Relazione Smart Coffee Machine

Introduzione

Lo scopo del progetto era quello di realizzare, tramite la piattaforma Arduino e la comunicazione seriale con una piattaforma realizzata in Java, un simulatore di una macchina del caffè intelligente, che con la rilevazione di movimento e distanza capisca quando accendersi, prepararsi per fare una caffè e tornare in modalità stand by.

Architettura e Progettazione Iniziale

Siamo partiti con una progettazione orientata agli oggetti e che al suo interno includeva la creazione di più task, ognuno con un determinato compito e la possibilità di comunicare con gli altri attraverso delle variabili globali condivise.

Questi sono i seguenti task:

- **MovementTask**:
- **DistanceTask**:
- **Maintenance Task**:

  Questo task gestisce gestisce lo scambio dei messaggi con la piattaforma realizzata su java attraverso la comunicazione seriale.

  Tale task viene eseguito quando il sistema realizzato entra nello stato di MAINTENANCE. Tale variazione di stato viene espressa attraverso il set della variabile condivisa maintenance. A questo punto il task controlla si mette in ascolto di messaggi in arrivo sulla seriale, in quanto attende il messaggio di ricarica che verrà inviato lato java, per effettuare la ricarica del caffè.

  Tale messaggio contiene un stringa indicante il numero di caffè con cui ricaricare la macchina. Questo dato viene convertito in un intero e sommato alla variabile condivisa che tiene traccia dei caffè da poter fare.

  Il messaggio ricevuto viene poi cancellato, in quanto Arduino non contiene il garbage collector, portando alla saturazione di memoria dopo pochi cicli di esecuzione. E la variabile maintenance viene rosetta e si esce dallo stato di MAINTENANCE.

- **MakeCoffee Task**: Questo task gestisce la produzione di un caffè. Nel nostro sistema è emulata attraverso il lampeggiamento di tre led in sequenza, per la durata totale di 3 secondi. Il tempo di accensione di ogni led viene tenuto memorizzato attraverso una variabile , che viene incrementata del periodo del task, ad ogni chiamata dello scheduler. In questo modo si riesce ad avere un periodo di esecuzione del task più basso, per essere responsivo, e una gestione dei led indipendente da esso. Tale task entra in esecuzione quando viene impostata a true la variabile booleana condivisa makeCoffee, e viene impostata a false dal task stesso. In questo modo il task viene mandato in esecuzione da main task e dopo essere terminato, permette di capire al main task il cambiamento di stato.

- **Main Task**:

Progettazione specifica

Durante la fase di design del progetto un pò più approfondita abbiamo deciso di considerare il main task, il task principale che gestendo gli stati del sistema, avrebbe fatto da controllore per far eseguire gli altri task nel momento opportuno. In quanto gli altri task possono essere divisi in due tipologie:

- Task di rilevazione : Questi task assegnano il valore letto dai sensori alle corrispettive variabili condivise, in modo da permettere al main task di avere i dati dei sensori sempre aggiornati, ma senza doversene occupare lui direttamente.
- Task di esecuzione : In modo differente, questi task vengono "avviati" dal main task ed eseguono delle routine come il lampeggiamento dei led e la comunicazione con la piattaforma in Java. Spiegandoli in modo più accurato, questi task vengono eseguiti dallo scheduler in continuazione come gli altri task e con stessa priorità, ma allora interno hanno un semplicissimo controllo sulla variabile condivisa rispettiva e nel caso in cui sia a false, il task non esegue niente. Altrimenti viene eseguita la routine specificata.

Durante questa fase, cia siamo imbattuti nella decisione di dove eseguire la modalità di risparmio energetico nello stato STAND BY. Inizialmente avevamo pensato una soluzione in cui tale operazione veniva gestita da un task a se stante, ma dopo varie considerazione abbiamo deciso per l'inserimento della sleep mode all'interno del main task, in quanto essendo il task principale era giusto che una funzionalità importante come questa fosse gestita al suo interno. Per la sua implementazione successiva c'erano due possibilità: l'assegnamento di un interrupt al sensor PIR, che risvegliasse il micro controllore quando veniva rilevato un movimento. Una sleep mode ciclica che si risvegliasse periodicamente con il timer di esecuzione dello scheduler. Fra le due soluziono abbiamo optato per la seconda in quanto abbiamo voluto mantenere il sistema basato su macchina totalemte sincrona, evitando qualsiasi elemento asincrono.