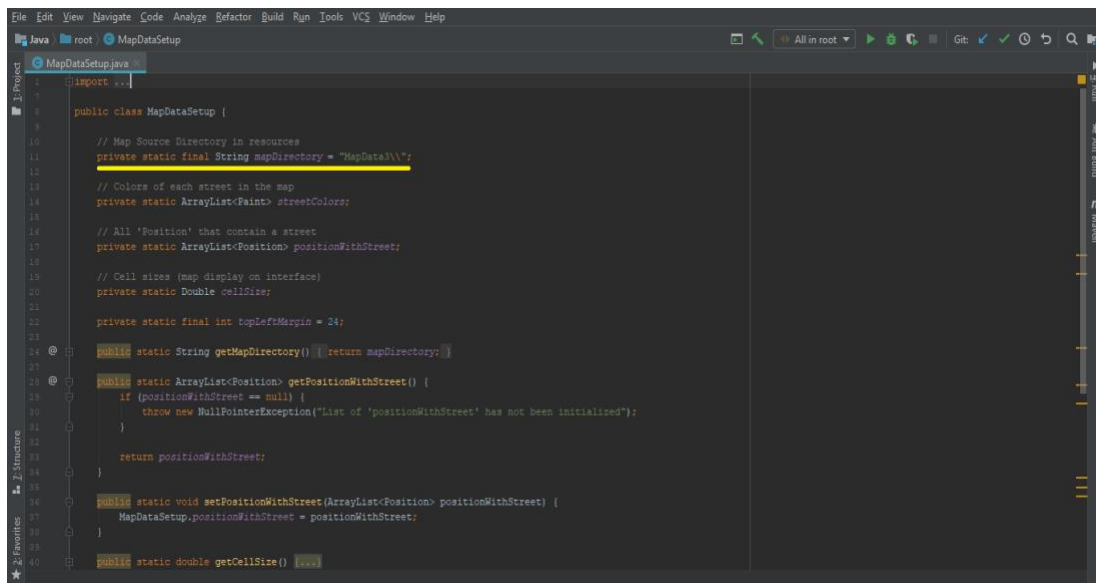
1. Indicare la mappa che si vuole visualizzare modificando la variabile `mapDirectory` all'interno della classe **MapDataSetup**.

Il percorso differisce in base al sistema operativo che si sta utilizzando.

Esempio: si supponga di voler aggiungere la mappa della directory MapData3:

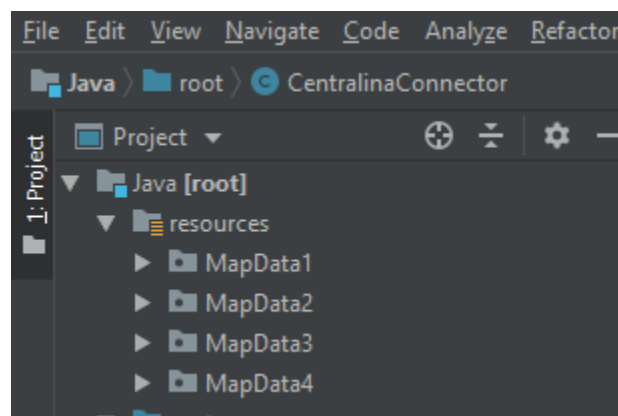
OSx: "MapData3/"

Windows: "MapData3\\"



```
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Java root MapDataSetup
MapDataSetup.java
1 import ...
2
3 public class MapDataSetup {
4
5     // Map Source Directory in resources
6     private static final String mapDirectory = "MapData3\\";
7
8     // Colors of each street in the map
9     private static ArrayList<Paint> streetColors;
10
11     // All 'Position' that contain a street
12     private static ArrayList<Position> positionWithStreet;
13
14     // Cell sizes (map display on interface)
15     private static Double cellSize;
16
17     private static final int topLeftMargin = 24;
18
19     public static String getMapDirectory() { return mapDirectory; }
20
21     public static ArrayList<Position> getPositionWithStreet() {
22         if (positionWithStreet == null) {
23             throw new NullPointerException("List of 'positionWithStreet' has not been initialized");
24         }
25         return positionWithStreet;
26     }
27
28     public static void setPositionWithStreet(ArrayList<Position> positionWithStreet) {
29         MapDataSetup.positionWithStreet = positionWithStreet;
30     }
31
32     public static double getCellSize() { ... }
33 }
```

Vengono rese disponibili quattro mappe nella cartella *resources*: MapData1, MapData2, MapData3, MapData4.

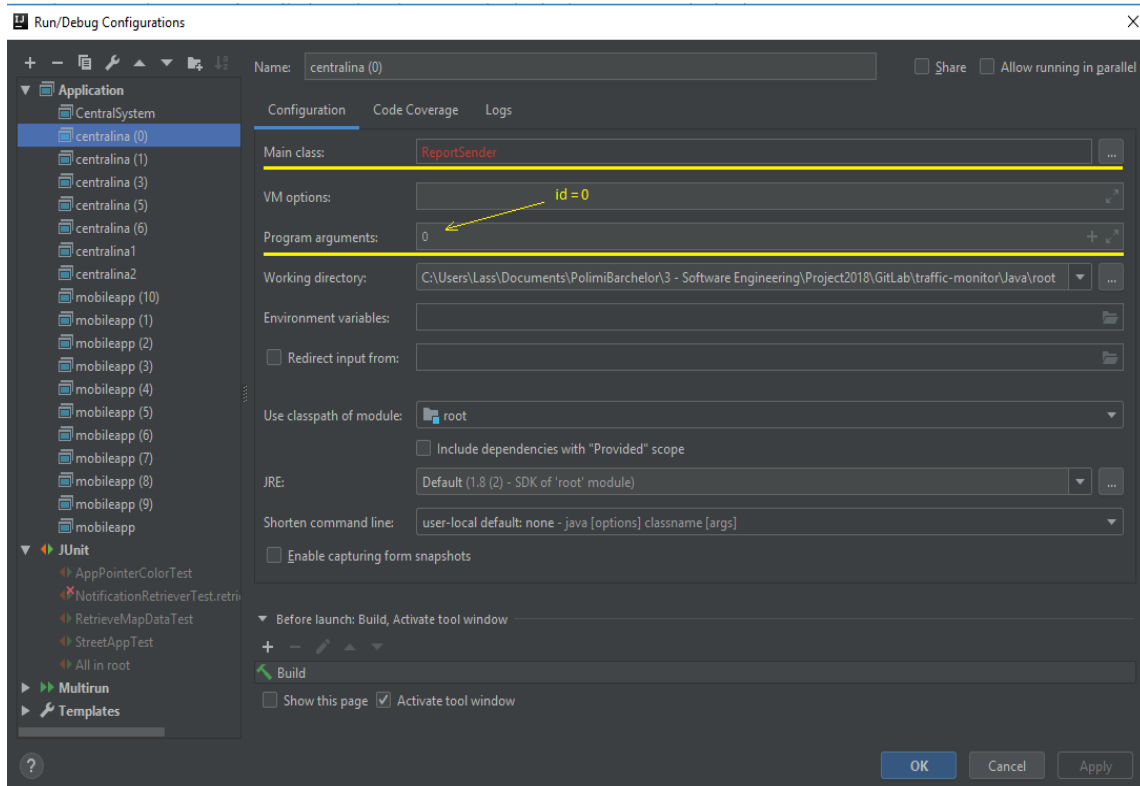


Nonostante ciò è sempre possibile aggiungere una configurazione a piacere seguendo lo standard utilizzato.

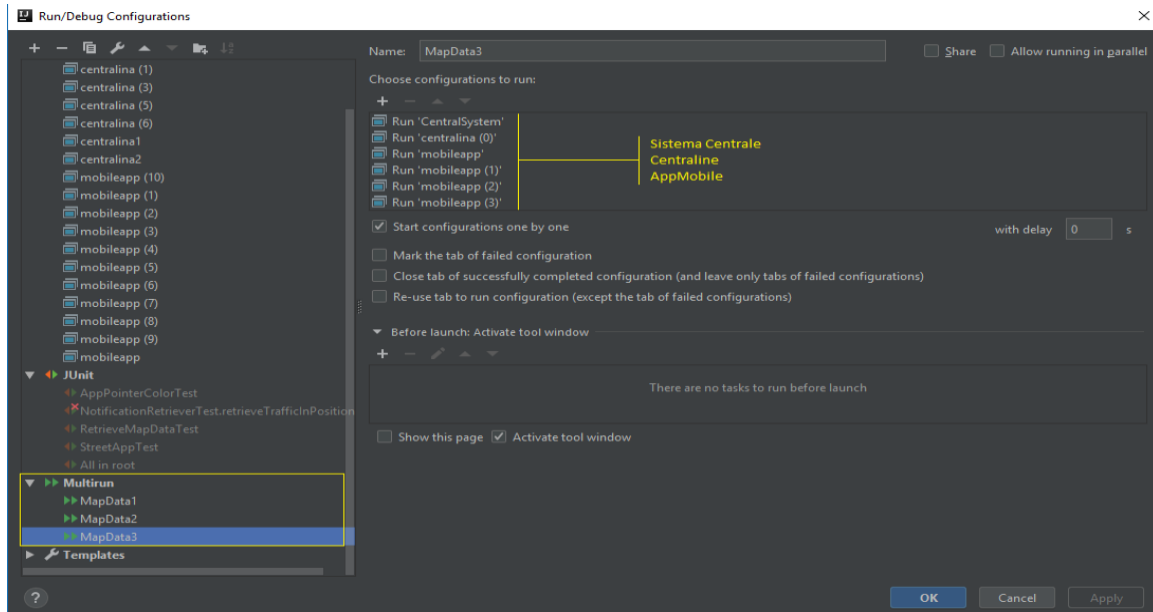
2. Ogni mappa necessita di un numero di centraline preimpostato. Ad ogni centralina dovrà essere assegnato uno specifico numero identificativo come parametro *args* al main di avvio. Di seguito la configurazione necessaria per ogni mappa presente:

MainClass Centralina: ReportSender

- MapData1:
Numero di centraline da istanziare: 2
id = 1, id = 5
- MapData2:
Numero di centraline da istanziare: 3
id = 0, id = 3, id = 6
- MapData3:
Numero di centraline da istanziare: 1
id = 0
- MapData4:
Numero di centraline da istanziare: 6
id = 0, id = 3, id = 6, id = 12, id = 16, id = 19



3. Bisogna avviare il sistema centrale per primo (classe **Main**.) In seguito si lanciano le centraline (classe **ReportSender**.)
- Infine, è possibile introdurre un numero di AppMobile arbitrario (classe **MobileAppInterface**.)
- A tale proposito consigliamo l'utilizzo del plugin [Multirun](#) per IntelliJ. Il plugin permette di lanciare con un solo click una serie di configurazioni diverse. Basta semplicemente creare le singole configurazioni e inserirle nella lista dell'unica configurazione Multirun. Il plugin si occuperà di lanciare in sequenza tutte le classi inserite.



Sistema centrale

Avvio

All'avvio del sistema, che avviene attraverso il *main()* presente nella classe **Main**, viene aperta la connessione RMI e viene reso disponibile uno stub della classe **SystemModuleConnector**, la quale implementa l'interfaccia **RemoteCentralSystemConnector**. Quest'ultima contiene i metodi necessari ai sotto-moduli (Centralina e AppMobile) per interagire con il sistema centrale, come ad esempio *sendReport()* e *RetrieveNotification()*.

Successivamente il sistema inizializza la mappa e i dati sulle strade. Queste informazioni vengono recuperate dalla cartella *resources* all'interno del progetto. Abbiamo incluso 4 diverse mappe in altrettante cartelle con nome *MapData[indice]*. La directory da cui recuperare i dati viene impostata all'interno della classe **MapDataSetup**, modificando opportunamente la variabile *mapDirectory*. La mappa consiste in una immagine .png denominata *map.png*. Il sistema processa l'immagine e restituisce una matrice dove ogni cella ha valore -1 se non esiste una strada in quel punto, oppure un valore intero che indica l'indice della strada corrispondente.

I dati sui nomi delle strade e la presenza delle centraline in esse sono recuperati dal file *streetData.txt*. Ogni riga contiene un primo carattere che indica la presenza o meno della centralina nella strada ('C' (centralina) ed 'E' (empty) rispettivamente), un separatore ' | ' ed il nome della strada.

Tutti i dati necessari al corretto funzionamento del sistema vengono salvati all'interno delle classi **MapManager** e **TrafficReportDatabase** all'avvio del sistema.

Una volta completati i settaggi preliminari viene lanciata l'interfaccia grafica.

Memorizzare informazioni di stato

D'ora in avanti il sistema si aggiorna ogni qual volta riceve una richiesta o un report da un sotto-modulo esterno. Esso provvederà a salvare opportunamente i dati sul traffico, aggiornando la variabile *savedTrafficData*, di tipo **StreetList**, all'interno della classe **TrafficReportDatabase**.

La classe **StreetList** contiene un'ArrayList di oggetti di tipo **Street**. Ogni variabile all'interno della lista contiene le informazioni sul traffico corrispondenti sotto forma di un numero intero. Nel caso la strada sia coperta da una centralina, la lista contiene un oggetto di tipo **StreetCovered** e la variabile *trafficAmount* corrisponde al valore ricevuto dalla centralina esterna corrispondente.

Nel caso la strada non sia coperta da una centralina, la lista contiene un oggetto di tipo **StreetEmpty**, con al suo interno una lista di **AppReport**. Ogni segnalazione ricevuta dall'AppMobile viene salvata all'interno di questa lista e il valore del *trafficAmount* corrisponde alle dimensioni della lista stessa.

La classe **TrafficReportDatabase** ha poi al suo interno un timer, sviluppato utilizzando la classe [Timer](#), che provvederà periodicamente ad aggiornare la lista di segnalazioni, eliminando eventuali segnalazioni scadute.

Invio notifiche ai sistemi esterni

L'invio delle notifiche avviene attraverso la classe **NotificationRetriever**.

Ogni AppMobile invierà periodicamente una richiesta di notifica al sistema centrale tramite RMI, segnalando la propria posizione attuale. Il sistema recupera la lista delle strade presenti in un intorno fisso dalla posizione dell'AppMobile e il corrispettivo dato di traffico attuale.

Successivamente questa lista viene filtrata eliminando tutte le strade con valore di traffico inferiore ad una data soglia. Infine, la lista di strade rimaste viene ritornata all'AppMobile che provvederà a visualizzare le notifiche ricevute.

Mostrare lo stato del sistema (Interfaccia grafica)

L'interfaccia grafica è sviluppata utilizzando la libreria [JavaFX](#).

Viene mostrata la mappa sulla sinistra dello schermo e i dati sul traffico sulla destra.

I dati sul traffico consistono in due liste:

- La lista (sulla destra) mostra il valore di traffico attuale per ciascuna *Street* ordinato a partire dalla strada più trafficata;
- La lista (sulla sinistra) mostra l'arrivo dei report dai vari sotto-moduli in tempo reale.

La mappa mostra tutte le strade e la posizione delle varie AppMobile.

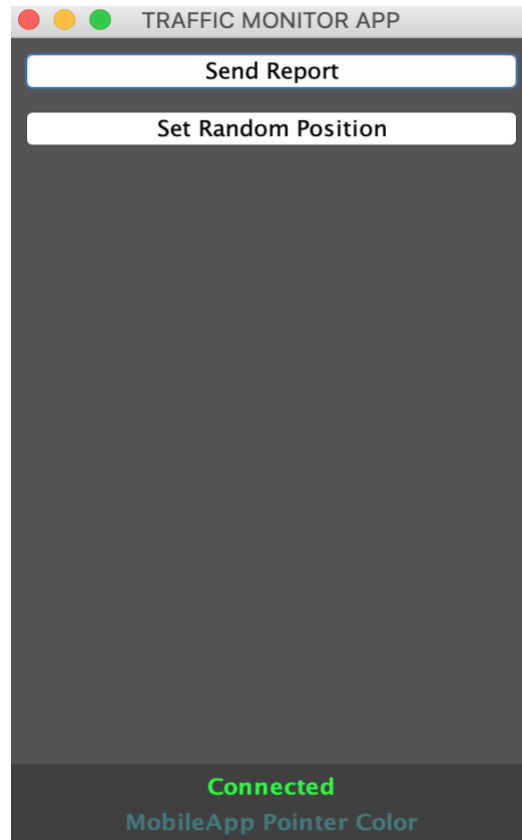
Le strade assumono un colore con gradazione scura proporzionale alla quantità di traffico presente.

È possibile riconoscere il nome di ogni strada tramite una label di testo in alto a sinistra della mappa. Questa si aggiorna ogni qual volta il mouse passa sopra una strada, indicandone il nome e il dato di traffico.

Le AppMobile possono essere riconosciute da degli indicatori colorati (ogni AppMobile ha un proprio colore specifico.) Questi indicatori vengono mostrati ogni qual volta un'applicazione interagisce con il sistema centrale (per una segnalazione o per il recupero delle notifiche.)



Applicazione mobile



Avvio interfaccia grafica

L'interfaccia grafica dell'applicazione mobile è stata sviluppata utilizzando la libreria grafica swing di Java all'interno della classe `MobileAppInterface`. Essa consiste in due `Button` e due `TextLabel`.

Il primo pulsante permette l'invio di una segnalazione di traffico al sistema centrale, mentre il secondo permette di modificare casualmente la propria posizione all'interno della mappa. Le due `TextLabel` permettono rispettivamente di segnalare la corretta connessione dell'applicazione al sistema centrale e il colore dell'indicatore sulla mappa all'interno dell'interfaccia del sistema centrale.

Invio segnalazioni di traffico

L'utente può segnalare una condizione di traffico nella propria posizione cliccando sul tasto *SendReport*. Viene inviata la posizione attuale dell'applicazione mobile al sistema centrale tramite RMI. Il sistema si occuperà di salvare opportunamente il report ricevuto. L'applicazione mobile deve aspettare un periodo di tempo preimpostato prima dell'invio di una seconda segnalazione, in modo da evitare lo spamming di report al sistema centrale.

Ricezione notifiche da sistema centrale

L'applicazione mobile implementa un timer all'interno della classe **UpdatingAppTimer**. La durata del timer è proporzionale alla velocità di movimento, calcolata all'interno della classe **MobileAppMovement**. Allo scadere del timer, l'applicazione si collega con il sistema centrale inviando la propria posizione attuale. Il compito del sistema è processare l'informazione ricevuta e ritornare una lista di strade in cui vi è traffico. Sfruttando i dati ricevuti dalle centraline e eventuali segnalazioni da altri utenti, salvati all'interno della classe **TrafficReportDatabase**, il sistema recupera una lista di strade con il corrispettivo traffico in un raggio fisso dalla posizione ricevuta, sotto forma di oggetti di tipo **StreetList**. L'applicazione mobile provvederà a mostrare un pop-up (*JOptionPanel*) con le varie notifiche ricevute.

Centralina

La divisione in classi all'interno della centralina rispecchia gli obiettivi che il modulo centralina deve raggiungere: la generazione dei report di traffico, la temporizzazione degli invii al sistema centrale e l'invio dei report vengono gestiti rispettivamente dalle classi **ReportCreator**, **TimerManager** e **CentralinaConnector**.

La classe **ReportSender** invece contiene solo metodi statici e fa da supporto alle altre tre, in particolare contiene il *main()* che si occupa di istanziare tutte le classi definendo anche l'id e quindi la strada sulla quale la centralina opera.

All'interno del **TimerManager** abbiamo il metodo *updateTimerCounter(int lastTrafficAmount)* che si occupa fra l'altro di aggiornare il tempo che deve trascorrere prima dell'invio del prossimo report: una semplice funzione definisce il valore in *ms* proporzionalmente all'ultima intensità di traffico rilevata nella via, più il traffico è intenso minore sarà l'attesa per la prossima rilevazione.

La temporizzazione è affidata ad un oggetto Timer in *backgroundTimerHandler()*.

Dentro a **ReportCreator** viene simulato l'andamento del traffico con un valore che viene casualmente incrementato o decrementato di un certo numero (anch'esso casuale) di unità.

Questa simulazione è affidata al metodo statico *retriveTrafficData* che viene richiamato da *trafficGenerator* anche qui attraverso un oggetto Timer. Questo è indipendente dal timer in **TimerManager**, cosa necessaria per evitare dipendenze fra l'istante di invio del report e quello di generazione (continua) del traffico.

CentralinaConnector apre la connessione con il sistema centrale in fase di creazione della centralina e contiene il metodo *retrieveTrafficInCentralina()*. Questo viene richiamato dal server centrale ogni qualvolta un'applicazione mobile invia una segnalazione di traffico da una strada in cui è presente anche una centralina. Viene così verificata la segnalazione fatta dall'utente.

Testing

Sono stati implementati 69 casi di test JUnit. Lo scopo è testare le funzionalità dei singoli metodi, appurando il loro corretto funzionamento sotto diverse configurazioni. Al momento della consegna, tutti i 69 test restituiscono un risultato positivo, come mostrato nell'immagine seguente. Ciò garantisce che i metodi e le classi testate funzionano come previsto.

