#swEngComo

Traffic Monitor

A.A. 2018/2019

POLITECNICO
MILANO 1863

Altieri Matteo	869296
Genesi Lorenzo	868898
Marforio Simone	867995

Indice

3

Struttura sistema

Componenti

Per ogni componente è stato creato un progetto apposito.

- Sistema centrale: si occupa di elaborare tutte le informazioni ricevute dai sistemi esterni e della comunicazione con essi
- Centralina stradale: si occupa del monitoraggio del segmento a cui è associata, elaborando un indice di traffico
- Applicazione: si occupa di comunicare all'utente le notifiche contenenti gli aggiornamenti sul traffico e permette di inviare segnalazioni sullo stato del traffico
- Interfaccia utente: permette di avere una visione generale sullo stato del sistema, attraverso una mappa che mostra i segmenti stradali, una lista delle centraline ed il loro stato di funzionamento e uno storico di tutti i dati ricevuti (dati centraline e segnalazioni)

Metodo di comunicazione

RMI

Database

SQLite

IDE

IntelliJ Idea 2018.3

Risorse esterne

Abbiamo utilizzato delle librerie e delle API esterne, le cui dipendenze sono già incluse nel file ".iml" di configurazione di ogni progetto.

- Testing JUnit: junit-4.12.jar, hamcrest-core-1.3.jar - su ogni componente
- Libreria database: lib\sqlite-jdbc-3.23.1.jar - su sistema centrale
- File database: db\traffic_monitor.db - su sistema centrale
- API Google maps: tramite script Javascript - su interfaccia utente

4

Funzionamento

Per documentare il funzionamento del progetto, prendiamo in esame ogni sua funzionalità e mostriamo come viene svolta all'interno dell'intero sistema, passando tra i componenti, le classi e i metodi.

Di fatto, le funzionalità sono:

- Monitoraggio del traffico con centraline stradali, con cambio della periodicità di invio dei dati
- Monitoraggio del traffico con segnalazioni utente
- Avviso cambiamento situazione traffico per gli utenti interessati tramite notifica
- Monitoraggio dello stato di funzionamento delle centraline e diagnostica
- Visualizzazione mappa con stato di traffico di ogni segmento, stato di funzionamento centraline e storico dati e segnalazioni

Progetto centralina stradale

Il progetto della centralina stradale utilizza un'interfaccia grafica per vedere in tempo reale il suo funzionamento e che permette volendo di inserire manualmente i dati di un rilevamento. La grafica viene gestita attraverso un pattern MVC dove:

- Model: CentralinaStradale.java
- View: CentralinaStradaleGUI.java
- Controller: ControllerCentralinaStradale.java

Il funzionamento della centralina è scandito dal timer *timer_periodoRilevazione()* che si occupa, allo scadere del valore di tempo indicato in *periodicita* di lanciare una diagnostica (il cui funzionamento sarà spiegato successivamente nello stato di funzionamento delle centraline), di effettuare la rilevazione (*effettuaRilevazione()*) e infine di inviare la rilevazione effettuata (*inviaRilevazione()*).

Il metodo *effettuaRilevazione()* ha due casi di funzionamento: quello automatico, che chiama *simulazioneTraffico()* o quello manuale, dove vengono presi i dati forniti nella GUI.

simulazioneTraffico() simula in modo pseudocasuale un traffico che aumenta sempre di più con il passare del tempo.

Ognuno dei due casi di funzionamento va a modificare gli attributi *numeroMacchine* e *velocitaMedia*, che serviranno per il calcolo dell'indice di traffico.

Dopo aver effettuato la rilevazione, come detto precedentemente, viene chiamato il metodo *inviaRilevazione()*. Questo calcola il nuovo indice di traffico con *calcolaIndice()*, basandosi sulle informazioni rilevate e quelle relative al segmento monitorato. L'indice di flusso viene calcolato come il rapporto tra la velocità media (*velocitaMedia*) registrata sul segmento e la velocità massima (*velMax*) della strada. Questo valore viene espresso in percentuale e trasformato nel suo reciproco (100-valoreCalcolato) per indicare il livello di traffico. Viene quindi salvato nell'attributo *indiceDiFlusso*.

Basandosi su questo dato, viene aggiornata la periodicità di invio del prossimo dato (*calcolaPeriodicita()*, sapendo che se il livello di traffico sta aumentando, il tempo di rilevazione e invio dovrà diminuire). Viene infine creato un *DatoCentralinaStradale* che sarà inviato tramite RMI al sistema centrale, precisamente al *GestoreCentralina*.

A questo punto, vengono resettati i valori utili alla rilevazione e viene fatto ripartire il timer con la

nuova periodicità (*reset()* e *startTimer()*).

Progetto sistema-centrale

Il sistema centrale riceve il dato della centralina stradale con RMI con il metodo *riceviDatoCentralina(DatoCentralina dato).* Notare che questa funzione sarebbe la stessa utilizzata dalle centraline automobilistiche non implementate, motivo per cui è necessario eseguire un cast da *DatoCentralina* a *DatoCentralinaStradale*. In seguito vengono salvate nel *GestoreCentraline* le informazioni relative all'ultimo invio e l'attesa massima (*aggiornaUltimoInvioCentralina(DatoCentralinaStradale datoStradale)*, *aggiornaAttesaMassima(DatoCentralinaStradale datoStradale)*) e viene invocata la funzione *diagnostica()* che verrà illustrata in seguito. Il dato viene inoltrato (*inoltraDatoCentralina(DatoCentralinaStradale dato)*) al *GestoreDato* che si occuperà di salvarlo nel database (*addDatoCentralina(DatoCentralina dato)*) e poi di elaborarlo (*elaboraDatoCentralina(DatoCentralinaStradale dato)*).

Il dato che arriva da una centralina stradale è già pronto con un indice di flusso associato al segmento, quindi una volta recuperato il segmento a cui si riferisce (*getSegmentoDaCoord(Coordinate c)*), si passa al *GestoreSegmentiStradali* con il metodo *aggiornaIndiceDiFlusso(int indice, SegmentoStradale s)*.

Quest'ultimo metodo richiama il *valutaEvento(int indice, SegmentoStradale s)* il cui funzionamento sarà spiegato successivamente nell'invio della notifica. Si controlla quindi se il segmento è monitorato da centralina (l'aggiornamento dell'indice può derivare anche dalle segnalazioni degli utenti, che hanno un valore temporale dal momento che non è possibile garantire la loro costanza), e si aggiorna l'indice sia sulla *listaSegmentiStradali* che sul databse con *updateIndiceDiFlussoSegmentoStradale(String codSegmento, int index)*.

Progetto applicazione

Il progetto dell'applicazione utilizza un'interfaccia grafica che permette di inviare le segnalazioni. La grafica viene gestita attraverso un pattern MVC dove:

- Model: Applicazione.java
- View: ApplicazioneGUI.java
- Controller: ControllerApplicazione.java

Al momento della creazione di un'applicazione viene settato l'RMI in entrata, necessario per la ricezione delle notifiche, comunicando al sistema centrale l'indirizzo IP e il numero di porta su cui sarà in ascolto l'applicazione.

Per inviare una segnalazione, sulla GUI è possibile scegliere uno dei 3 livelli di traffico disponibili (La strada presenta molti rallentamenti - La strada ha qualche rallentamento - La strada è libera) e utilizzare l'apposito bottone. E' possibile inserire manualmente la posizione (latitudine e longitudine).

Alla pressione del bottone, vengono prelevati i dati dalla GUI, viene creata una nuova *SegnalazioneUtente* e attraverso il metodo *inviaSegnalazione(SegnalazioneUtente segnalazione)* l'oggetto viene inviato tramite RMI al sistema centrale, precisamente al *GestoreDato*. Durante la creazione della *SegnalazioneUtente* ad ogni tipo di segnalazione viene associato un indice di traffico calcolato come valore medio della fascia di appartenenza (15 - 50 - 85).

Progetto sistema-centrale

Il *GestoreDato* riceve la segnalazione tramite RMI attraverso il metodo *riceviSegnalazione(SegnalazioneUtente segnalazione)*. Si occupa quindi di salvarla nel database (*addSegnalazioneUtente(SegnalazioneUtente segnalazione)*) e di chiamare il metodo per elaborarla (*elaboraDatoSegnalazione(SegnalazioneUtente dato)*).

Per elaborare una segnalazione ci si appoggia su una struttura dinamica composta da una lista di liste, rappresentata dall'attributo *listaSegnalazioni*. Su questa struttura sono salvate tutte le segnalazioni ricevute, raccogliendole per riga in base al segmento a cui sono associate.

ES: (dove s1, s2, sono le segnalazioni)
Segmento 1 Via Anzani -> s1 s4 s5 .
Segmento 2 Via Valleggio -> s2 .
Segmento 3 Via dei Mille -> s3 s6 .

La prima operazione è quella di associare la segnalazione ad un segmento (*getSegmentoDaCoord(Coordinate coord)*). La segnalazione viene elaborata solo se è relativa ad un segmento non monitorato da una centralina, altrimenti fornirebbe solo informazioni superflue.

Si controlla quindi scorrendo la struttura se esistono già segnalazioni associate al segmento relativo alla nuova segnalazione. Se sì, allora si aggiunge la segnalazione alla riga, se no si aggiunge una nuova riga alla struttura. Ogni segnalazione ha una validità temporale di *timer_validitaSegnalazione* secondi, dopo la quale viene eliminata con il timer *timerSegnalazione(SegnalazioneUtente s)*. Se su una riga si hanno almeno *n_segnalazioni* allora si procede a calcolare un indice medio tra tutte le segnalazioni relative al segmento a cui sono associate, e si aggiorna il suo indice con *aggiornaIndiceDiFlusso(int indice, SegmentoStradale s)*. In questo modo si evita di fare affidamento su poche segnalazioni che potrebbero essere imprecise.

Il funzionamento del metodo *aggiornaIndiceDiFlusso()*, chiamato sul *GestoreSegmentoStradale*, è lo stesso del caso precedente con il dato della centralina, con l'accorgimento che, dal momento in cui il segmento non è monitorato da una centralina, l'informazione riguardo il suo indice di flusso ha una validità temporale indicata dall'attributo *timer_decadimentoIndice*.

Tutto parte dopo un aggiornamento di indice di flusso relativo ad un segmento, quando viene chiamata il metodo *valutaEvento(int nuovoIndice, SegmentoStradale segmento)* nel GestoreSegmentoStradale. Qui viene fatto un confronto tra l'indice di flusso precedente, l'indice di flusso attuale e il nuovo indice di flusso del segmento. Ogni indice viene associato ad una fascia di traffico:

- fascia 1: 0 - 30
- fascia 2: 31 - 70
- fascia 3: 71 - 100

Se dall'indice precedente a quello attuale si è passati ad una fascia superiore, e questo cambiamento viene confermato con il nuovo indice, allora si chiama il metodo *generaNotificaDaEvento(SegmentoStradale segmento, int indiceDiFlusso)* del *GestoreNotifica*. In questo modo si evita di generare delle notifiche ad ogni minimo cambiamento anche temporaneo dello stato di traffico.

Nel *GestoreNotifica* viene quindi creata una nuova notifica con una descrizione dello stato di traffico basata sull'indice di flusso, e viene chiamato il metodo *inviaNotifica(Notifica notifica, SegmentoStradale segmento)*.

Qui viene preso il punto medio del segmento e si richiede al *GestoreUtente* quali sono gli utenti che cadono nel raggio di notifica con *controllaUtentiNelRaggio(Coordinate centro)*. Un utente cade nel raggio di notifica se la differenza tra il centro del segmento su cui è avvenuto l'evento e la sua posizione è minore del raggio di notifica. In questo momento quindi il *GestoreUtente* con *richiediPosizioneUtente()* chiede a tutte le applicazioni tramite RMI di inviare la posizione attuale dell'utente che sta utilizzando l'applicazione.

La posizione viene ricevuta dal *GestoreDato* attraverso il metodo chiamato con RMI *riceviPosizionUtente(DatoPosizione dato)*, che salva nel database la nuova posizione dell'utente, e inoltra subito il dato anche al *GestoreUtente* con *aggiornaPosizioneUtente(String username, Coordinate nuovaPosizione)*.

Dal *GestoreUtente* si hanno quindi i dati aggiornati per verificare quali utenti cadono nel raggio di notifica, e si può tornare al *GestoreNotifica*, dove si completa l'invio delle notifiche agli utenti interessati tramite RMI, chiamando il metodo remoto *riceviNotifica(Notifica notifica)*.

Progetto applicazione

In *Applicazione* il metodo appena chiamato tramite RMI con il *mostraNotifica(Notifica notifica)* fa vedere nella GUI il contenuto della notifica.

Progetto centralina stradale

Nella centralina stradale sono presenti due attributi di tipo booleano: *attivo* e *funzionante*. L'attributo funzionante può essere modificato manualmente attraverso la GUI e simula un malfunzionamento della centralina. In modo automatico, ogni volta che termina il periodo di rilevazione, viene invocato il metodo *diagnostica()* che nel caso di guasto modifica il valore dell'attributo *attivo* e blocca il funzionamento della centralina, smettendo di rilevare e inviare i dati di traffico.

Progetto sistema-centrale

Anche il sistema centrale prevede un metodo di *diagnostica()* delle centraline, che permette di tenere sotto controllo il loro funzionamento. Questo metodo viene invocato periodicamente e, scorrendo l'intera lista delle centraline, controlla se è già passato il periodo di attesa massima che era stato fornito dall'ultimo dato ricevuto. Nel caso in cui il tempo sia scaduto il *GestoreDato* interpreta questo ritardo come un problema della centralina e la segna come non funzionante attraverso il metodo *segnalaCentralinaGuasta(int index)*. Infine riporta questa modifica anche sul database.

Progetto interfaccia-utente

Il progetto dell'interfaccia utente utilizza un'interfaccia grafica che permette di visualizzare tutte le informazioni richieste. La grafica viene gestita attraverso un pattern MVC dove:

- Model: InterfacciaUtente.java
- View: InterfacciaUtenteGUI.java
- Controller: ControllerInterfacciaUtente.java

All'avvio, tramite RMI l'*InterfacciaUtente* richiede i seguenti dati al sistema centrale:

- con *mostraStorico()* chiama *inviaStorico()* al *GestoreDato*
- con *mostraStatoCentraline()* chiama *inviaStatoCentralina()* al *GestoreCentralina*
- con *mostraMappa* chiama *inviaSegmentiStradali()* al *GestoreSegmentoStradale* e *inviaStatoCentralina()* al *GestoreCentralina*, utilizzati per a mostrare lo stato di traffico e la posizione delle centraline sulla mappa

Sul sistema centrale ognuno di questi metodi non fa altro che ritornare una lista di dati:

- Il *GestoreDato* richiede al database tutti i dati delle centraline e delle segnalazioni (*getDatiCentralina(), getSegnalazioni()*)
- Il *GestoreCentralina* ritorna la lista delle centraline (*listaCentraline*) in suo possesso
- Il *GestoreSegmentoStradale* ritorna la lista dei segmenti (*listaSegmentiStradali*) in suo possesso

mostraStorico() dopo aver ricevuto i dati, differenzia i dati delle centraline e delle segnalazioni, ne esegue il cast e aggiorna la view.

mostraStatoCentraline() procede allo stesso modo con le informazioni relative alle centraline.

mostraMappa() invece si appoggia sulla classe *Code*. Questa contiene uno script in Javascript, a cui vengono aggiunti i dati appena ricevuti. Con il metodo *getCodice()* si ottiene quindi lo script che utilizza le API di Google Maps e che contiene le informazioni dei segmenti e delle centraline da mostrare. Di fatto viene generata una mappa che evidenzia lo stato di traffico dei segmenti e le posizioni delle centraline sulle strade. Lo script verrà eseguito dal browser integrato di Java, a cui viene riservato un JPanel nella GUI.