Alma Mater Studiorum · Università di Bologna

Campus di Cesena

SCUOLA DI SCIENZE

Corso di Laurea in Ingegneria e Scienze Informatiche

Analisi, Progettazione e Implementazione di una serra intelligente per la coltivazione di piante

Docente del Corso:

Chiar.mo Prof.

RICCI ALESSANDRO

Studente:

671579

VIGNOLA FRANCESCO

francesco.vignola@studio.unibo.it

Indice

Elenco delle figure

Introduzione

La Smart Greenhouse è un ambiente controllato allo scopo di garantire la crescita di differenti specie di piante, ottimizzandone la produzione. Nel particolare, questa gestisce in modo autonomo, ovvero minimizzando l'intervento umano, il monitoraggio e il controllo del microclima al suo interno, analizzando i parametri climatici che, direttamente o indirettamente, forniscono informazioni utili per la crescita e la produzione delle piante. Gli aspetti importanti da considerare in tale contesto, oltre al microclima, sono:

- le condizioni climatiche all'interno della serra, per evitare il prolificarsi di batteri e muffe;
- le condizioni del terreno, ovvero verificare se il terreno è sufficientemente umidificato affinchè le piante possano idratarsi;
- la luce assorbita dalle piante, in particolare quella solare, in quanto fondamentale per la fotosintesi clorofilliana.

I benefici apportati dalla serra automatizzata sono:

- crescita e produzione delle piante durante tutto l'anno, senza quindi attendere la stagione più appropriata;
- manutenzione costante, e soprattutto precisa, della serra grazie alla sua automatizzazione/intelligenza, evitando fenomeni dannosi per le piante (asfissia radicale, nascita e prolificazione di muffe e batteri, ecc...);
- l'utente è sempre informato, anche a distanza, riguardo alle condizioni della serra.

Inoltre, la serra acquisisce un ruolo attivo grazie all'utilizzo della rete e, in particolare, di protocolli realizzati ad-hoc affinchè il reperimento e la notifica delle

informazioni siano veloci, non richiedano grosse quantità di risorse e siano semplici da realizzare. Quindi, si è cercato di realizzare un sistema in cui ogni componente sia in grado di rendersi riconoscibile ed acquisisce intelligenza grazie al fatto di poter comunicare il proprio stato e accedere ad informazioni aggregate da parte di altri. È proprio per queste motivazioni che la serra può essere definita, a tutti gli effetti, una "cosa" all'interno della rete, in linea con il concetto dell'Internet of Things, e un sistema machine-to-machine.
Il concetto di M2M e di IoT sono, spesso, molto correlati tra loro, ma pur sempre due concetti differenti: il M2M utilizza una comunicazione point-to-point isolata, senza alcuna interazione umana, quindi poco scalabile e flessibile ma più sicura in quanto non esposta ad attacchi della rete; a differenza, l'IoT è stato pensato per creare un collegamento tra il mondo fisico e il mondo digitale, con cui l'uomo può interagire offrendo, quindi, una maggiore flessibilità e scalabilità sia a livello applicativo che architetturale.
Il concetto M2M può essere riconoscibile (anche se non interamente) nella serra con la comunicazione seriale tra arduino e raspberry, in cui si è utilizzato un protocollo ad-hoc, ma molto elementare, per lo scambio dei messaggi. Mentre, il concetto di IoT è riconoscibile nella comunicazione tra raspberry e android, in cui si è utilizzato un protocollo molto leggero e nato proprio in questo contesto, ovvero MQTT.

Restrizioni alla Smart Greehouse

Dato che realizzare il suddetto progetto richiede tempi, costi, e conoscenze (soprattutto in ambito agrario) non disponibili ai fini dell'esame, ho deciso di semplificare il progetto. Il prototipo si pone i medesimi obiettivi del progetto originale, ma considera un'unica specie di piante anzichè molteplici; le dimensioni della serra sono ridotte e, i dispositivi utilizzati, anche conosciuti come sensori e attuatori, sono molto semplici e non idonei per un contesto reale.

Capitolo 1

Specifica dei Requisiti

La serra è dotata di tre sistemi:

- il controllore della serra (Arduino): si occupa di adempiere agli obiettivi prefissati, e di aggiornare opportunamente la centralina.
- la centralina (Raspberry Pi): rimane in ascolto di messaggi ricevuti dal controllore e li dirama allo smatphone, affinché l'utente ne venga a conoscenza; mostra sul display le informazioni necessarie.
- l'app: mostra all'utente lo stato della serra ed i valori dei relativi parametri.

Questi sottosistemi interagiscono al fine di garantire un microclima ideale, una sufficiente quantità di luce per le piante, un'umidità del terreno idonea, monitorando, inoltre, il livello di acqua disponibile nella tanica. L'interazione non è necessaria, ovvero, il controllore è in grado di eseguire i suoi compiti anche senza comunicare con la centralina, in quanto utilizza dei parametri di riferimento di default. La centralina e lo smartphone vengono utilizzati al solo scopo di rendere conoscibile lo stato della serra all'utente, e affinché questo possa cambiare i valori dei parametri a propria discrezione.

Nei seguenti tre paragrafi saranno delineate le specifiche dei requisiti per ciascun sottosistema.

1.1 Controllore della serra - Arduino

1.1.1 Controllo del microclima

Il controllo del microclima si basa sul controllo della temperatura all'interno della serra.

In particolar modo, se la temperatura eccede la soglia massima impostata verrà acceso il sistema di raffreddamento, mentre se la temperatura è al di sotto della soglia minima, verrà acceso il sistema di riscaldamento. In entrambi i casi, al raggiungimento della temperatura ottimale il sistema dovrà essere spento.
Ad ogni cambiamento di stato e, ogni qualvota si rileva una nuova temperatura, il sistema deve inviare un messaggio alla centralina.
La soglia minima e massima vengono calcolate tenendo conto dell'intervallo impostato dall'utente più un'eventuale tolleranza (anche questa configurabile), ovvero:

$$minThreshold = minTemperature - (rangeTemperature * tolerance)$$
$$maxThreshold = maxTemperature + (rangeTemperature * tolerance)$$

dove *rangeTemperature* è uguale alla differenza tra la temperatura massima e quella minima di default o impostata dall'utente; *tolerance* indica quanto il sistema deve essere reattivo al rilevamento di una temperatura oltre la soglia.
La temperatura ottimale, viene calcolata considerando un intervallo medio tra la temperatura massima e quella minima, considerando la tolleranza del sistema:

$$meanTemperature_{min} = minTemperature + (rangeTemperature * tolerance)$$
$$meanTemperature_{max} = maxTemperature - (rangeTemperature * tolerance)$$

Il sistema di raffreddamento, si basa sulla tecnica, molto diffusa nelle serre, del rinfrescamento evaporativo realizzato tramite due ventole ed un nebulizzatore: una ventola aspira aria dall'esterno, l'altra la espelle all'esterno, ed il nebulizzatore aiuta a velocizzare il processo di rinfrescamento.

1.1.2 Controllo dell'umidità del terreno

Il monitoraggio dell'umidità del terreno deve essere eseguito costantemente, ad esclusione degli orari più caldi. L'intervallo temporale predefinito va dalle ore 10 alle ore 18, e può essere configurato diversamente. All'infuori di tale intervallo, se il terreno è secco, dovrà essere irrigato affinchè raggiunga il giusto livello di umidità.
L'irrigazione viene eseguita azionando una pompa ad immersione, posta all'interno di un serbatoio. Se il livello d'acqua presente nella tanica non è sufficiente, non dovrà essere azionata la pompa, in modo da evitare che si bruci. É, quindi, necessario controllare anche il livello di acqua presente nella tanica, in modo tale da poter avvisare l'utente quando è al di sotto del livello minimo. In tale condizione,

il sistema invia un messaggio di allarme ed aziona un buzzer e un led lampeggiante. Il controllo del livello dell'acqua non deve dipendere dal controllo dell'umidità del terreno in quanto l'utente deve essere avvisato ogni qual vota il livello di acqua è inferiore al livello minimo, anche se il terreno è umido e non si deve irrigare. In tal modo si garantisce che l'utente possa intervenire prima che il sistema rileva che il terreno è secco. Ogni volta che si irriga viene inviato un messaggio alla centralina.

1.1.3 Controllo della luce solare

Il sistema deve costantemente monitorare la quantità di luce che le piante assorbono durante il giorno e memorizzare il numero di ore/minuti/secondi in cui la pianta viene esposta alla luce. In questo modo, se la quantità di luce solare non è sufficiente, verrà accesa una lampada per compensare la mancanza di luce all'infuori dell'orario di rilevazione stabilito.
L'intervallo in cui il sistema deve rilevare la quantità di luce va dalle 7 del mattino alle 20 di sera, ma è possibile configurarlo a piacimento. Se il sistema accende la luce artificiale, ma non riesce a raggiungere la quantità di luce necessaria fino all'inizio della rilevazione del giorno successivo, il sistema deve comunque spegnere la luce. Invece, se il sistema ha rilevato una quantità di luce solare sufficiente, non deve essere accesa la lampada, ma attendere l'inizio della rilevazione del giorno successivo.
Per adempiere a questo scopo, è necessario che il sistema disponga di un orologio fisico da aggiornare ogni 30 minuti (se disponibile la connessione con la centralina) in modo da garantire al controllore di lavorare anche senza interagire con la centralina.

1.2 Centralina - Raspberry

La centralina è una componente fondamentale per l'interazione con l'utente, ma non necessaria. Essa realizza un ponte tra la rete esterna e quella interna, in grado di mettere in comunicazione il controllore della serra con l'utente. La centralina permette di accedere allo stato della serra anche dal display. Quest'ultimo è costantemente spento e alla pressione del pulsante home si accende mostrando le informazioni importanti in forma sintetica (se la rilevazione della luce solare è in corso o se si sta illuminando con luce artificiale; se si sta irrigando il terreno; se si sta riscaldando o rinfrescando l'aria).

Se, mentre le schermo è acceso, viene premuto il pulsante back o next, il display visualizzerà in modo circolare informazioni più dettagliate per ogni compito.

1.3 Android

L'app Android deve mostrare tutte le informazioni inerenti allo stato della serra e mostrate dalla centralina, in modo da intervenire se necessario. Oltre a tali informazioni, l'app deve mostrare la configurazione con cui il sistema lavora, quindi temperatura minima e massima, intervallo di rilevazione della luce solare e quello in cui si deve evitare di irrigare, ecc., permettendo all'utente di cambiarla a sua discrezione.
L'app non è stata sviluppata in quanto si è scelto di utilizzarne una presente nel Play Store, che supporti il protocollo MQTT e che offra diverse funzionalità.

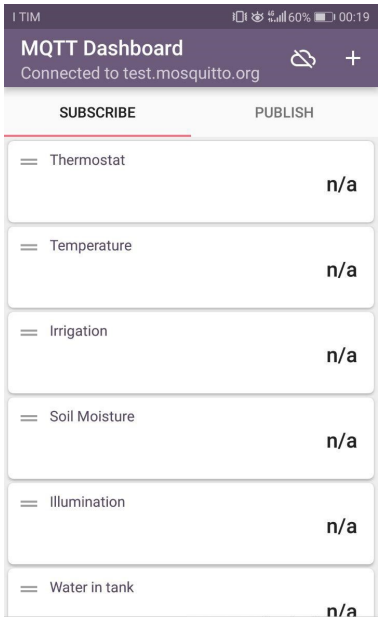

Figura 1.1: Activity Subscribe

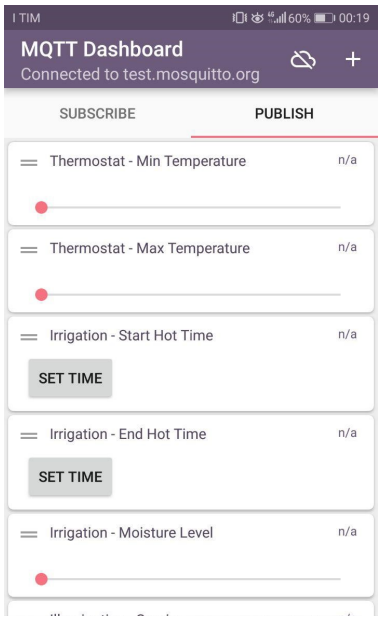

Figura 1.2: Activity Publish

Capitolo 2

Progettazione architetturale del software

In questo capitolo, dopo aver analizzato le specifiche dei requisiti, verrà presentata la progettazione architetturale del software, per entrambi i sistemi[1]. Il principio fondamentale della progettazione architetturale è il disaccoppiamento tra il software e l'hardware, favorendo così una progettazione top-down, ovvero ad alto livello. Solo in tal modo, è possibile creare componenti modulari quanto più indipendenti tra loro, estendibili e riusabili.
In linea a quanto detto, la modellazione architetturale si basa principalmente sul paradigma ad oggetti, sfruttando opportune architetture e pattern di progettazione per ciascun sistema.

2.1 Scelta dell'architettura software - Arduino

La scelta dell'architettura software è ricaduta sul modello multi-tasking con scheduler cooperativo, utilizzando variabili condivise per la comunicazione tra task e memorizzando solo l'ultimo messaggio arrivato per la comunicazione verso l'esterno, nel mio caso con il raspberry. Le motivazioni di questa scelta, sono legati sia ai principi fondamentali dell'ingegneria del software sia alla natura del sistema in esame. Da una parte l'architettura a task rende modulare il sistema, favorendo la qualità del software e ottenendo vantaggi, quali facilità di comprensione, estendibilità, riusabilità e manutenzione. Dall'altra parte, il sistema cambia il suo comportamento in base allo stato in cui si trova e ai segnali in ingresso,

[1]La centralina basata sul Raspberry Pi 1 Model B e il gestore della serra basato su Arduino Uno Rev3

per cui mi è sembrato naturale optare per per un'architettura a task basata su macchina a stati finiti sincrona. Ed infine, non avendo eventi discreti, come la pressione del pulsante, a causa della natura analogica dei sensori utilizzati, non ho ritenuto opportuno utilizzare un'architettura ad eventi, adottando così la tecnica del polling.
L'architettura a task modellata con macchine a stati finiti sincrone aumentano la reattività del sistema, scegliendo opportunamente la frequenza con cui il task deve essere eseguito. L'architettura a task prevede che ogni task esegue un compito ben definito, ed è per questo motivo che ho implementato sei task:

- ComunicationServiceTask: si occupa della comunicazione con il raspberry;
- IrrigationControllerTask: controlla se il terreno ha bisogno di essere irrigato;
- TemperatureControllerTask: controlla se il clima, all'interno della serra, è favorevole;
- IlluminatingLampTask: controlla la quantità di luce solare/artificiale necessaria alle piante;
- TankMonitoringTask: controlla il livello di acqua presente nel serbatoio;
- ClockControllerTask: gestisce l'orologio, sincronizzandolo ogni 30 minuti se è connesso a raspberry.

La comunicazione fra task è stata realizzata mediante variabili condivise sia per semplicità di implementazione sia per evitare di saturare la memoria della scheda Arduino, associando ad ogni task una coda di messaggi o creando una coda di messaggi a livello globale. Le stesse considerazione sono state effettuate per la comunicazione via seriale, decidendo così di salvare solo l'ultimo messaggio, perdendo quelli più vecchi. Inoltre, un ulteriore considerazione a sostegno di questa scelta è che il periodo con cui vengono inviati i messaggi ad Arduino è superiore al periodo con cui li esamina, quindi non verrà perso alcun messaggio. Ciò è dovuto al fatto che ad inviare i dati ad Arduino è l'utente, tramite smartphone o raspberry, ed il tempo di reazione dell'individuo è minore (ovvero più lento) rispetto a quello della scheda Arduino.
Un altro argomento fondamentale da considerare nell'architettura a task è la tipologia di scheduler. Anche in questo caso ho considerato sia la reattività del sistema, influenzata dalla priorità associata ad ogni task, e la semplicità d'implementazione

(corse critiche, mutua esclusione), optando per l'utilizzo dello scheduler cooperativo. Lo scheduler cooperativo, di tipo Round-Robin, permette l'esecuzione di un task fino al suo completamento (*run-to-completation*), senza alcuna interruzione, gestendo la priorità dei task in base all'ordine di inserimento. Il periodo dello scheduler è stato scelto come massimo comune divisore dei periodi dei task, in modo da evitare che lo scheduler si attivi con un periodo molto piccolo o molto grande. Nel primo caso, si avrebbe che in alcuni momenti nessun task viene schedulato in quanto il periodo di ogni task è maggiore, nell'altro caso si avrebbe una esecuzione ritardata di uno o più task, essendo il periodo dello scheduler maggiore rispetto al periodo del task, inficiando, quindi, la reattività del sistema.
Infine, passando dal livello architetturale di programmazione, ho optato per una programmazione guidata dalle interfacce, in modo tale che ogni task sia svincolato dall'effettiva implementazione delle risorse, siano essi sensori, attuatori o semplici classi; mentre per l'implementazione della macchina a stati finiti sincrona, ho adottato l'enum per modellare gli stati in quanto la macchina utilizzata è di tipo Mealy, per cui le azioni dipendono dallo stato attuale e dall'ingresso attuale rendendo non necessaria una modellazione ad oggetti degli stati.

Un'ultima considerazione è da fare sull'interfaccia *BroadcastReceiver*. Tale interfaccia, a mio avviso, è fondamentale per rendere, il CommunicationServiceTask, indipendente dagli altri task. Tale astrazione mi offre la possibilità di aggiungere uno o più task senza dover modificare il CommunicationServiceTask, in quanto è necessario solo implementare l'interfaccia BroadcastReceiver per poter ricevere un messaggio, previa registrazione. In tal modo posso effettuare una registrazione di un task al CommunicationService anche a run-time. Il task che vuole registrarsi al CommunicationTask deve implementare il metodo *onReceive(String action, String message)*. Questa astrazione fa uso del Decorator Pattern, che nel mio caso, è stato applicato già a compile-time. É fondamentale che in questo metodo non vengano svolte operazioni bloccanti.
Il controllo di un nuovo messaggio in arrivo viene eseguito con un periodo di 100 ms.

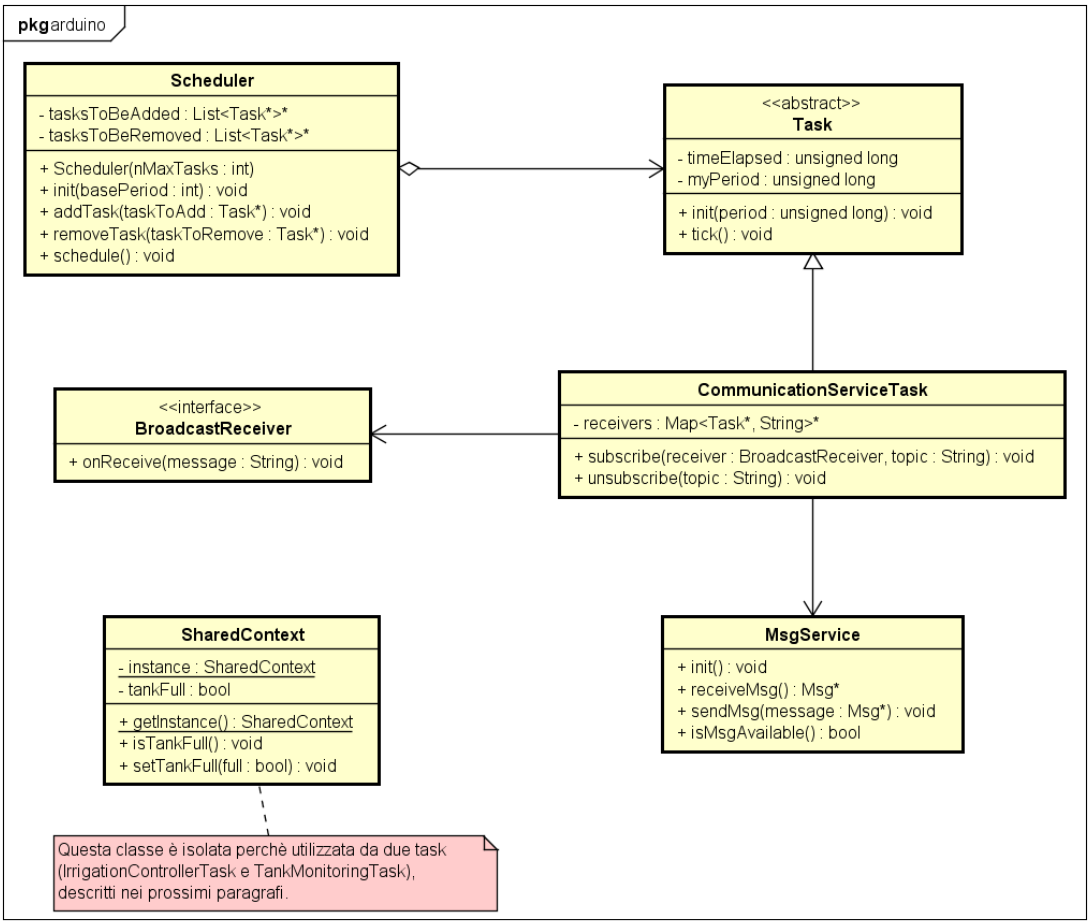

Figura 2.1: Diagramma delle classi dell'architettura utilizzata per Arduino

2.1.1 TemperatureControllerTask

Il sistema esegue, con un periodo di 1 secondo, il controllo della temperatura interna della serra, rinfrescando o riscaldando l'aria se questa non è ideale. Se viene rilevata una temperatura diversa dalla precedente, questa viene inviata alla centralina, in modo che l'utente possa vedere in real-time la temperatura esatta.

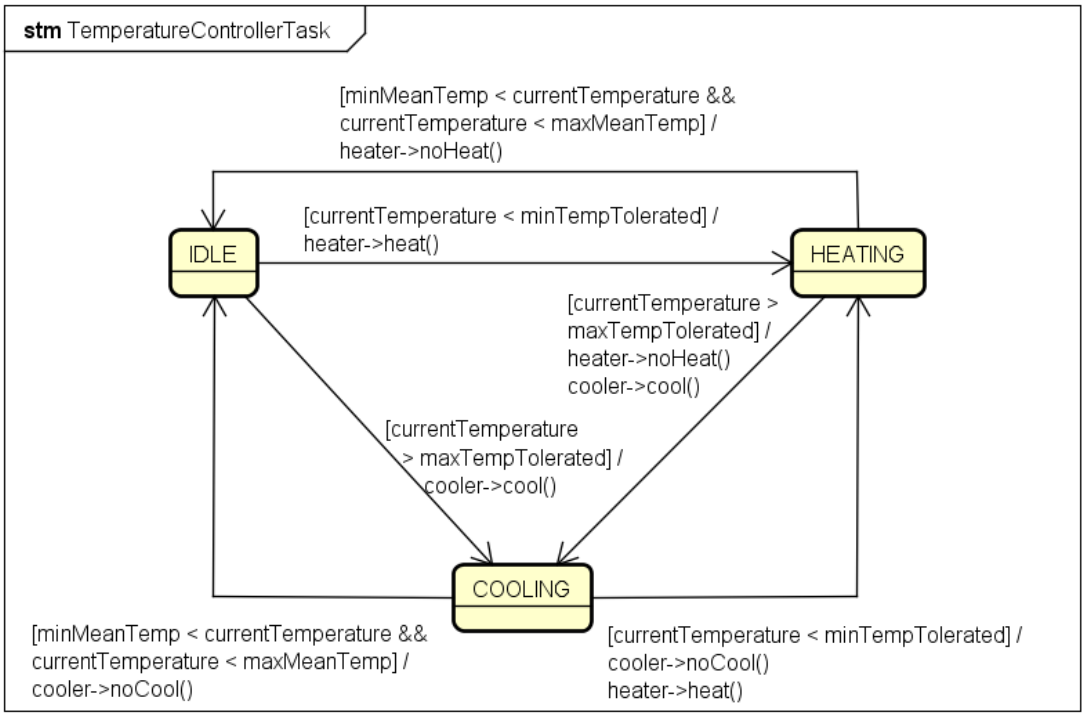

Figura 2.2: Macchina a stati finiti per TemperatureControllerTask

Di seguito è riportato il diagramma delle classi che evidenzia la struttura del *IrrigationTaskController* e delle risorse utilizzate.

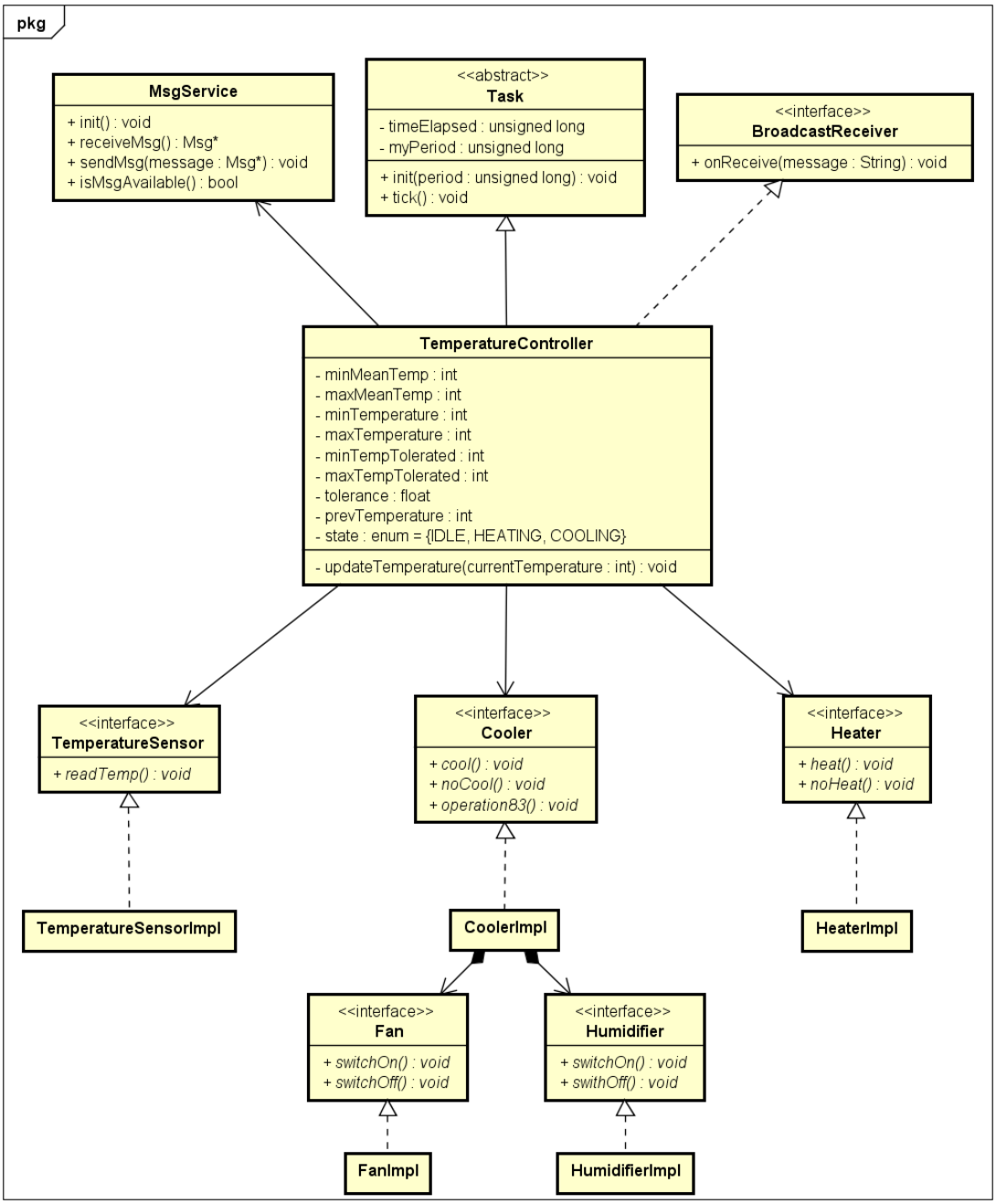

Figura 2.3: Diagramma delle classe per TemperatureControllerTask

2.1.2 IrrigationControllerTask

Il sistema esegue, con un periodo di 1 secondo, il controllo dell'umidità presente nel terreno, avvia l'irrigazione se l'umidità del terreno è inferiore a quella minima e se c'è acqua nel serbatoio. Infine, è fondamentale non irrigare durante l'intervallo di tempo in cui le ore sono più calde o nell'intervallo configurato dall'utente.

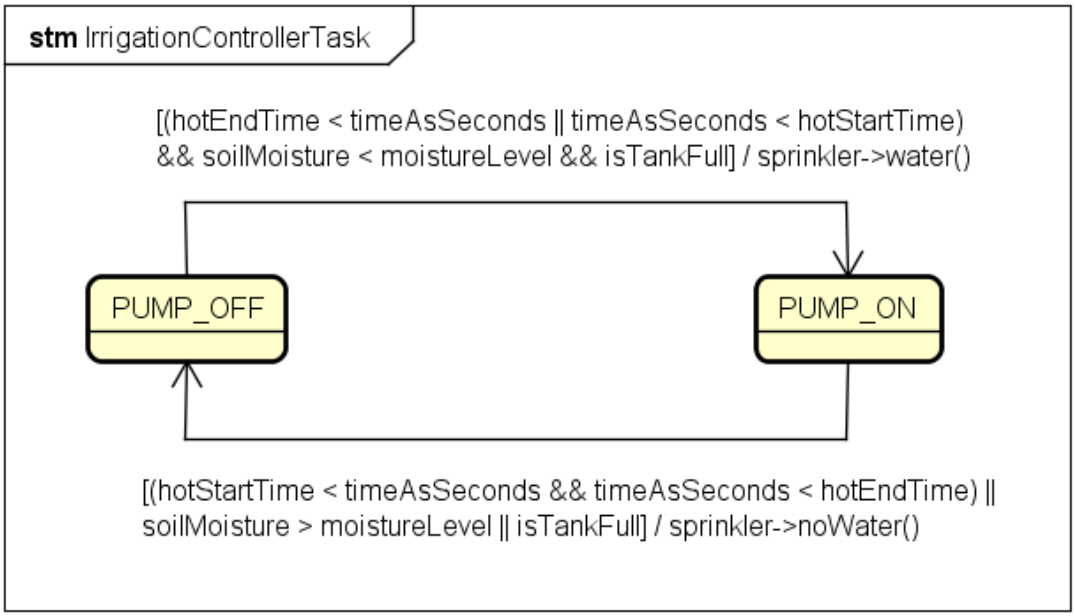

Figura 2.4: Macchina a stati finiti per IrrigationControllerTask

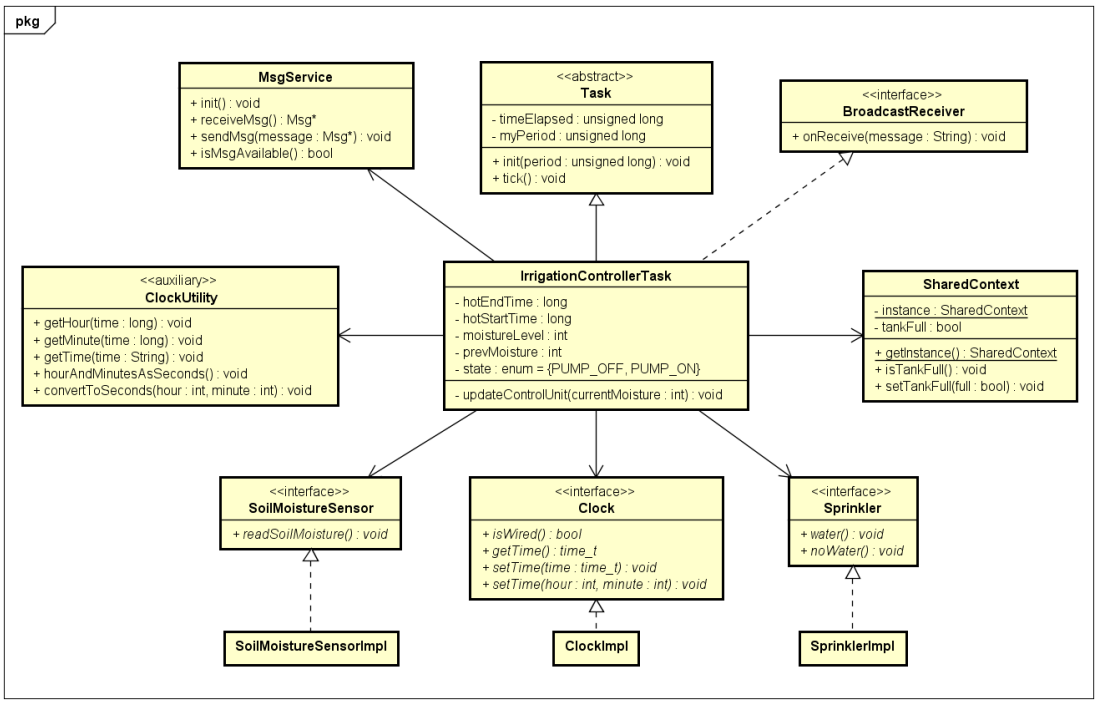

Figura 2.5: Diagramma delle classi per IrrigationControllerTask

2.1.3 IlluminatingLampTask

Il sistema, nelle ore indicate, deve controllare la quantità di luce (in termini di secondi) a cui le piante sono state esposte durante la giornata. Tale intervallo di tempo è impostato di default, ma può essere configurato dall'utente. Al di fuori di questo intervallo, se la quantità di luce è inferiore a quella necessaria, si deve accendere la lampada fino a che non si raggiunge la quantità necessaria o fino all'inizio della nuova rilevazione. La rilevazione viene effettuata una volta al giorno. Il periodo di rilevazione è di 1 secondo.

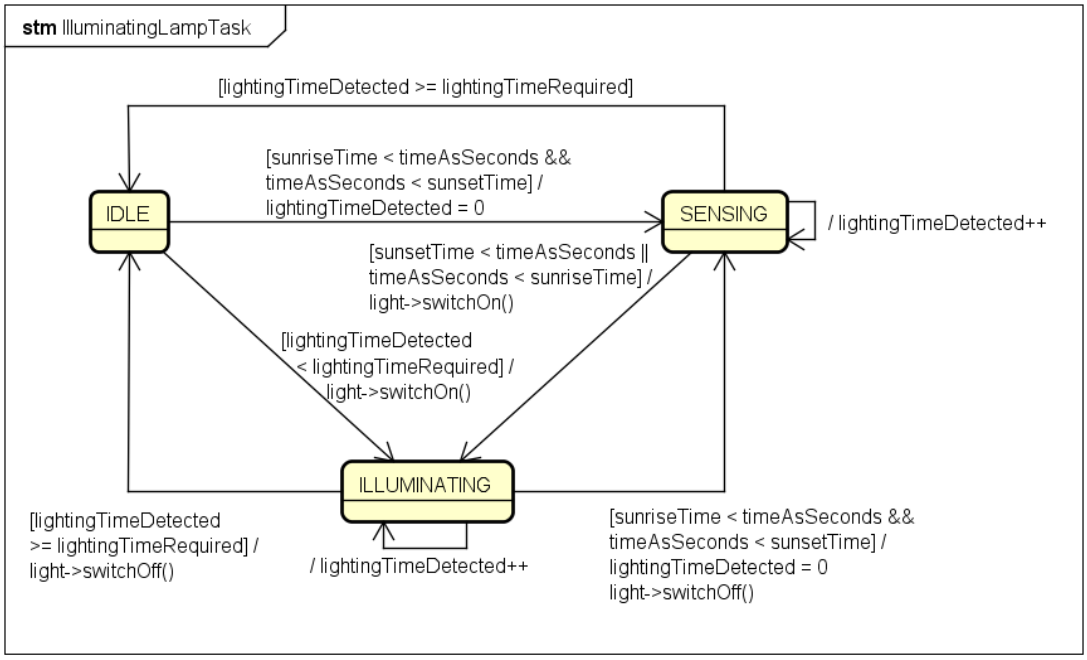

Figura 2.6: Macchina a stati finiti per IlluminatingLampTask

Di seguito è riportato il diagramma delle classi che evidenzia la struttura dell'*IlluminatingLampTask* e delle risorse utilizzate.

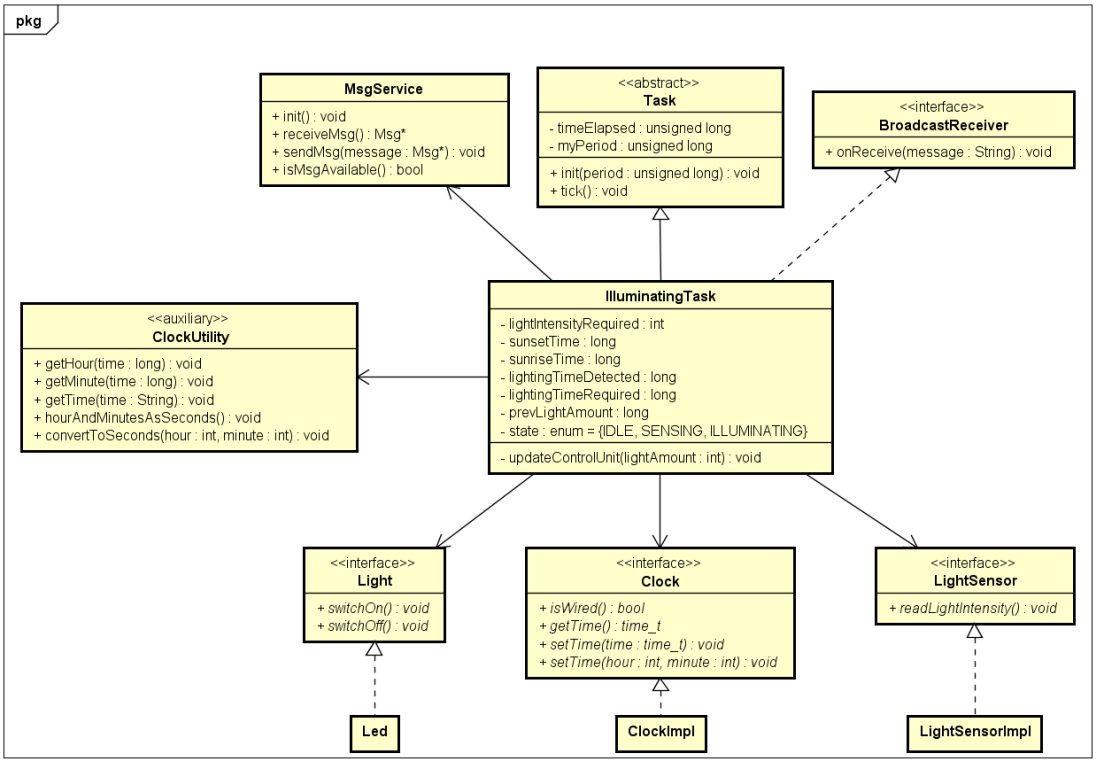

Figura 2.7: Diagramma delle classi per IlluminatingLampTask

2.1.4 TankMonitoringTask

Il sistema deve controllare, con un periodo di 500 ms, se è presente l'acqua nel serbatoio, ed entrare nello stato di allarme facendo lampeggiare il led rosso e azionando il buzzer. Se il sistema riceve un messaggio per silenziare l'allarme, si spegne il buzzer ma il TankMonitoringTask rimane ancora in allarme fin quando non viene rilevata una quantità sufficiente di acqua. Si noti che l'allarme non incide su tutto il sistema, ma i controllori della temperatura e dell'illuminazione continuano a svolgere normalmente i propri compiti.

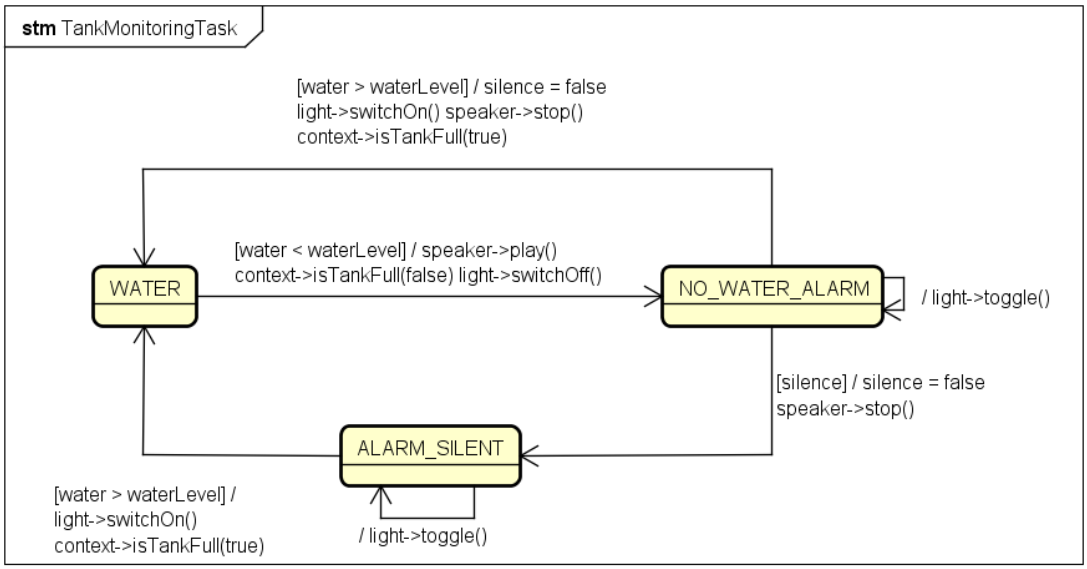

Figura 2.8: Macchina a stati finiti per TankMonitoringTask

Di seguito è riportato il diagramma delle classi che evidenzia la struttura del *TankMonitoringTask* e delle risorse utilizzate.

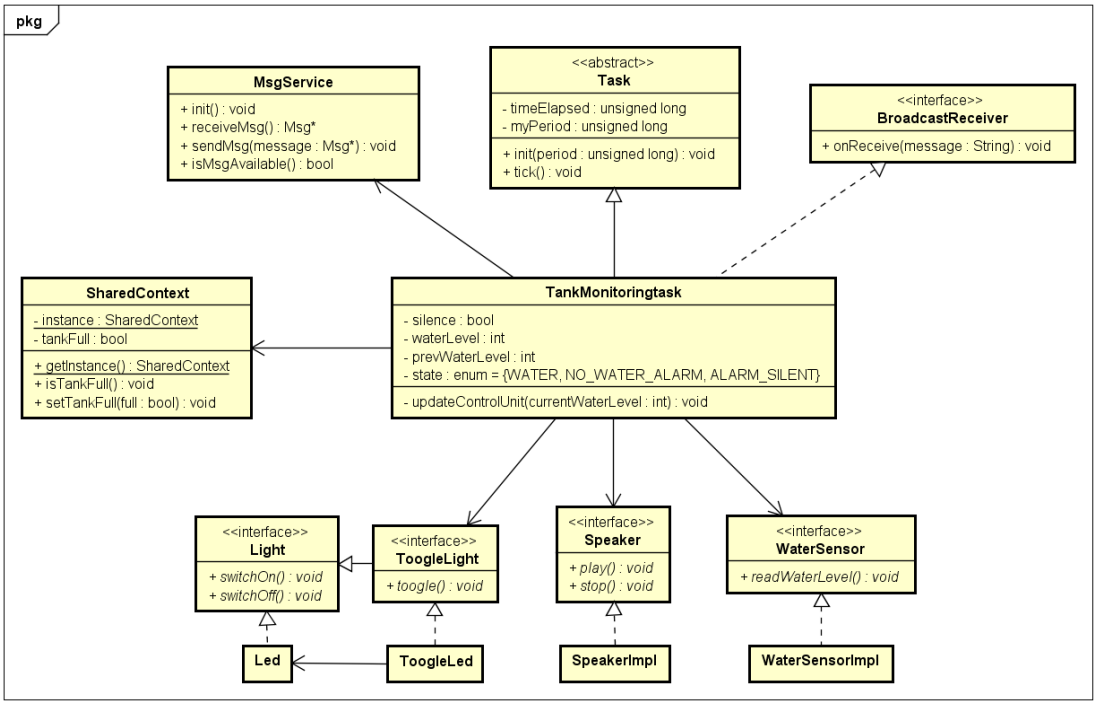

Figura 2.9: Diagramma delle classi per TankMonitoringTask

2.1.5 ClockControllerTask

Il sistema deve sincronizzare l'orologio ogni 30 minuti. Se questìultimo non è sincronizzato, viene inviato un messaggio a raspberry. Fin quando non si ottiene una risposta, Arduino invia ogni minuto la richiesta di sincronizzazione.
Questa modalità di gestione dell'orologio è stata realizzata in modo tale che il resto del sistema continui a svolgere i propri compiti.

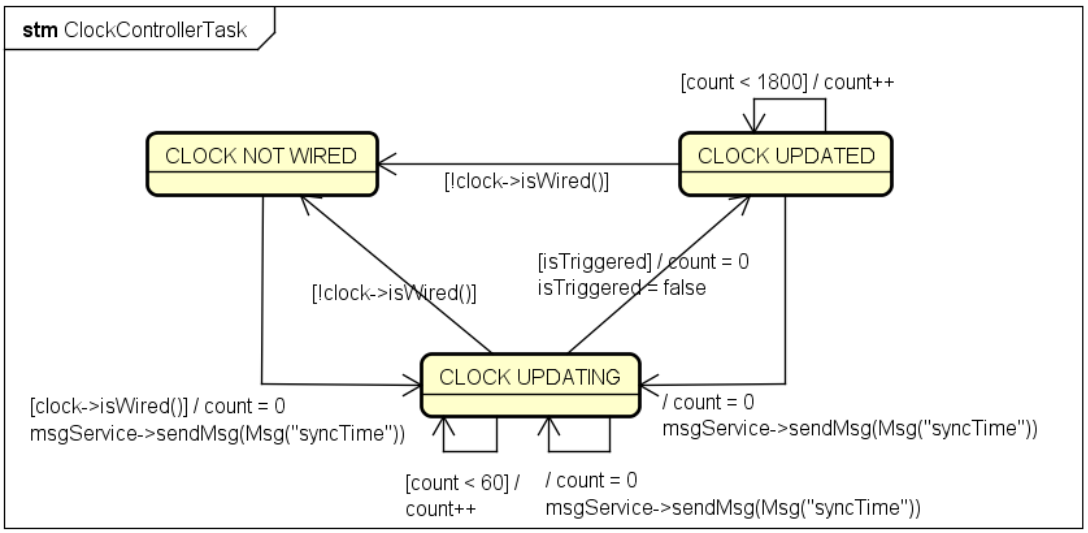

Figura 2.10: Macchina a stati finiti per ClockControllerTask

Di seguito è riportato il diagramma delle classi che evidenzia la struttura del *ClockControllerTask* e delle risorse utilizzate.

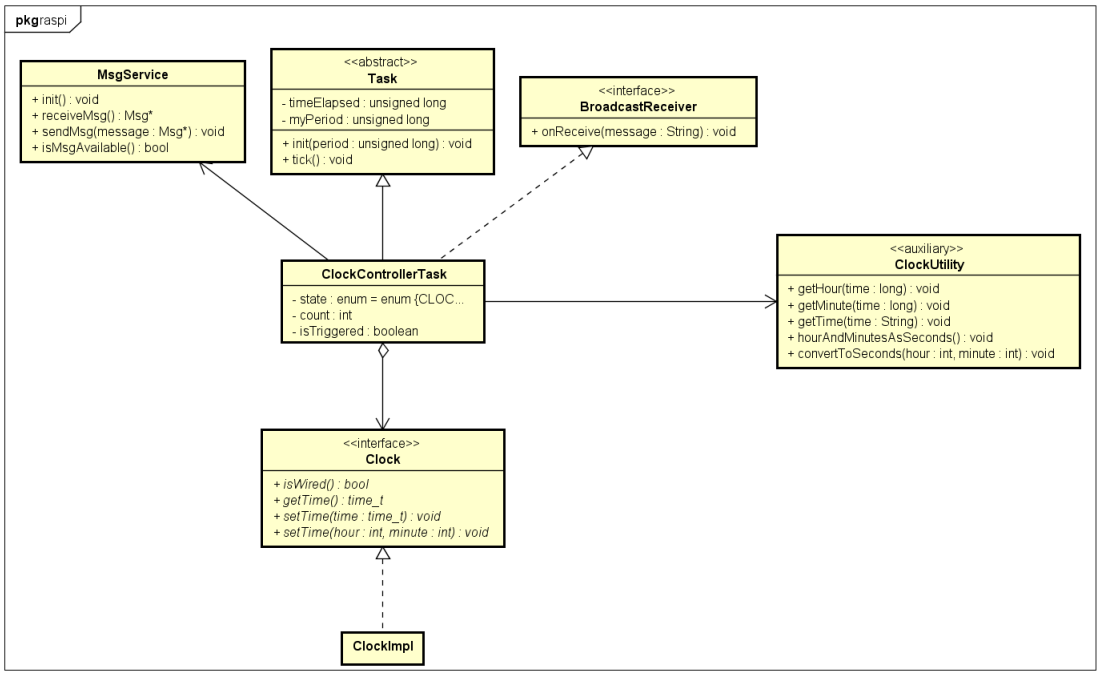

Figura 2.11: Diagramma delle classi per ClockControllerTask

2.2 Scelta dell'architettura software - Raspberry

L'architettura software con cui è stato progettato il sistema è un'architettura multi-threading basata sugli agenti, in cui ogni task, a livello logico, viene modellato con un unico thread, a livello fisico. Le motivazioni che mi hanno spinto ad adottare questa architettura riguardano sia aspetti legati alla reattività del sistema sia alla semplicità di sviluppo. Se da una parte, tale scelta, permette una gestione asincrona degli eventi offrendo una migliore reattività del sistema, dall'altra, utilizzando un singolo thread per un task, ci permette di facilitare lo sviluppo in quanto ogni task elabora sequenzialmente gli eventi ad esso destinati. Inoltre, si è scelto di modellare il comportamento di ogni singolo agente tramite una macchina a stati finiti del tutto asincrona, in quanto il suo comportamento dipende dallo stato in cui si trova. A differenza della scelta fatta per Arduino, ovvero di utilizzare la macchina di Mealy, per ovvi motivi, in questo caso ho utilizzato gli statecharts, ovvero un diagramma che permette di integrare le due tipologie di macchine a stati finiti, optando per una *programmazione per differenze*.
Proprio per questo motivo, si è scelto di implementare gli stati dello statechart, non tramite gli enum ma tramite vere e proprie classi. La maggior parte delle azioni vengono svolte all'interno dello stato, una dipende sia dallo stato in cui si trova

il sistema, sia dall'ingresso corrente. La modellazione degli stati è stata basata sullo State Pattern, che grazie alla delegazione è in grado di offrire una maggiore flessibilità e comprensibilità.

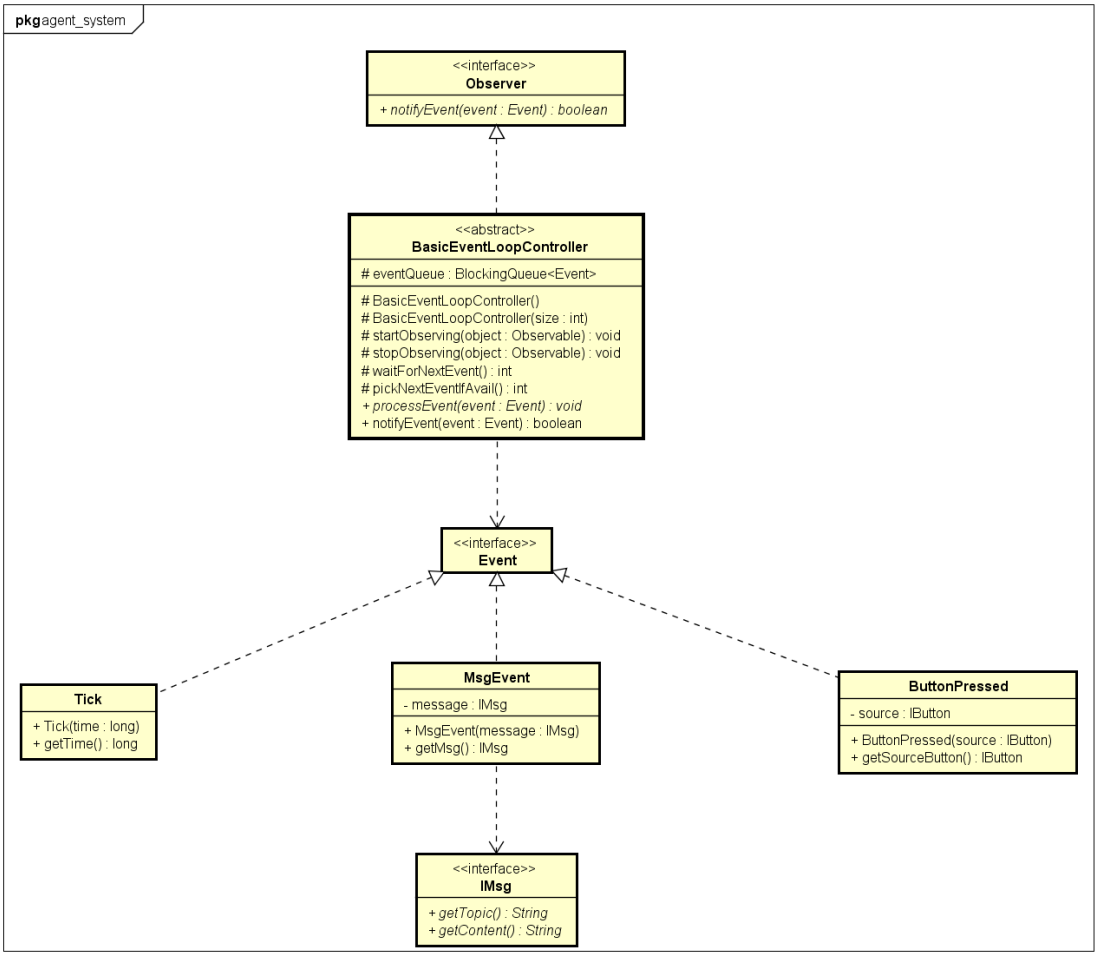

Figura 2.12: Diagramma delle classi principali dell'architettura di Raspberry

Nel presente scenario, la centralina ha lo scopo di effettuare un ponte di comunicazione tra Arduino e Raspberry, che dialogano con diversi protocolli di comunicazione, una via seriale l'altra via rete (ethernet). Per questo motivo è fondamentale disporre di due agenti che inoltrassero il messaggio sull'altro canale di comunicazione. Questi due agenti non costituiscono soltanto un *Messaging Bridge*, ma aggiornano il repository ed inviano eventi all'agente DisplayManager.
Di seguito è riportato il diagramma delle classi che evidenzia la struttura del messaging system.

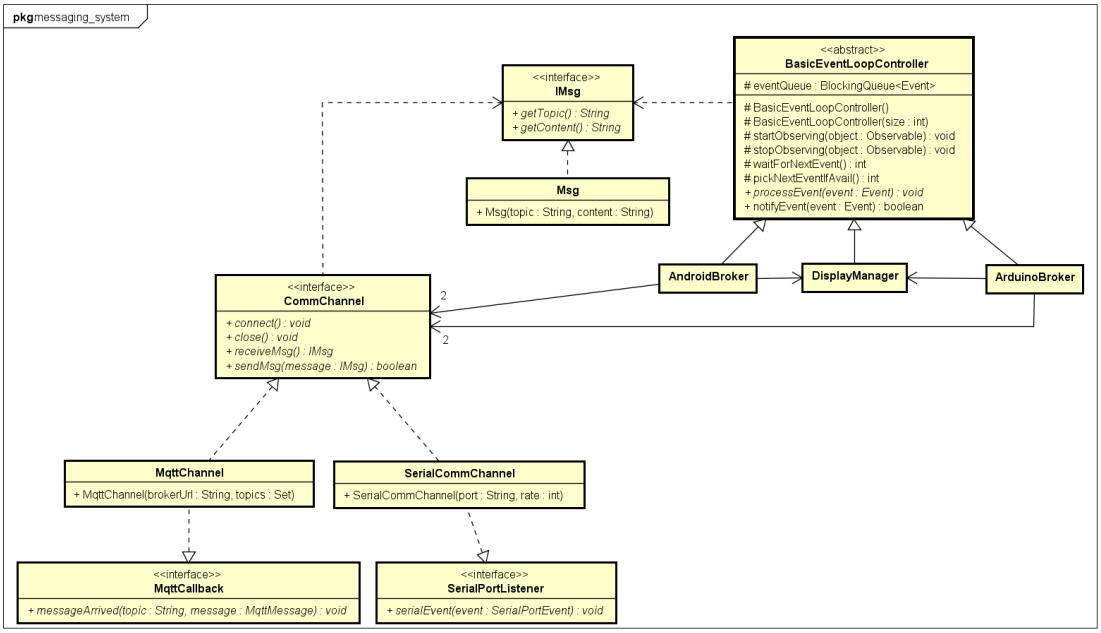

Figura 2.13: Diagramma delle classi per il sistema di comunicazione

2.2.1 Modellazione del DisplayManager

Un altro agente fondamentale è quello che si occupa di gestire il display. Quest'ultimo è sempre spento e si accende solo alla pressione del pulsante home, mostrando le informazioni principali in formato sintetico, e l'ora attuale. Alla pressione del pulsante next, o back, si passa alla pagina successiva, o precedente, in senso circolare. Ogni pagina, successiva a quella principale, mostra più informazioni circa ogni compito che Arduino deve svolgere. Se durante la visualizzazione di queste pagine, viene premuto il pulsante home, lo schermo si spegne, altrimenti se non viene effettuata alcuna azione, il sistema dopo un minuto spegne automaticamente lo schermo.

Se, mentre il display è spento o durante la visualizzazione delle diverse informazioni, si riceve un messaggio di allarme, il display mostra la scritta "Allarme: acqua assente" e il display lampeggia, fin quando l'utente non preme il pulsate home o invia un messaggio tramite smartphone. In questo momento il display viene spento e viene inviato ad Arduino un messaggio per silenziare l'allarme.

Di seguito verrà mostrata la modellazione dei vari aspetti del software, presentando i relativi diagrammi di classe e, successivamente, la macchina a stati finiti.

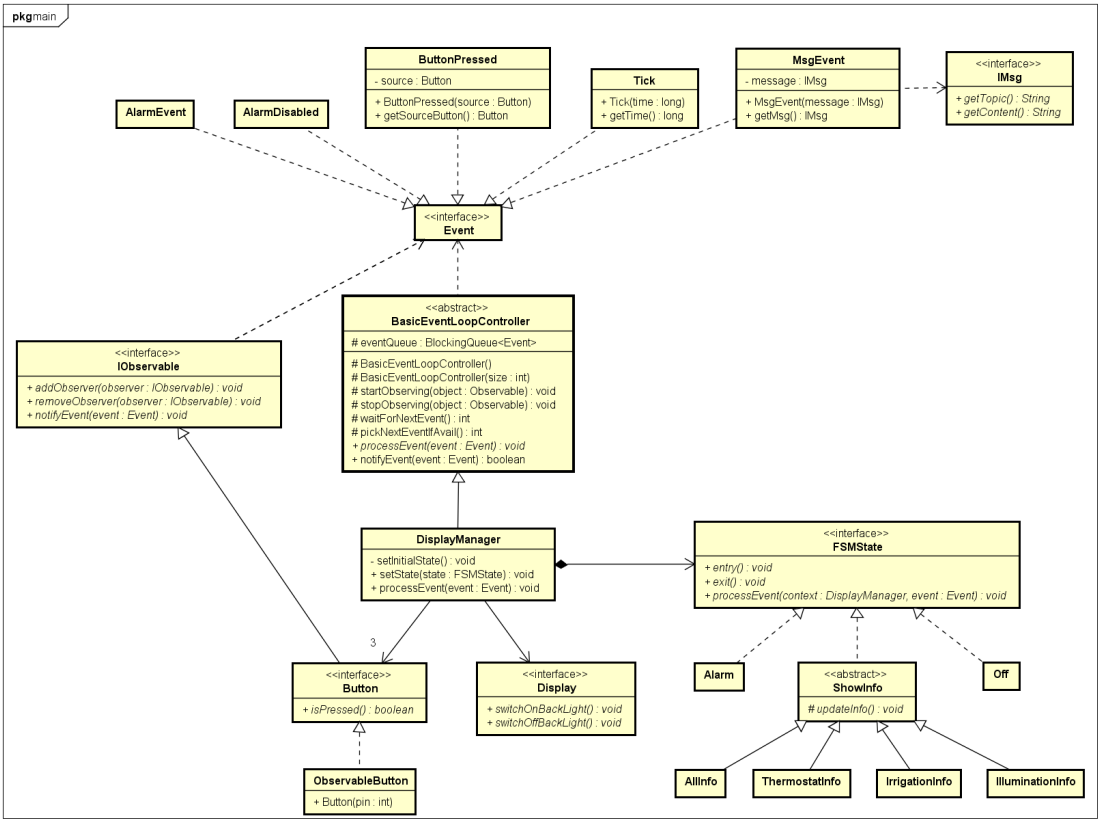

Figura 2.14: Diagrammi delle classi per DisplayManager

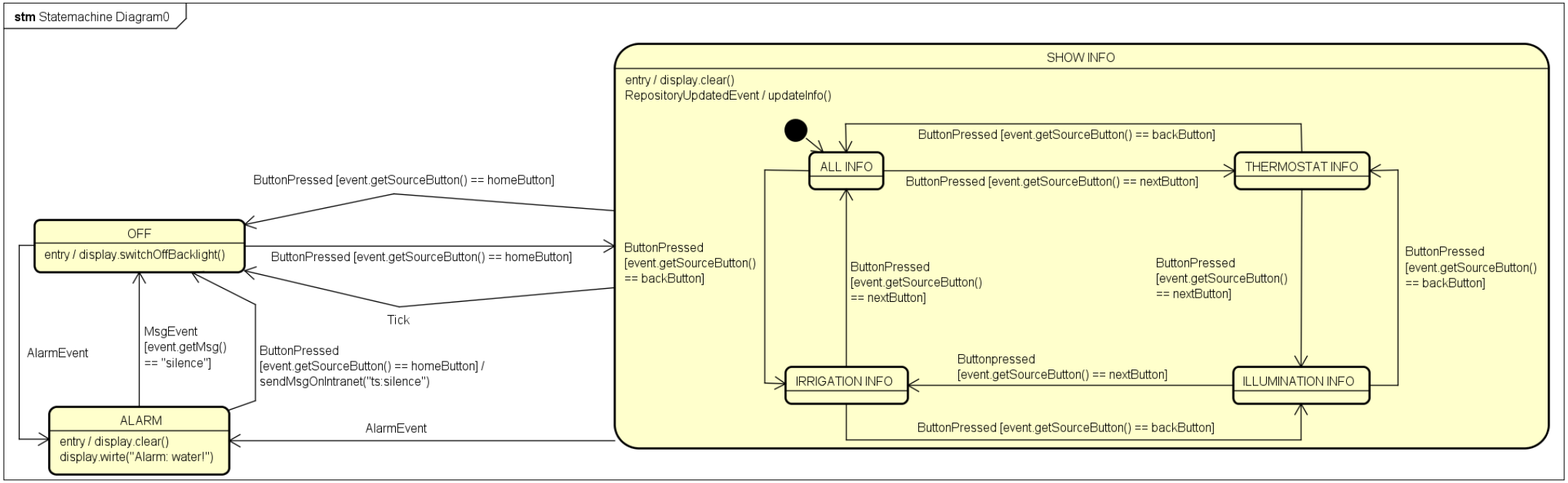

Figura 2.15: Statechart per DisplayManager

2.2.2 Modellazione di oggetti *osservabili*

Per la modellazione dei pulsanti, del timer e del repository ho utilizzato l'Observer Pattern, con una implementazione di default utilizzata dall'ObservableButton dall'ObservableTimer e dall'ObservableRepository, permettendo di disaccopiare il subject dall'observer.
Per quanto riguarda il display, non essendo un oggetto osservabile, ma solo un device privo di stato sul quale vengono mostrate le informazioni, non è stato applicato il suddetto pattern.
Il Repository è un classe che permette l'accesso ai dati, in questo caso memorizzati in ram, che al cambiamento di una singola informazione notifica gli osservatori inviando l'evento *RepositoryUpdated*.

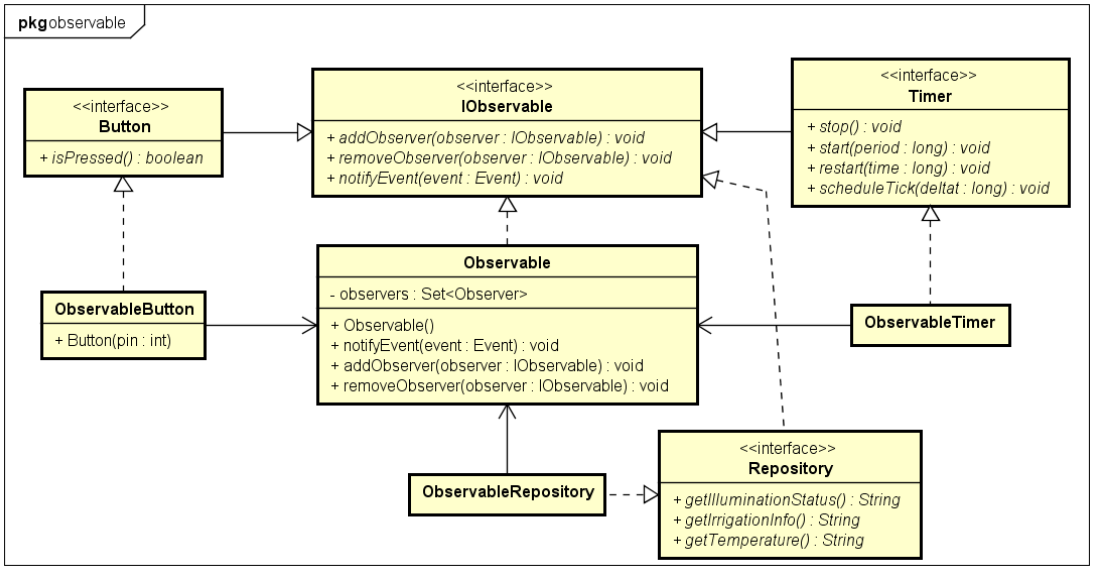

Figura 2.16: Diagramma delle classi per l'applicazione del pattern Observer

Di seguito è riportato il diagramma delle classi che evidenzia la struttura del display.

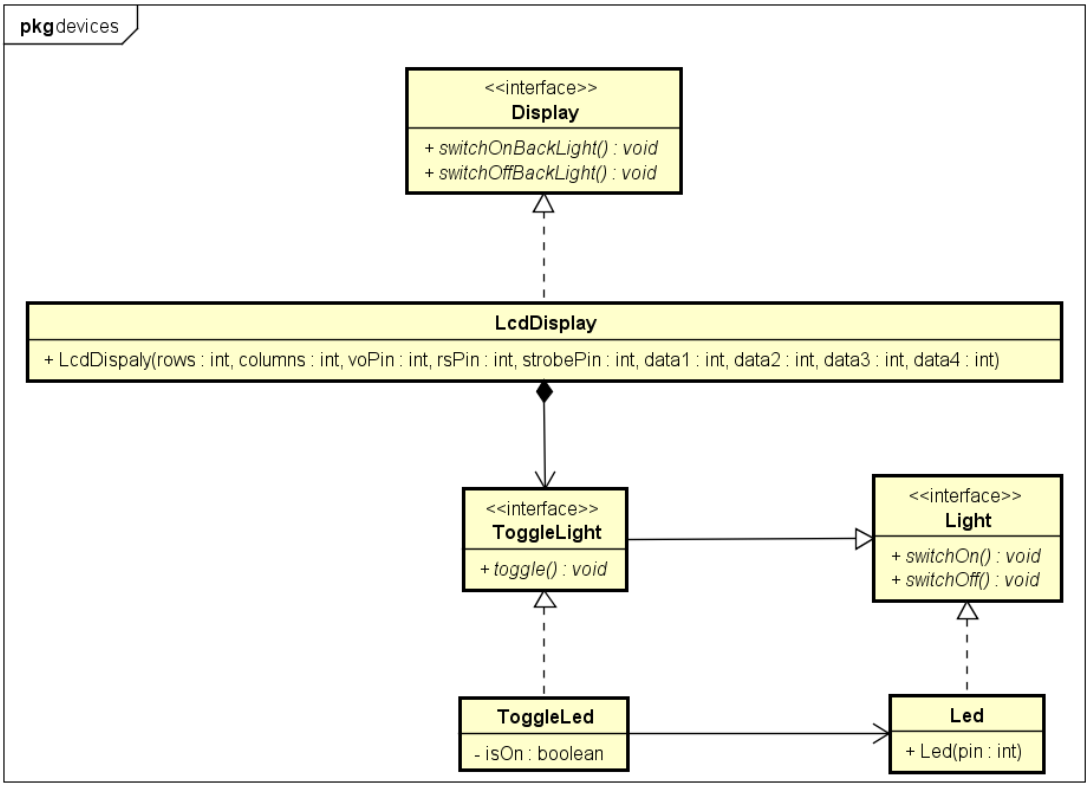

Figura 2.17: Diagramma delle classi per il display

2.2.3 Diagramma delle classi completo

Di seguito è riportato il diagramma delle classi dell'intero sistema, disponibile anche come immagine nel file .zip.

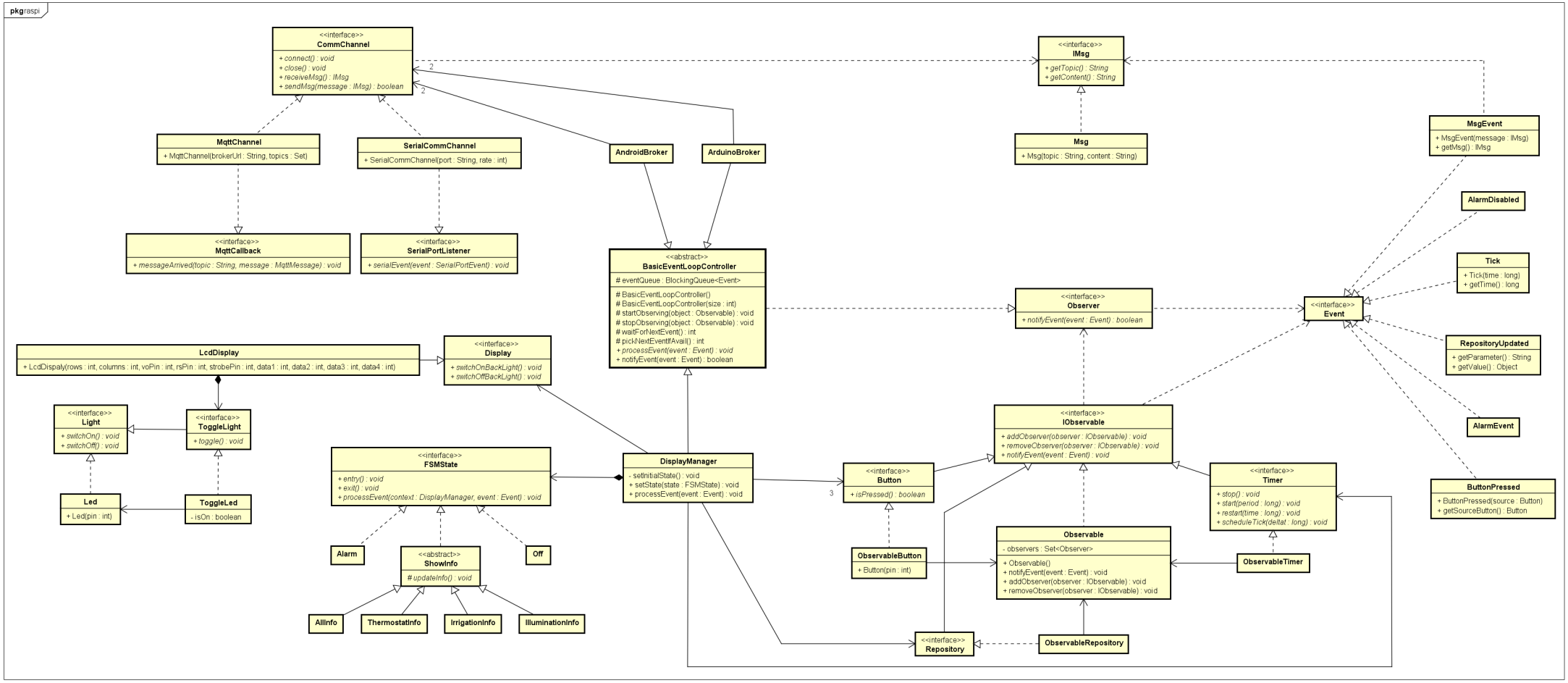

Figura 2.18: Diagramma delle classi completo sull'architettura di Raspberry

Capitolo 3

Progettazione elettronica

La progettazione elettronica è stata realizzata con il tool open source Fritzing[1]. Per ottenere un progetto, che includesse sensori ed attuatori molto simili a quelli realmente utilizzati, ho fatto uso di librerie esterne. La progettazione elettronica è stata poco complessa ma molto delicata, dal momento che, volendo realizzare un prototipo funzionante ed applicabile nella realtà, ho dovuto gestire device che richiedono corrente alternata 220-240V, altri che richiedono corrente continua da 12V e altri ancora da 5V. Il tutto prestando molta attenzione ad evitare di bruciare entrambe le schede, Arduino Uno e Raspberry Pi 1 Model B.

3.1 Circuito elettronico - Arduino

I componenti adottati per la realizzazione della serra sono stati scelti in modo tale da minimizzare l'uso della breadboard e/o i relativi costi. Quindi, anzichè creare un modulo di commutazione *(on-off)* ad-hoc ho utilizzato un modulo con 4 relè già configurato con diodi e resistenze. I diodi sono fondamentali per evitare che la corrente circoli nel senso opposto, danneggiando la scheda; invece, le resistenze sono utili per ridurre l'intensità di corrente. Per cui è stato utilizzato un modulo di 4 relè per collegare i device (pompa ad immersione, asciugacapelli, aereosol, lampada) alla corrente alternata 220-240V, e due transistor npn con due diodi per collegare le ventole alla corrente continua da 12V. Tutti i moduli restanti sono invece stati collegati direttamente alla scheda Arduino, in quanto non richiedono ulteriori componenti. Infine, ho realizzato una "multipresa" per evitare

[1] `http://fritzing.org`

di sguainare i cavi dei device. La Figura 3.2, mostra l'intero progetto elettronico necessario per realizzare la serra.
Per poter collegare i device, che necessitano di corrente alternata ad Arduino, ho utilizzato il modulo con 4 relay JQC-3FF-S-Z della TONGLING, i cui pin Vcc e GND sono stati collegati ai pin Vcc e GND di Arduino, mentre i pin signal-data IN1 e IN2 sono stati collegati ai pin analogici, realizzando così una multipresa 3.1 in cui ogni presa viene comandata separatamente.

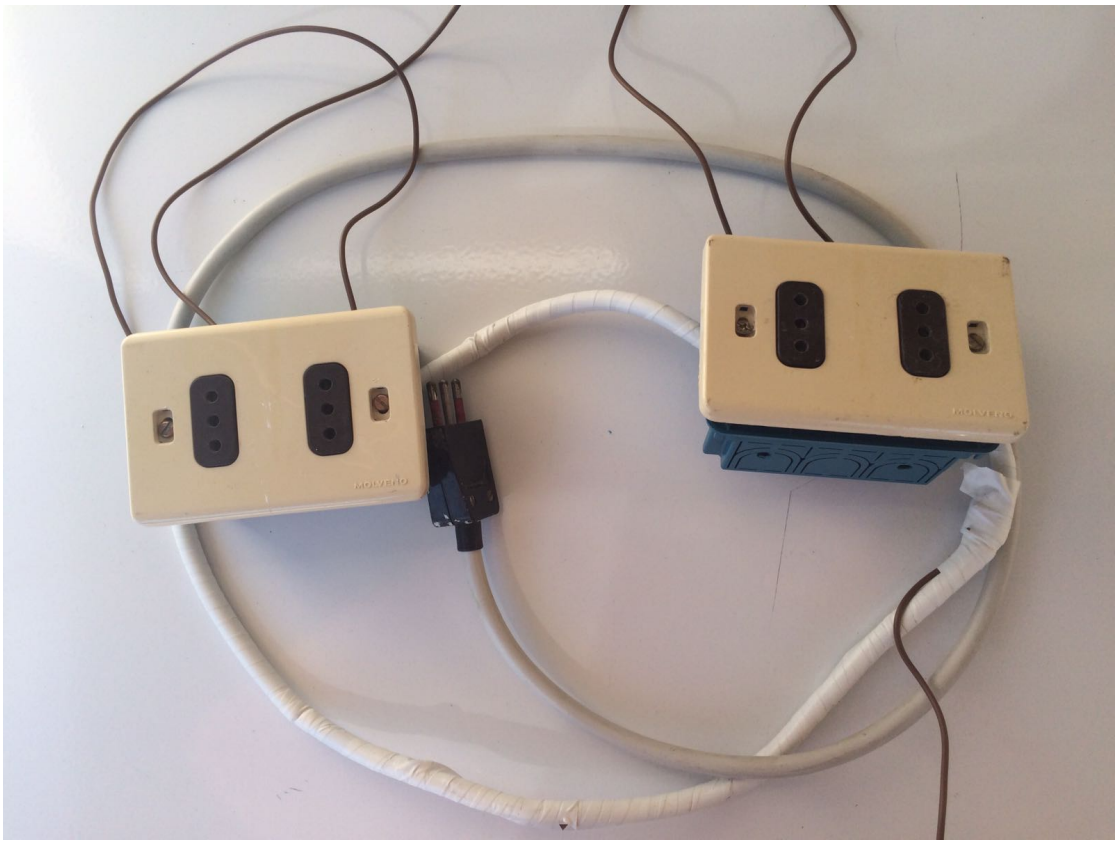

Figura 3.1: Multipresa con modulo relay - vista frontale

Ogni relè ha 3 morsettini: COM (comune), NO (normalmente aperto) e NC (normalmente chiuso). Al COM va inserita l'alimentazione da 220V-240V messa in parallelo agli altri 3 moduli relè in modo tale da utilizzare un'unica alimentazione. Mentre, ai morsettini NO e NC deve essere collegato il cavo della fase, in modo che questa possa essere alimentata.
Nel progetto ho scelto di utilizzare il morsettino NO, in modo tale che il circuito sia sempre aperto (evitando che nella presa circoli corrente). In questo caso, se si volesse chiudere il circuito, si dovrebbe impostare il pin di Arduino a LOW.

Invece, per le due ventole, che richiedono corrente continua da 12V, sono stati utilizzati due transistor, con il collegamento in serie di una resistenza e il collegamento in parallelo di un diodo.

L'umidità del terreno viene rilevata tramite un apposito sensore (FC-28 in figura), costituito da 3 pin, di cui i pin Vcc e GND sono stati collegati ai pin Vcc e GND di Arduino, mentre il pin signal-data è stato connesso al pin analogico di Arduino.
Inoltre, dato che la pompa può essere avviata solo in presenza di acqua nella tanica, è stato usato anche un sensore per rilevare il livello dell'acqua. Tale sensore è stato connesso allo stesso modo del sensore per la rilevazione dell'umidità del terreno.
Infine, è fondamentale la gestione del tempo, dal momento che in alcuni orari non è possibile irrigare, altrimenti le piante potrebbero risentirne. Per cui è stato utilizzato anche un modulo RTC, in modo tale che l'ora venga mantenuta nonostante la scheda Arduino dovesse spegnersi o mancasse la connessione con il raspberry.
Infine, la luce solare viene controllata tramite una fotoresistenza posta all'interno della serra, in modo tale da considerare la luce effettiva che le piante assorbono durante il giorno.

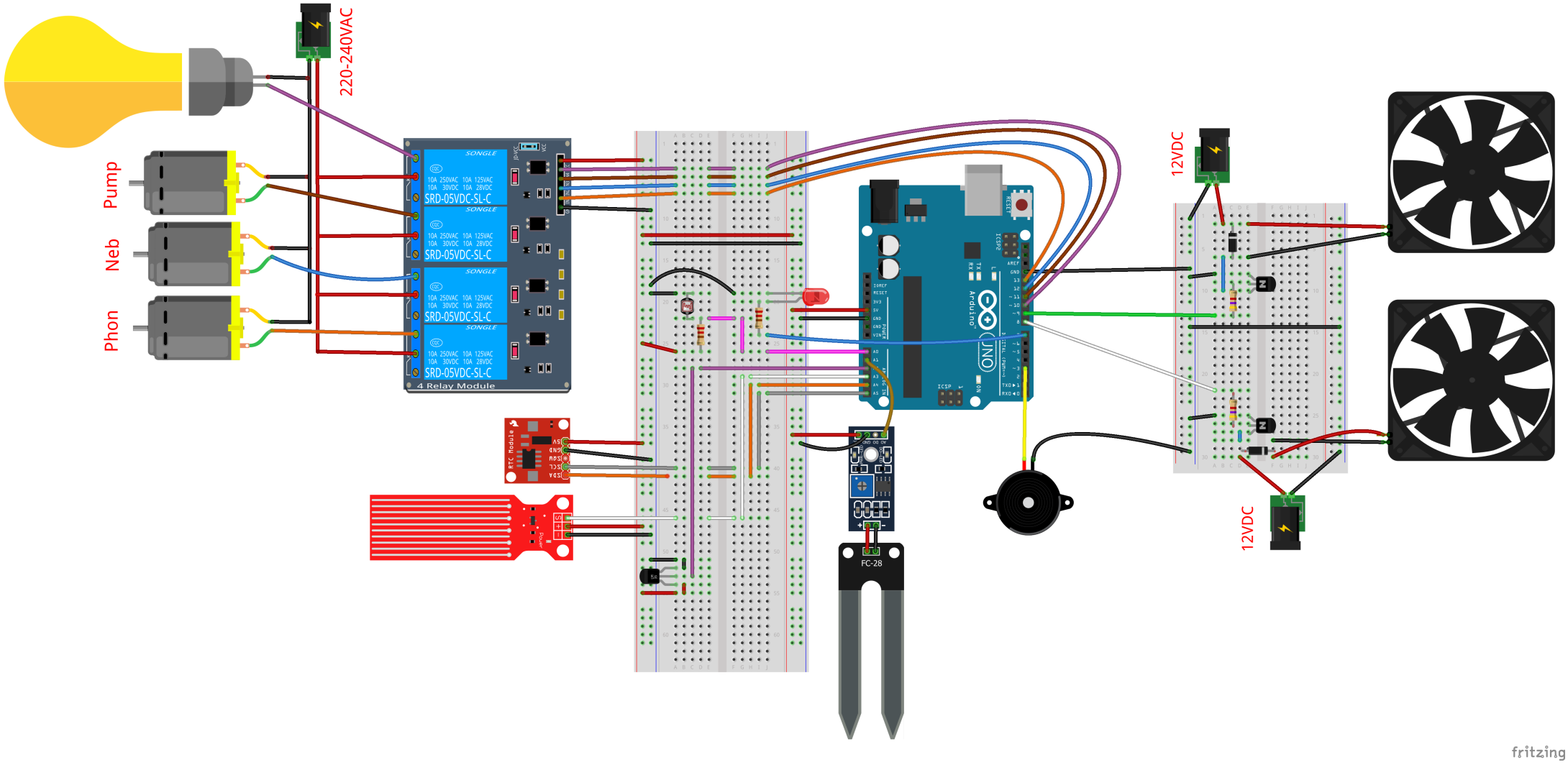

Figura 3.2: Schema del circuito di Arduino

3.2 Circuito elettronico - Raspberry

Il Raspberry è connesso tramite cavo usb con la scheda Arduino e tramite cavo di rete ethernet ad Internet per poter comunicare con lo smartphone. Il circuito del Raspberry è molto semplice, in quanto è connesso solo ad un display lcd 16x2 e a tre semplici pulsanti per mostrare le informazioni. Il progetto elettronico completo è mostrato in Figura 3.3.

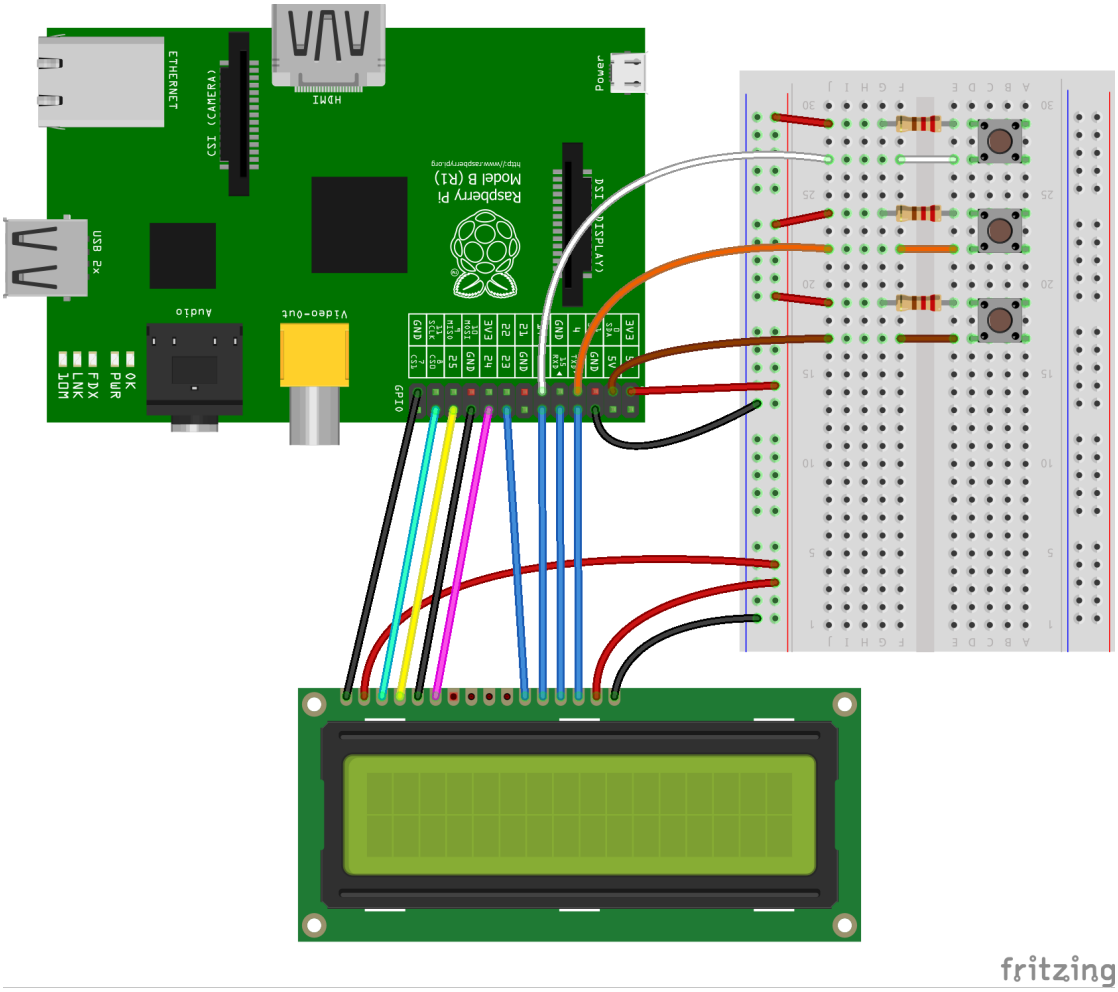

Figura 3.3: Schema del circuito di Raspberry Pi 1 Model B

Conclusioni

In questo progetto sono stati analizzati ed applicati vari argomenti esaminati durante il corso di Sistemi Embedded. Mi posso ritenere molto soddisfatto della progettazione del sistema, al di là della sua riuscita. Il mio personale obiettivo è stato di conoscere gli aspetti dell'IoT e dei sistemi embedded, senza alcuna presunzione di sorta, e di analizzare aspetti legati alla concorrenza.
In conclusione, il sistema non è esenti da limiti, soprattutto nella sua applicabilità alla realtà. Si possono apportare numerose modifiche o miglioramenti, come:

- l'uso del cloud;
- l'interazione con l'utente;
- la sostenibilità del sistema;
- l'uso di algoritmi di intelligenza artificiale;
- altri sistemi che collaborano con Arduino e Raspberry.