POLITECNICO
MILANO 1863

Corso di Ingegneria del Software

Prof. Marco Brambilla

Prova Finale

Design

Gruppo 2

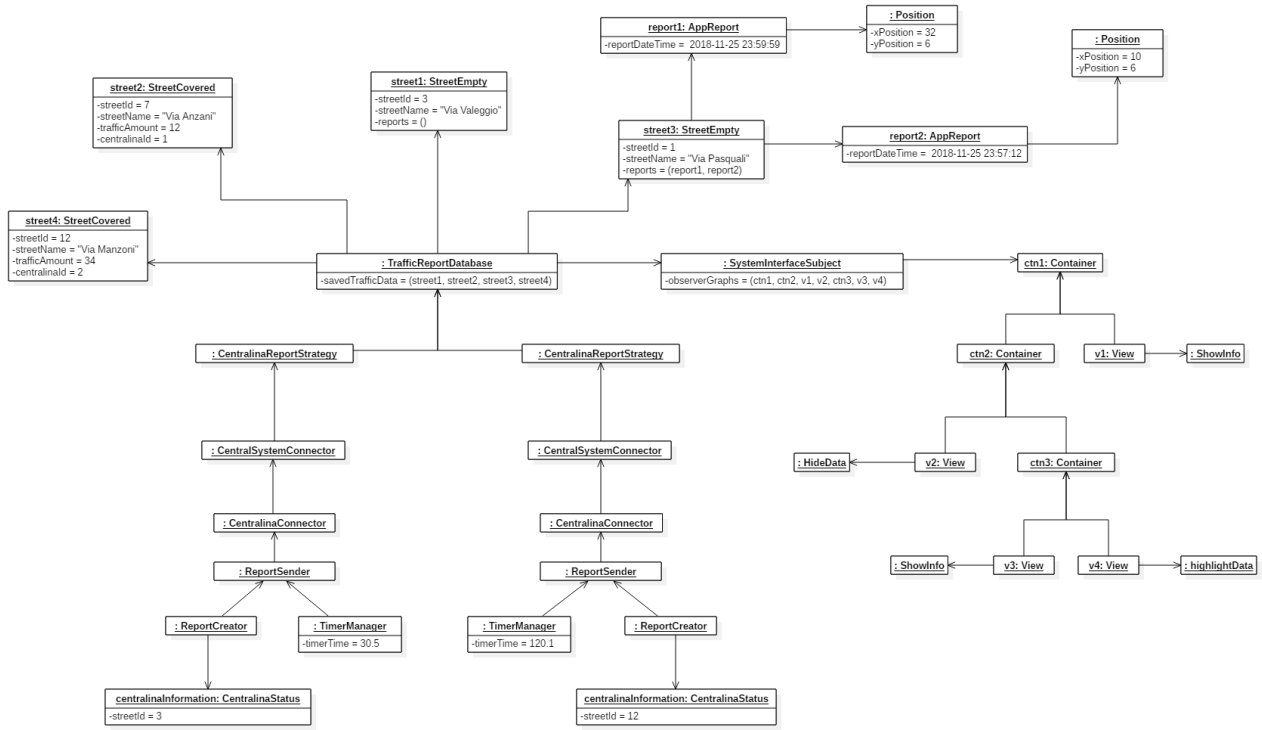
Cappelletti Andrea | Primerano Matteo | Maglione Sandro

Anno accademico 2018-2019

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.

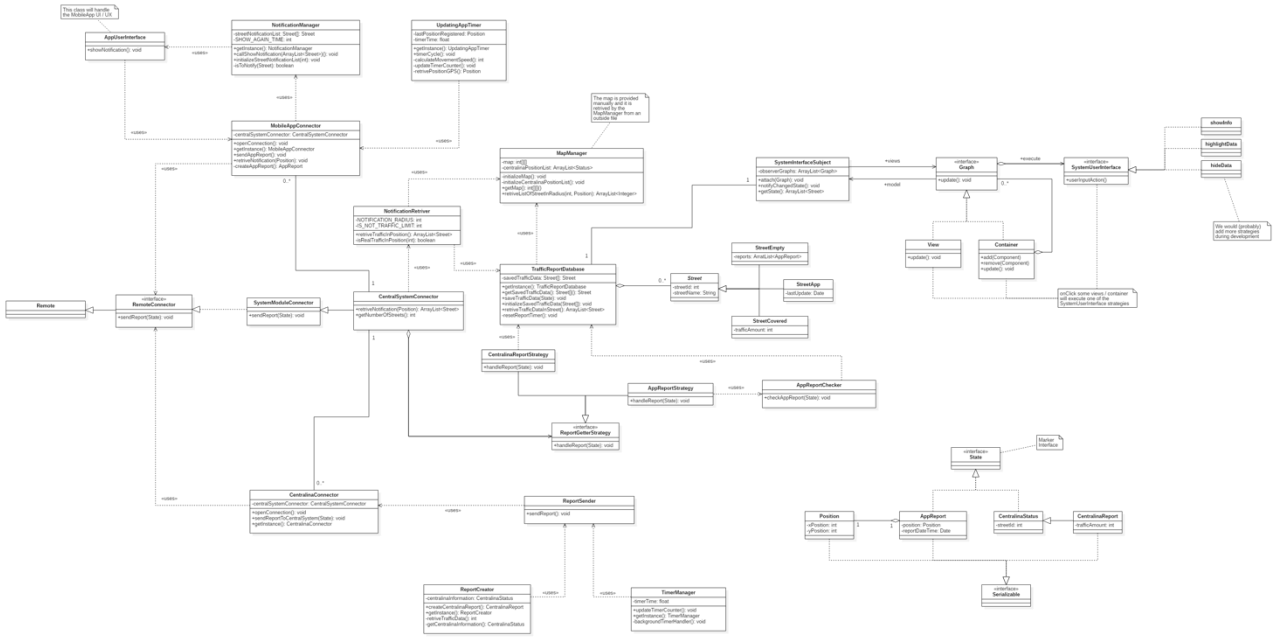
INDICE DEI CONTENUTI

Object Diagram



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 richiesto dalla classe `SystemInterfaceSubject` per essere mostrato sull'interfaccia utente (MVC Pattern).

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 `MobileAppConnector`, `UpdatingAppTimer`, `NotificationManager` e `AppUserInterface`.
- 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 façace 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. Anche l'applicazione mobile implementa un timer, che gestisce l'invio dell'aggiornamento della posizione GPS al sistema centrale.

Il sistema gestisce la ricezione del dato di traffico implementando uno Stretegy Pattern. Quando il sistema centrale riceve una segnalazione dall'utente, l'oggetto inviato `AppReport` passa attraverso la classe `AppReportChecker` che implementa il Proxy Pattern, prima di salvare il dato all'interno del sistema.

Tutti i dati sul flusso di traffico vengono memorizzati all'interno della classe `TrafficReportDatabase` come oggetti *`Street`*. In particolare, viene salvato un oggetto `StreetEmpty` nel caso la relativa strada non si coperta da una centralina, e `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. 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.

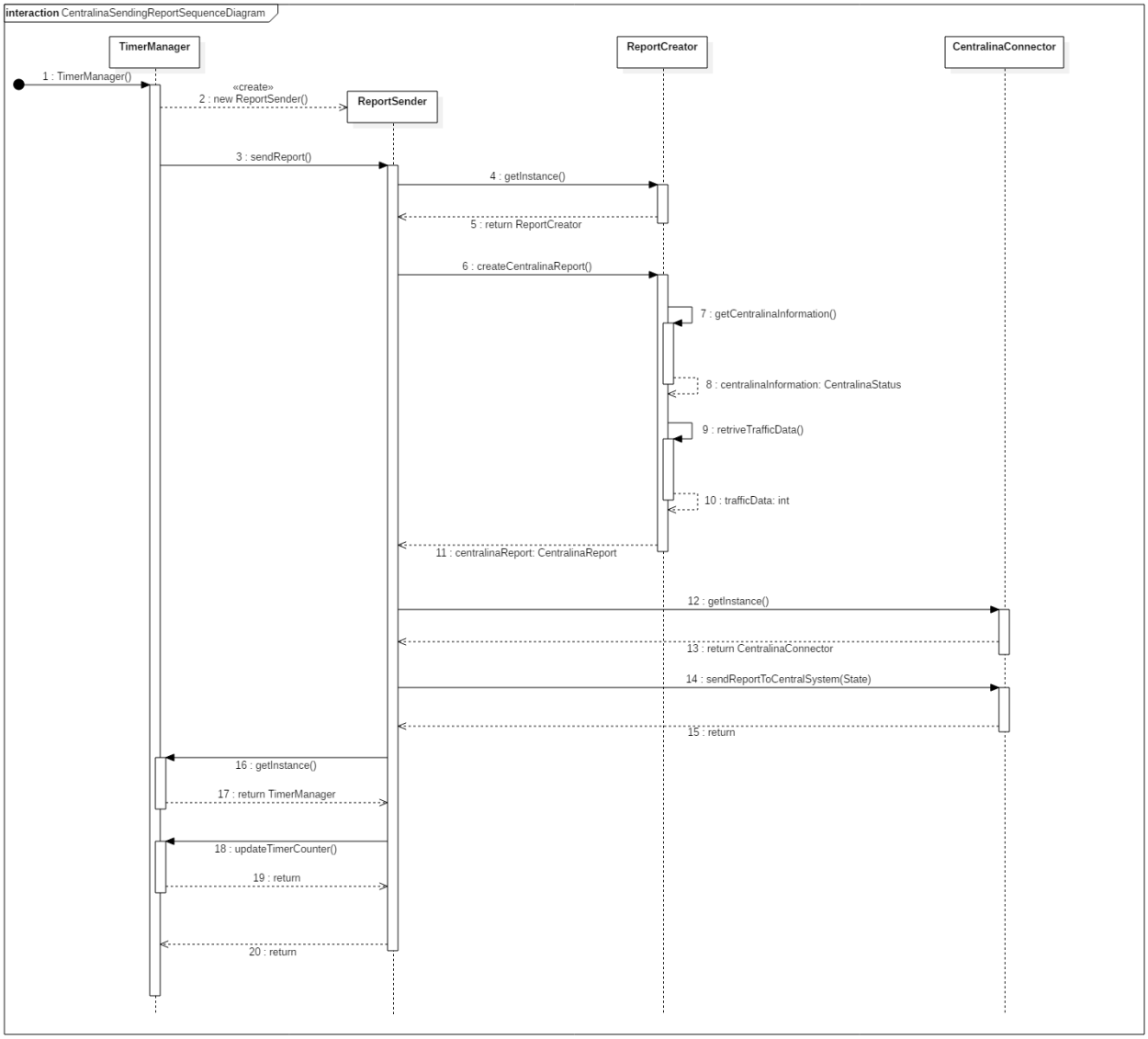
L'interfaccia utente all'interno del sistema centrale implementa l'MVC Pattern, composto dai pattern Observer, Composite e Stretegy.

- Il Model del pattern è la classe `SystemInterfaceSubject` che svolge il ruolo di Subject nell'observer pattern.
- Le View sono degli oggetti di tipo `Graph`: questi sono gli Observers nell'observer pattern e implementano il Composite Pattern.
- Il Controller è l'interface `SystemUserInterface` la quale implementa lo Strategy Pattern, svolgendo diverse funzionalità a seguito dell'interazione con l'utente.

Gli oggetti di tipo State implementano il Marker Interface Pattern.
Essendo questi utilizzati e scambiati da molte classi all'interno dell'intero sistema sono stati omessi gli espliciti collegamenti, per evitare di compromettere la chiarezza grafica del diagramma.

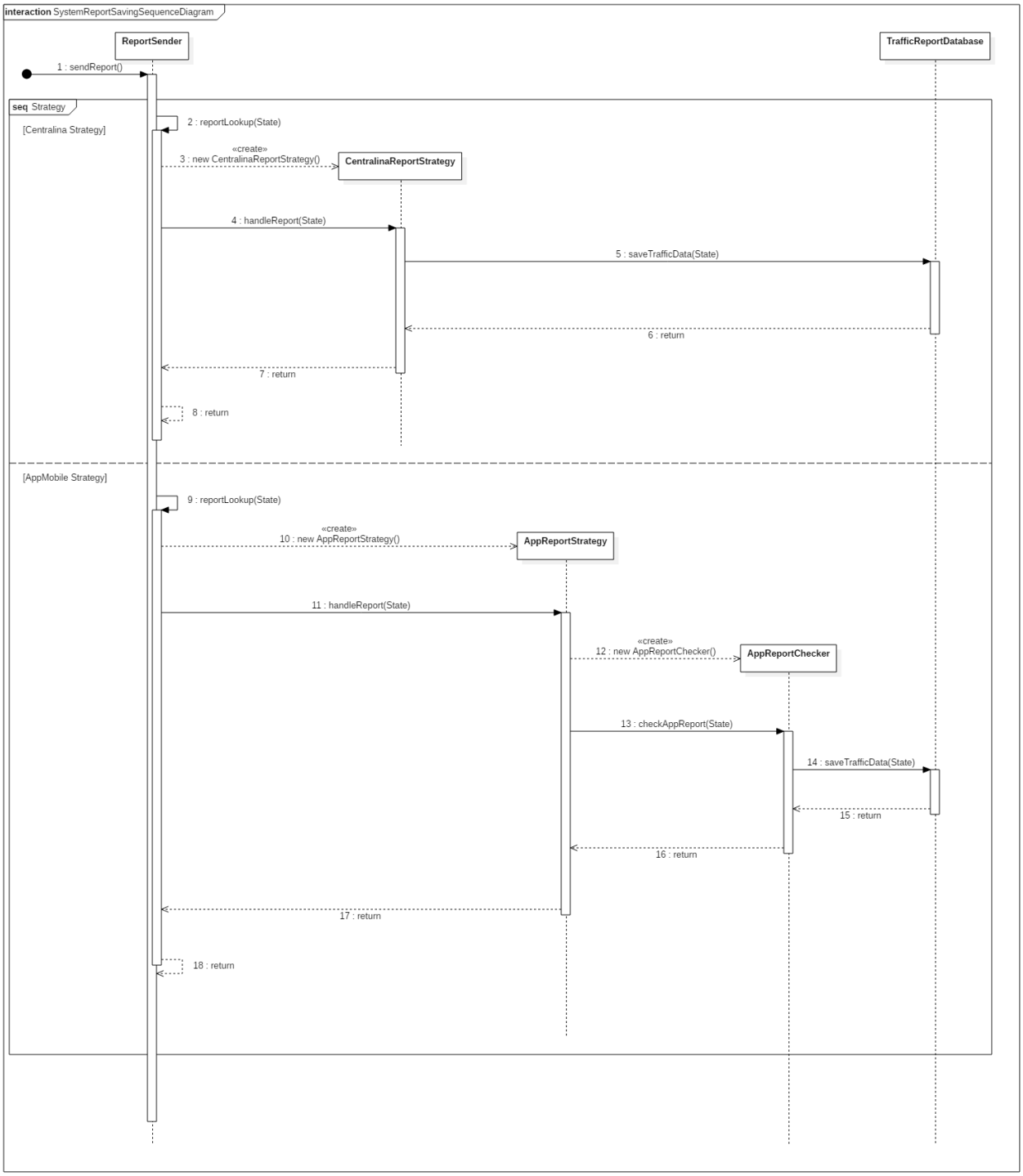
Le classi che svolgono funzioni in comune (di tipo Manager) sono state rese Singleton.

1 | Sequence Diagram



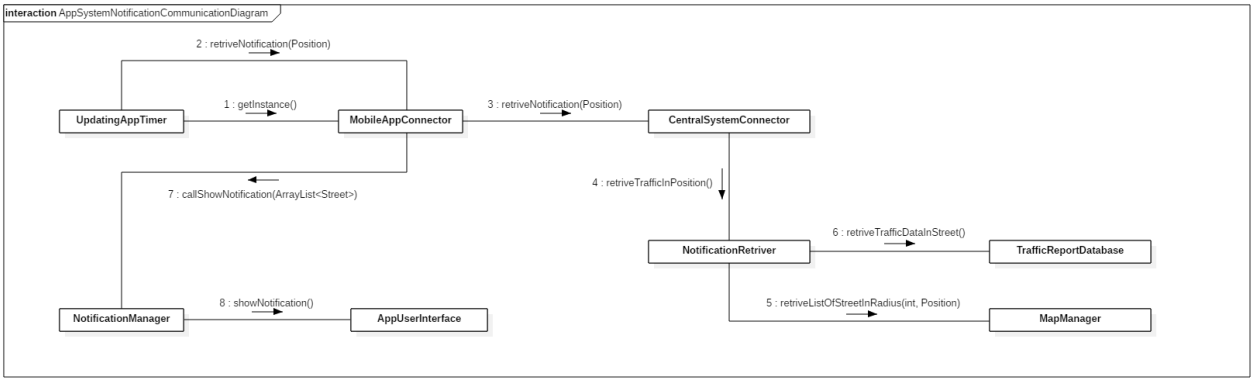
Questo *Sequence Diagram* descrive il processo di invio del dato sul traffico da parte della centralina al sistema centrale. Allo scadere del timer interno alla centralina viene rilevato il dato di traffico tramite il sensore e viene costruito un oggetto `CentralinaReport`. Questo viene poi inoltrato alla classe `CentralinaConnector` che invia il dato al sistema centrale. Infine, viene resettato il timer in base al flusso di traffico rilevato.

2 | Sequence Diagram



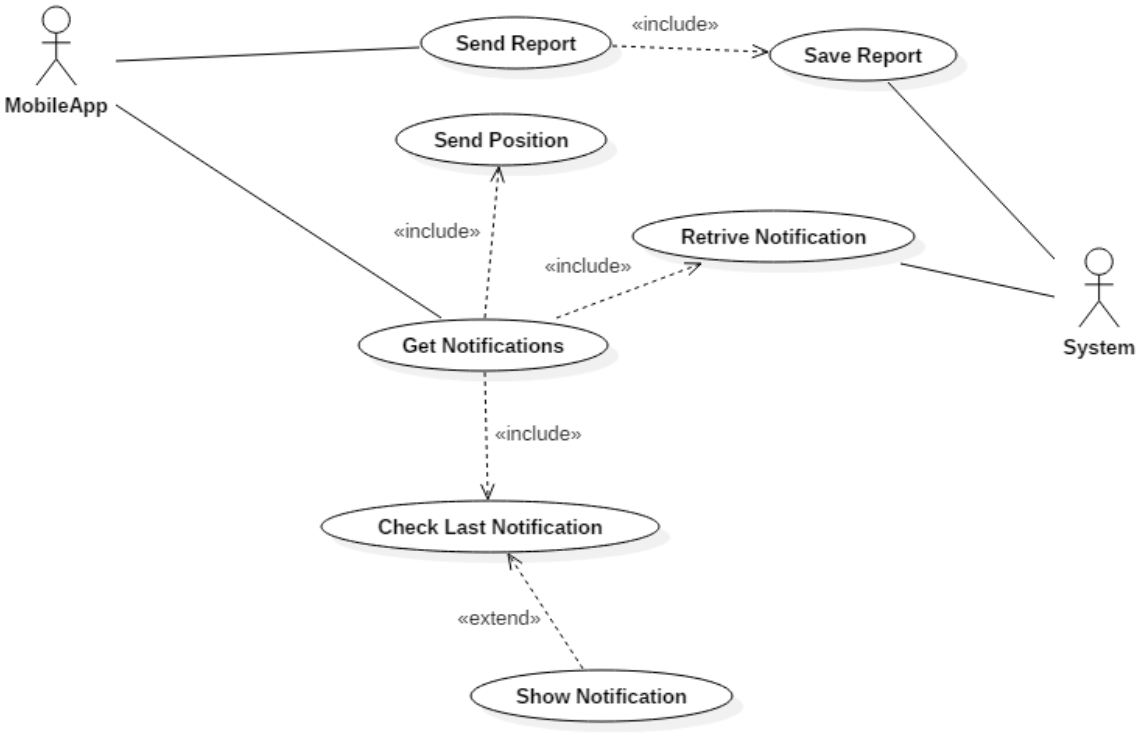
Questo *Sequence Diagram* descrive il processo di salvataggio di un report da parte del sistema centrale. Viene identificata la sorgente del dato e avvengono due processi diversi nel caso in cui il sistema riceva il dato dalla centralina o dall'applicazione mobile (Strategy Pattern). Il dato in arrivo da una centralina viene salvato direttamente all'interno della classe `TrafficReportDatabase`. La segnalazione dell'utente viene invece filtrata attraverso la classe `AppReportChecker` (Proxy Pattern). Il dato viene poi inoltrato e salvato all'interno della classe `TrafficReportDatabase`.

Collaboration Diagram



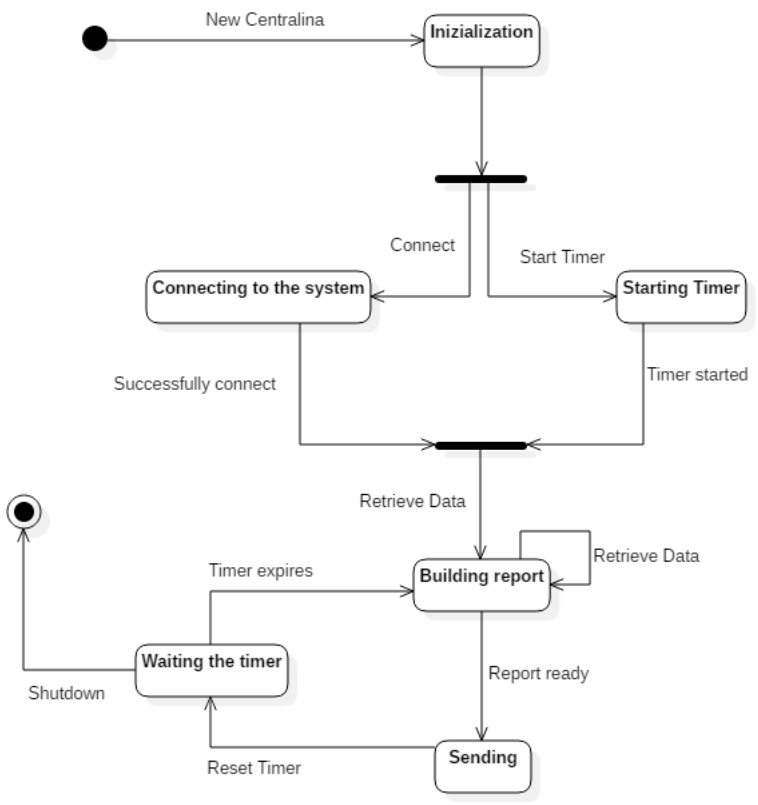
Questo *Collaboration Diagram* mostra l'interazione tra le classi dell'applicazione mobile e del sistema centrale nel contesto dell'invio di una notifica. L'applicazione mobile invia la propria posizione GPS al sistema centrale. Il sistema si occupa di recuperare il dato sul traffico in un raggio fisso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

Use case Diagram



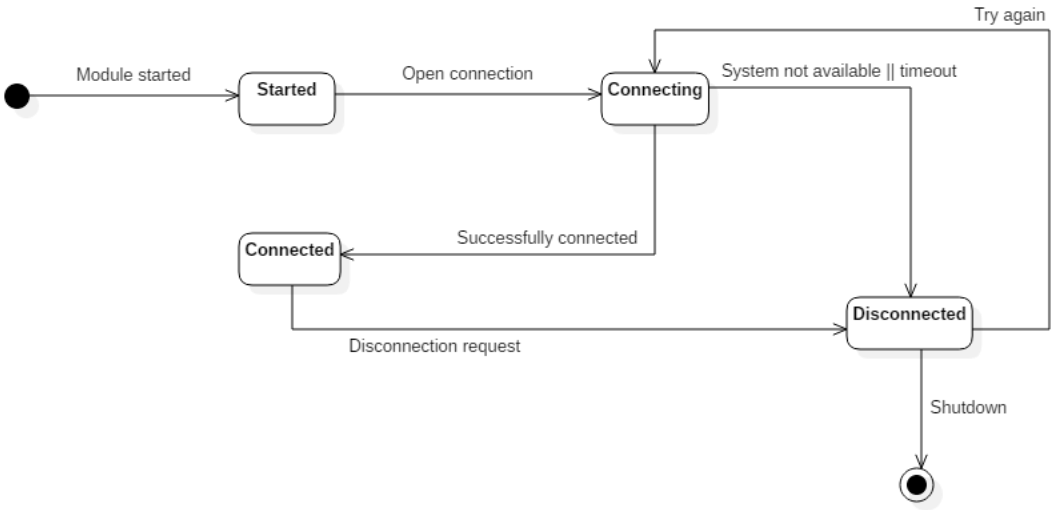
Questo *Use Case* propone i casi d'uso dell'applicazione mobile, ricavabili direttamente dal diagramma i* dei requisiti con ulteriori estensioni. In particolare, l'applicazione mobile può inviare una segnalazione al sistema e ricevere notifiche tramite un algoritmo che controlla quando è stata ricevuta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

1 | Statechart Diagram



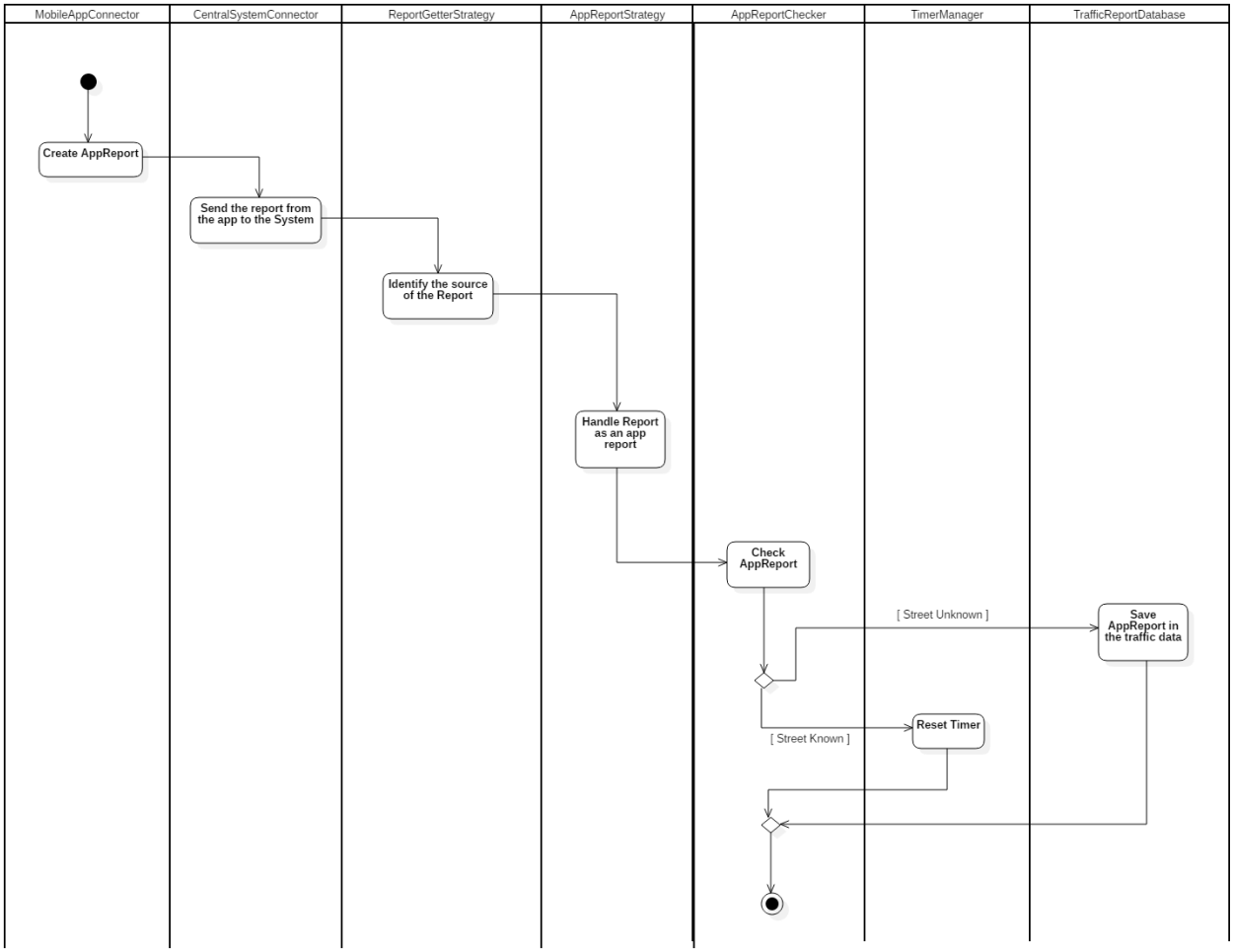
Questo *Statechart Diagram* modella le varie fasi di una centralina (dall'inizializzazione al suo utilizzo costante).

2 | Statechart Diagram



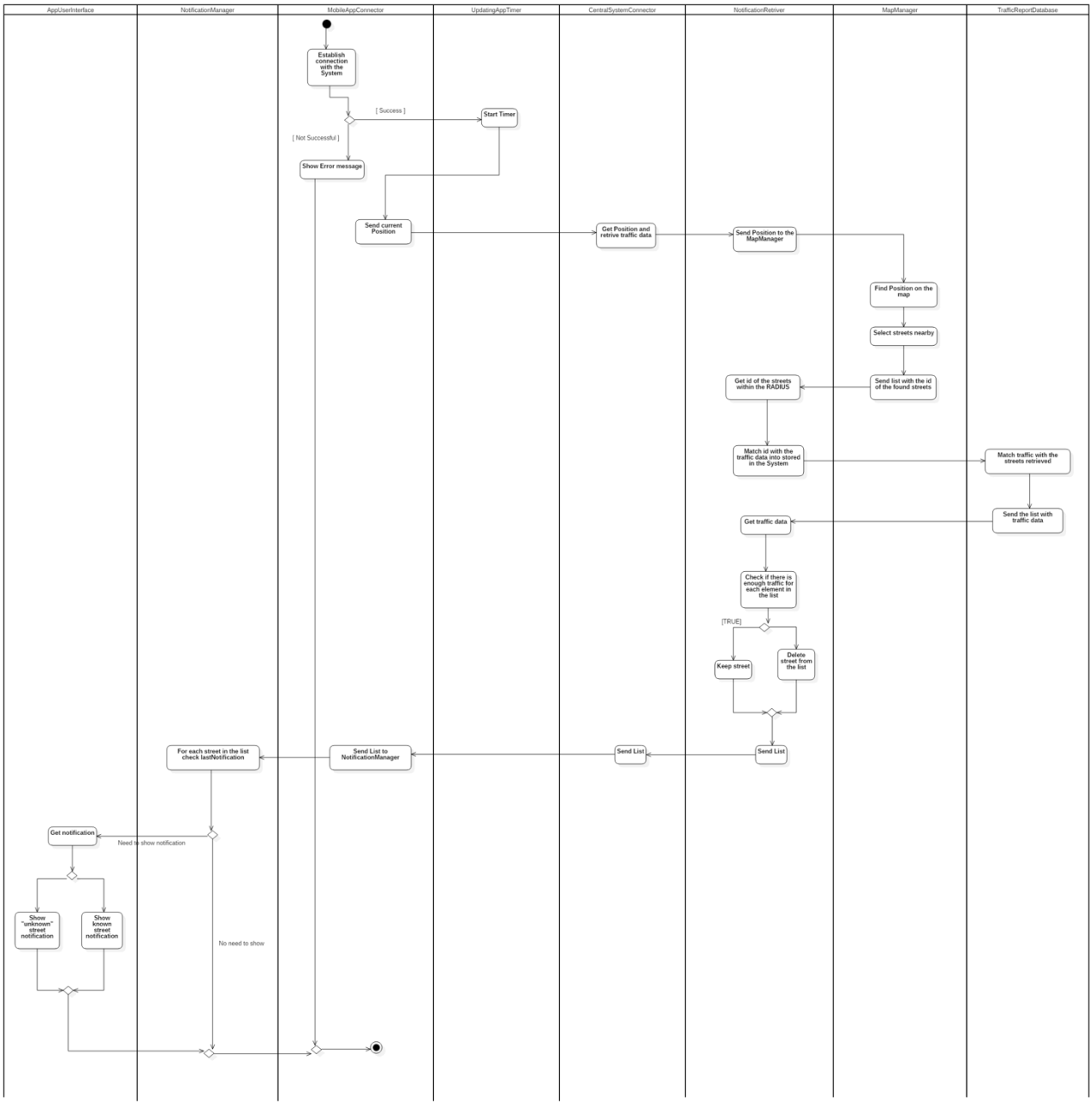
Questo *Statechart Diagram* modella la fase di connessione di tutti i moduli (Centralina e Applicazione Mobile) al sistema centrale tramite RMI.

1 | Activity Diagram



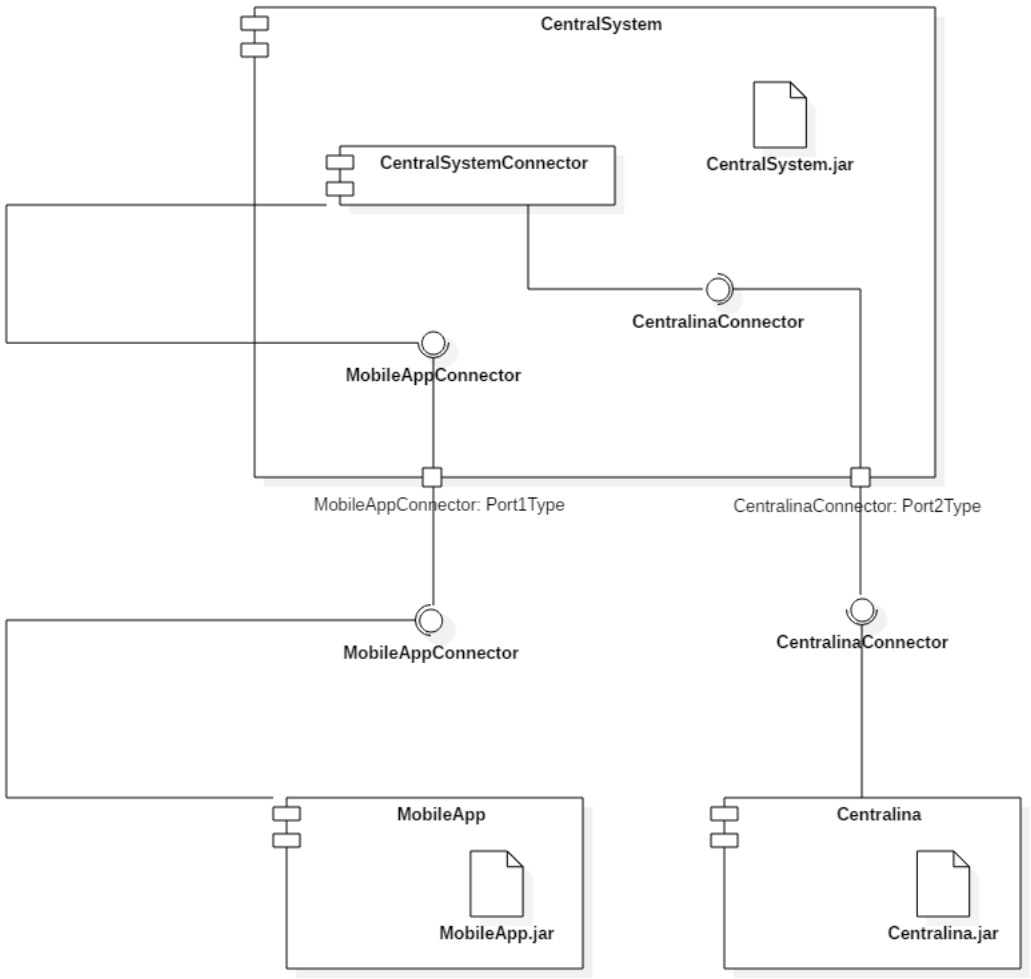
Questo *Activity Diagram* descrive la segnalazione di un report da parte dell'applicazione mobile. Il `ReportGetterStrategy` è in comune fra centralina e applicazione. Quest'ultimo utilizza lo strategy pattern e applica due algoritmi differenti a seconda se il report è stato inviato da una centralina o da un'applicazione mobile. In questo caso è proveniente dall'applicazione e se la segnalazione proviene da una strada nella quale è presente una centralina (strada "known") viene azzerato il timer di quest'ultima permettendo così l'aggiornamento del trafficAmount. Se invece proviene da una strada nella quale non vi è una centralina (strada "unknown") viene salvato nella lista di report presente nella streetUnknown.

2 | Activity Diagram



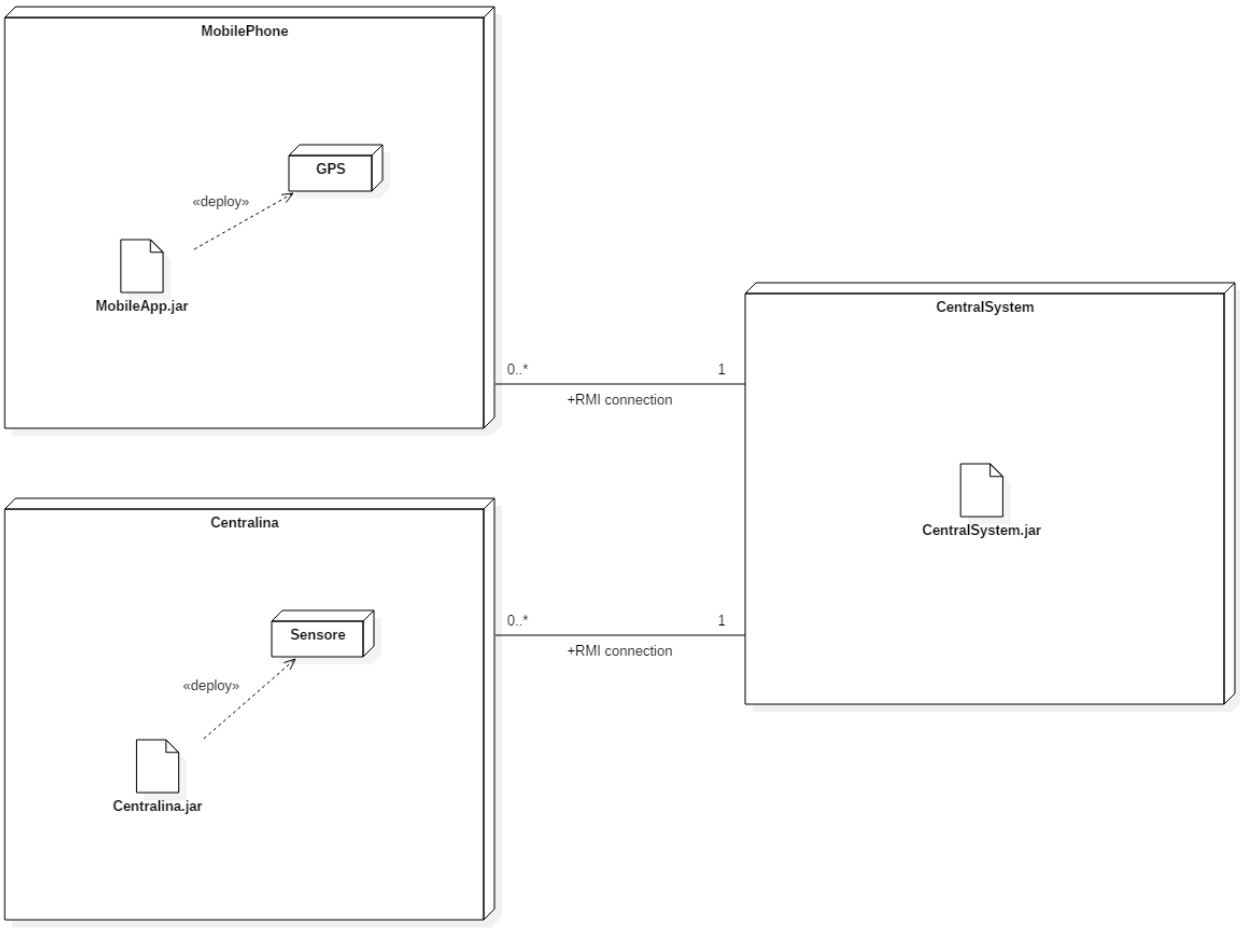
Questo *Activity Diagram* descrive l'invio della posizione da parte di un'applicazione mobile. Una volta identificato in che strada si trova la posizione inviata, tenendo conto delle strade che sono comprese nel raggio della posizione, vengono richiesti i dati di traffico per queste ultime. Si passa in seguito al controllo per decidere se c'è sufficiente traffico da segnalarlo. In seguito, si restituisce la lista al `NotificationManager`, il quale, tramite un opportuno controllo, deciderà se far visualizzare la notifica o meno. Quest'ultima differisce per i due tipi di strada (known e unknown).

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.

Deployment Diagram



Questo *Deployment Diagram* mostra la distribuzione dei file eseguibili e dei componenti fisici coinvolti nell'interazione tra sistema centrale e sotto-moduli.