CONSEGNA 2 – “SMART GARAGE”

RELAZIONE

di Lorenzo Campanelli, Luca Ragazzi, Mattia Leonessi

La realizzazione dell’assignment “Smart Garage” è basata su un’architettura con scheduler cooperativo e task sincroni (macchine a stati finiti time triggered).

In particolare, si è deciso di utilizzare un’implementazione dello scheduler (fornita dal prof. a lezione) che mandasse in power saving il micro in attesa del prossimo tick del timer, al fine di risparmiare energia (in un’ottica di alimentazione a batteria).

In fase di progettazione sono state individuate 2 funzionalità principali: la porta del garage (GarageDoorTask) e il sistema di assistenza al parcheggio (ParkingAssistanceTask).

In più è emersa la necessità di un task (CommandReaderTask) che si occupasse di gestire i messaggi in arrivo sulla seriale e che chiamasse gli opportuni metodi degli altri task.

Al centro dell’interazione tra i vari task c’è l’oggetto SharedContext che raggruppa variabili condivise tra i task e metodi per il setting e la verifica del valore di tali variabili.

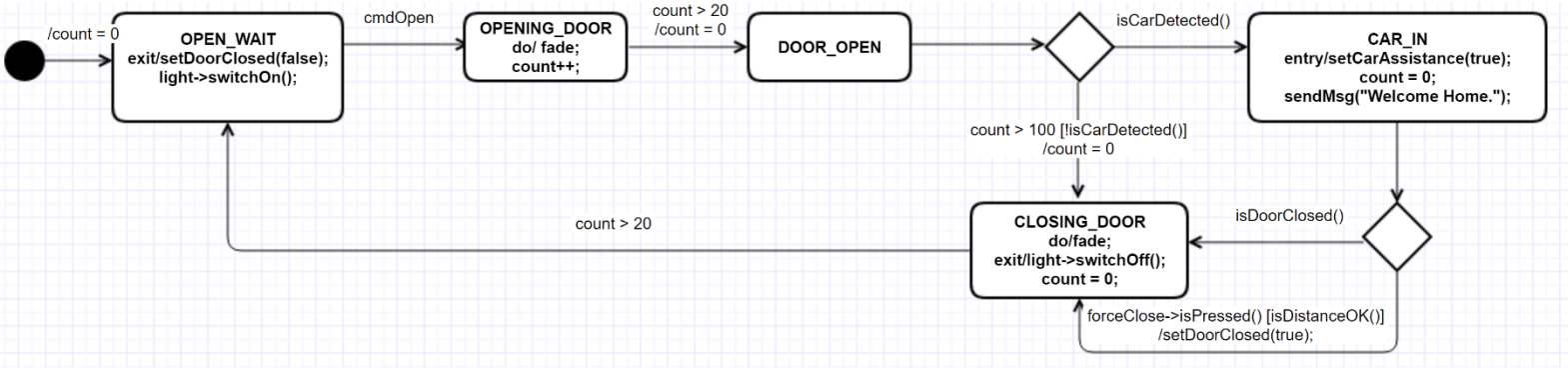
Per la gestione degli switch (TOUCH e CLOSE) viene effettuato un polling. Abbiamo preferito usare il polling al posto di una gestione ad interrupt per mantenere una certa coerenza con lo scheduler cooperativo (quindi, non preemptive) ed evitare problemi legati alla gestione degli interrupt.

Per quanto concerne la scelta del periodo di ticking dello scheduler abbiamo optato per 100 ms, considerato che, in base a valutazioni empiriche, tutti i task funzionavano correttamente con tale periodo e tenendo avendo sempre in mente l’esigenza di non perdere eventi legati alla pressione dei pulsanti tattili.

Per la gestione di sensori (Pir e Sonar) e degli attuatori (Led) sono state utilizzate le librerie fornite durante le esercitazioni di laboratorio. Inoltre è stata utilizzata la libreria MsgService (fornita durante il corso), che implementa un’astrazione per l’invio e la ricezione di messaggi su seriale che facilita la gestione delle comunicazioni tra Arduino e PC.

Di seguito vengono riportati i diagrammi degli stati relativi alle macchine a stati che rappresentano i 2 task principali:

FSM – GarageDoorTask



Inizialmente il sistema si trova fermo nello stato OPEN_WAIT, in attesa di ricevere il comando di apertura inviato dal PC sulla seriale. Quando lo riceve, transita nello stato OPENING_DOOR nel quale avviene l'apertura della porta del garage, simulata dall'accensione progressiva di un led.

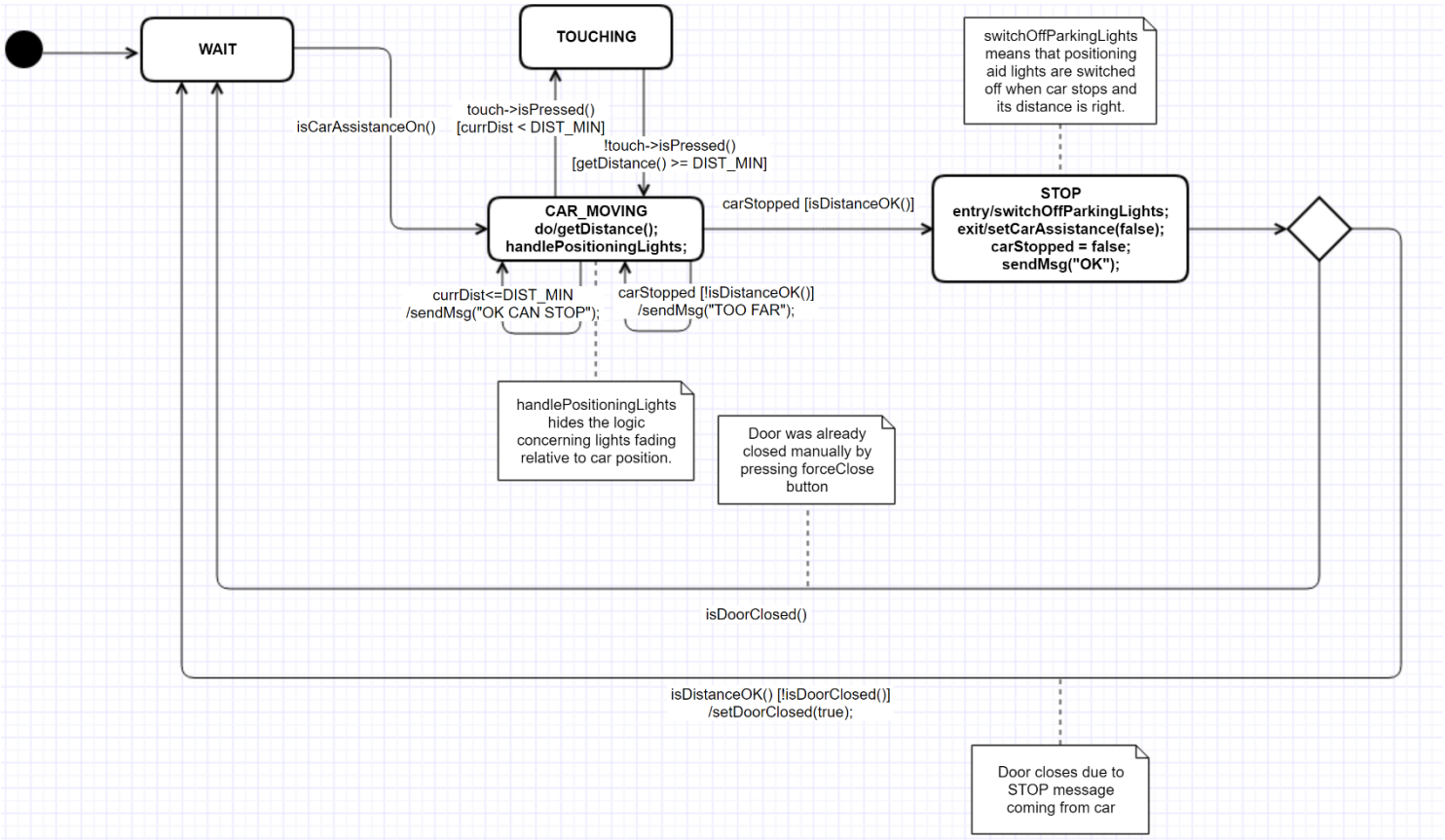
Ipotizzando di avere un periodo di 100ms e che l'apertura della porta impieghi 2 sec., allora il sistema dovrà transitare nello stato DOOR_OPEN quando la var. count (che viene incrementata ogniqualvolta avviene l'esecuzione del task da parte dello scheduler) ha raggiunto il valore 20.

Dopodiché si aprono 2 possibilità: una è che il pir non rilevi alcun movimento, quindi l' auto non è entrata e trascorsi 10 secondi la porta viene chiusa (passaggio allo stato CLOSING_DOOR). L'altra possibilità è che la macchina venga rilevata dal pir (passaggio a CAR_IN): a questo punto il sistema invia al pc che simula la macchina il messaggio "Welcome Home." e fa entrare in azione il task di assistenza al parcheggio.

Dallo stato CAR_IN il sistema esce, o perché l'assistenza al parcheggio ha fatto chiudere in automatico la porta (nel caso in cui abbia ricevuto lo STOP dalla macchina e questa si trovi ad una distanza <= DIST_CLOSE[1]), oppure perché è stato premuto il pulsante per forzare la chiusura del garage e la distanza della macchina era distanza <= DIST_CLOSE[1]. In entrambi i rami lo stato in cui si passa è CLOSING_DOOR in cui avviene la chiusura la porta simulata dallo spegnimento progressivo del led LR. Una volta trascorsi 2 secondi il sistema ritorna allo stato iniziale.

[1] Nel diagramma il controllo distanzaCorrente <= DIST_CLOSE è incapsulato nel metodo isDIstanceOK()

FSM – ParkingAssistanceTask



Inizialmente il task si trova nello stato di WAIT in cui è in attesa che il GarageDoorTask lo scateni.

Una volta avviata l'assistenza al parcheggio si passa nello stato CAR_MOVING. Questo stato rappresenta il "core" di questo task, infatti è qui che avviene la gestione dei led LDIST che indicano la vicinanza della macchina al muro, il controllo se l'auto tocca il muro (pressione pulsante Touch) e controllo dello STOP eventualmente inviato dall'auto per indicare che si è fermata.

La distanza dell'auto dal muro viene mappata nell'intensità dei led LDIST in base alle specifiche richieste.

Nel caso in cui venga premuto il pulsante TOUCH e la macchina si trovi ad una distanza dal muro minore di DIST_MIN viene inviato il messaggio "TOUCHING" e il task transita nello stato TOUCHING finché il pulsante non viene rilasciato oppure l'auto si allontana ad una distanza maggiore di DIST_MIN. Questo stato oltre ad avere una valenza concettuale è presente per il fatto che così il messaggio viene stampato una volta sola anche se il pulsante continua ad essere premuto.

Nel caso in cui l'auto tenti di fermarsi (inviando STOP al garage) ad una distanza maggiore di DIST_CLOSE, che non consentirebbe alla porta di chiudersi, il sistema invia alla macchina il messaggio "TOO FAR" e rimane nello stato CAR_MOVING.

Se l'auto giunge ad una distanza dal muro minore di DIST_MIN viene inviato il messaggio "OK CAN STOP" per indicarle che può fermarsi.

Se l'auto si ferma (indicato dal valore della variabile booleana carStopped, che viene settata in base alla pressione del pulsante STOP nella GUI) ad una distanza <= DIST_CLOSE[1] allora il sistema entra nello stato STOP in cui vengono spenti i led LDIST di assistenza al parcheggio, viene stoppata l'assistenza parcheggio (mediante l'istruzione setCarAssistance(false)), viene resettato il valore della variabile carStopped e infine viene inviato il messaggio OK mediante seriale all'auto.

Da qui si aprono 2 alternative: se la porta era già stata chiusa forzatamente mediante pressione del tasto CLOSE allora la macchina transita direttamente allo stato iniziale WAIT senza inviare al GarageDoorTask il comando di chiudere la porta.

Al contrario, se la porta è ancora aperta e la distanza della macchina dal muro ne consente la chiusura allora il task comunica al GarageDoorTask che può chiudere la porta (mediante l'istruzione setDoorClosed(true)) dopodiché ritorna allo stato iniziale.

GESTIONE SERIALE – CommandReaderTask

Il CommandReaderTask non fa altro che, ad ogni tick, controllare se è presente qualche messaggio sulla seriale. In caso affermativo tenta di interpretare tale comando e se corrisponde ad un comando noto esegue il metodo appropriato. Nello specifico, se giunge il comando di apertura della porta invoca il metodo openDoor() sul garage mentre se l'auto si ferma (premendo stop nella GUI) invoca il metodo setCarStopped per indicare che l'auto si è fermata al ParkingAssistanceTask.

COMPUTER DI BORDO AUTO – Applicazione Java

L'applicazione Java rappresenta un simulatore del computer di bordo dell'automobile.

Si è deciso di realizzare una GUI in JavaFX piuttosto che in Swing poiché 2 dei 3 componenti del gruppo possiedono un PC con schermo con risoluzione QHD, e purtroppo Swing non si adatta a risoluzioni QHD. Di conseguenza un'interfaccia realizzata in Swing sarebbe stata visualizzata piccolissima su questi schermi risultando inutilizzabile.

La GUI presenta 3 bottoni (Arriving, Stop e Clear) e una TextArea utilizzata per visualizzare i messaggi provenienti dall'Arduino.

Il tasto "ARRIVING" viene utilizzato per inviare al garage un messaggio contenente la stringa "a" comprensiva di terminatore di stringa che comanda al garage di aprire la porta.

L'implementazione del monitor seriale per la lettura dei messaggi in arrivo avviene mediante una BlockingQueue nella quale vengono accodati tutti i messaggi in arrivo da seriale per poter essere poi gestiti. Questa implementazione può non risultare "estremamente piacevole" ma è stata molto utile in fase di sviluppo per individuare gli errori.

Il Serial Monitor alla base della GUI è implementato facendo ricorso all' interfaccia CommChannel e alla sua implementazione SerialCommChannel fornite a lezione.

Se l'auto viene rilevata e quindi viene letto il messaggio "Welcome Home." in arrivo dal garage, viene creato ed avviato un thread che si occupa della lettura dei messaggi in arrivo e dell'aggiornamento della TextArea. L'aggiornamento della TextArea avviene mediante la chiamata al metodo statico runAndWait della classe PlatformImpl di JavaFX. Questo è necessario poiché l'unico thread autorzzato ad esguire operazioni sulla GUI è il JavaFX Application Thread (analogo di SwingUtilities.invokeAndWait per Swing).

Il "problema" è che il metodo runAndWait non fa parte dell'API pubblica della versione attuale di JavaFX e quindi potrebbe essere soggetto a cambiamenti in future versioni di Java, ma si è ritenuto di utilizzarlo lo stesso per questo assignment a fini didattici.

L'invocazione di PlatformImpl.runAndWait al posto di Platform.runLater (che al contrario del primo fa parte delle API pubbliche), si è resa necessaria per non inondare l'event dispatcher thread di JavaFX di eventi da gestire, che porterebbero ad una GUI non responsive.

Quando viene premuto il pulsante STOP viene inviata sulla seriale la stringa "s" comprensiva di terminatore e, se l'auto è entro la distanza necessaria per la chiusura della porta del garage, essa viene chiusa, viene visualizzato il messaggio "OK" in arrivo dal garage, dopodiché la GUI una volta trascorso il tempo di chiusura del garage ritorna allo stato iniziale.

Per motivi di praticità, una volta ritornati allo stato iniziale c'è la possibilità di far entrare un'altra macchina, ipotizzando che la macchina dentro al garage "sparisca".