Alma Mater Studiorum – Università di Bologna
Campus di Cesena

Scuola di Scienze

Corso di Laurea in Ingegneria e Scienze Informatiche

APPLICAZIONI DI TECNOLOGIE IOT E WEARABLE ALLA DOMOTICA: UN CASO DI STUDIO

Relazione finale in:

PROGRAMMAZIONE DI SISTEMI EMBEDDED

Relatore:
Prof. Alessandro Ricci

Presentata da:
Mattia Capucci

II Sessione di Laurea
Anno Accademico 2014/2015

Ai miei genitori, a mio fratello, alla mia ragazza e tutti i miei familiari e amici che mi hanno sostenuto (e sopportato) in questo periodo.

Indice

Elenco delle figure

Introduzione

Oggigiorno milioni di persone fanno uso di Internet per gli utilizzi più disparati: dalla ricerca di informazioni sul Web al gioco online; dall'invio e ricezione di email all'uso di applicazioni social e tante altre attività.

Mentre milioni di dispositivi ci offrono queste possibilità, un grande passo in avanti sta avvenendo in relazione all'uso di Internet come una piattaforma globale che permetta a oggetti di tutti i giorni di coordinarsi e comunicare tra di loro.

È in quest'ottica che nasce **Internet of Things**, l'Internet delle cose, dove un piccolo oggetto come un braccialetto può avere un grande impatto nel campo medico per il monitoraggio da remoto di parametri vitali o per la localizzazione di pazienti e personale e l'effettuazione di diagnosi da remoto; dove un semplice sensore ad infrarosso può allertarci a distanza di una presenza non autorizzata all'interno della nostra abitazione; dove un'autovettura è in grado di leggere i dati dai sensori distribuiti sulla strada.

Questa tesi vuole ripercorrere gli aspetti fondamentali di Internet of Things, dai sistemi embedded fino alla loro applicazione nella vita odierna, illustrando infine un progetto che mostra come alcune tecnologie IoT e wearable possano integrarsi nella domotica, come per esempio l'utilizzo di uno smartwatch, come Apple Watch, per il controllo dell'abitazione.

Capitolo 1

Internet of Things

1.1 Cenni storici

Una prima definizione di Internet delle cose risale al 1982 nell'Università Carnegie Mellon a Pittsburgh, dove un distributore modificato di Coca Cola diventò il primo apparecchio connesso ad Internet in grado di inviare informazioni sull'inventario e se le bibite, caricate di recente, fossero fresche. Il termine "Internet of Things" fu coniato, però, per la prima volta solo nel 1999 dall'imprenditore inglese Kevin Ashton per descrivere la connessione di oggetti del mondo fisico ad Internet. In un articolo alla RFID Journal scrive:

And that's a big deal. We're physical, and so is our environment. Our economy, society and survival aren't based on ideas or information – they're based on things. You can't eat bits, burn them to stay warm or put them in your gas tank. Ideas and information are important, but things matter much more. Yet today's information technology is so dependent on data originated by people that our computers know more about ideas than things. If we had computers that knew everything there was to know about thing – using data they gathered without any help from us – we would be able to track and count everything, and greatly reduce waste, loss and cost. We would know when things needed replacing, repairing or recalling, and whether they were fresh or past their best.

Kevin mostra i limiti di una tecnologia basata solo sulle idee, su dati generati dagli utenti e vede, invece, le prospettive che possono offrire gli oggetti di tutti i giorni connessi in rete, in grado di fornirci informazioni sul loro stato, permettendoci quindi di sapere quando uno di essi necessita di essere riparato o sostituito.

Nello stesso anno nasce l'Auto - ID Center dove Kevin, insieme a David Brock e Sanjay Sarma, contribuiscono a sviluppare l'EPC (Electronic Product Code), un sistema globale basato su RFID dove gli oggetti sono connessi ad Internet attraverso dei tag RFID. Questo sistema è stato una delle prime tecnologie che hanno reso possibile una rete distribuita di sensori wireless pervasivi.

Una prima problematica nella connessione di un così gran numero di dispositivi a Internet è il limitato spazio di indirizzi offerto da IPv4 (4.3 bilioni). Grazie ad IPv6, lo spazio di indirizzi è stato esteso a 2^{128}, permettendo ad un qualsiasi everyday object di collegarsi in rete offrendo, inoltre, caratteristiche di sicurezza avanzate.

A oggi, il numero di dispositivi connessi è in continua crescita e, secondo Cisco, entro il 2020 giungeremo quota 50 bilioni.

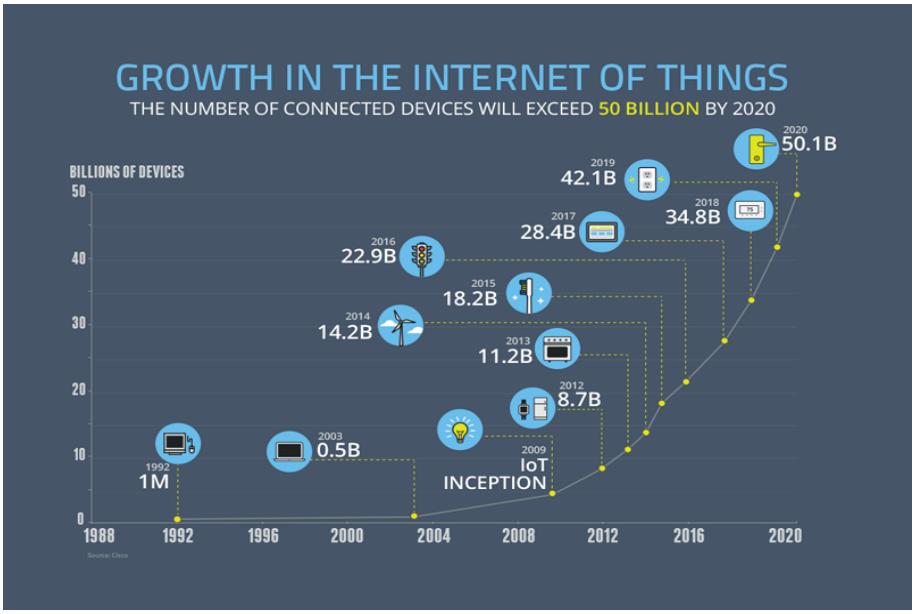

Figura 1.1: Crescita del numero di dispositivi connessi dal 1988 al 2020

1.2 Internet of Things oggi

Oggi una definizione di Internet of Things potrebbe essere la seguente:

"Reti di dispositivi in grado di acquisire informazioni dall'ambiente, computare e quindi agire producendo opportuni effetti, connessi mediante varie tecnologie di rete che permette loro di interagire in modo cooperativo"

Una caratteristica fondamentale per le *things*, quindi, è la capacità di comunicare direttamente o indirettamente con la rete Internet: questo permette loro di gestire in modo opportuno grandi volumi di dati generati dai sensori, che non potrebbero essere memorizzati in locale. Tali informazioni diventano quindi accessibili mediante servizi Internet o Web, permettendo ad altre applicazioni di accedervi e di interagire in base ai dati ricevuti.

I campi di applicazione in cui le things diventano *smart* sono numerosi e questa vastità è dovuta anche dalle potenzialità che oggetti di tutti i giorni possano offrirci se connessi a Internet e/o interconnessi tra di loro. Le discipline in cui Internet of Things ha avuto maggior successo sono le seguenti:

- Medicina
- Domotica
- Avionica
- Industria automobilistica
- Produzione industriale
- Smart City

1.2.1 Le applicazioni di Internet of Things in campo medico

L'aumento di sensori per dispositivi medici ha reso possibile il monitoraggio da remoto della salute di un paziente. Questa rete di sensori, attuatori e altri dispositivi prende il nome di *Internet of Things for Medical Devices* (IoT–MD).

L'IoT–MD crea un ambiente in cui i parametri vitali di un paziente sono trasmessi dai dispositivi medici in una piattaforma cloud, in cui sono memorizzati e analizzati, permettendo di fare diagnosi da remoto e in real-time. Per esempio, un sistema IoT–MD per il trattamento di malattie croniche, come il diabete o l'ipertensione, segue lo stile di vita del paziente, aiutandolo con un piano di assistenza personalizzato, composto da trattamenti, medicazioni e diete.

I vantaggi di un sistema Internet of Things in campo medico sono molteplici:

- Migliora il decorso clinico di un paziente
- Riduce il costo delle cure
- Gestione delle malattie in real-time
- Migliora la qualità della vita

1.2.2 Le applicazioni di Internet of Things in un ambiente domestico

Un qualsiasi oggetto domestico di uso comune, come una lampada, un termostato o un forno, è candidato a far parte di Internet of Things. Avere una lampadina collegata in rete ci permette non solo di accenderla e spegnerla a distanza, ma ci permette di decidere quando accenderla, per quanto tempo e quanta luminosità debba emettere.

L'idea di sfruttare Internet of Things in un contesto domestico è quella di avere il pieno controllo delle risorse al suo interno e ricevere informazioni da esse. Un frigorifero, ad esempio, può memorizzare gli alimenti al suo interno e avvisarci nel momento in cui uno di essi stia per scadere; un forno può essere acceso a distanza per procedere alla cottura.

Internet of Things trova spazio anche nella sicurezza della casa, dove una rete di sensori di movimento e telecamere monitorano le stanze e inviano tutte le informazioni al Cloud, rendendo possibile il controllo dello stato di sicurezza da remoto.

1.2.3 Internet of Things e l'avionica

Con il termine *avionica* si indicano tutti gli equipaggiamenti elettronici installati a bordo degli aeromobili e preposti al pilotaggio. L'avionica include i sistemi di navigazione e comunicazione, autopiloti e sistemi di condotta di volo.

Ogni sistema dell'avionica può essere rivoluzionato da Internet of Things. Per esempio, può aiutare le linee aeree a collezionare grandi volumi di dati, per poi tradurli in informazioni che possono essere utilizzate per determinare lo stato o le performance di un particolare sistema.

I sensori sono distribuiti in tutto l'aereo, monitorando, per esempio, parametri come il consumo di carburante nel motore. Quando l'aereo è atterrato, queste informazioni possono essere recuperate dallo staff a terra, che può intraprendere azioni tempestive in caso di malfunzionamenti per riportare l'aereo in servizio il prima possibile.

Entro cinque anni sarà possibile per lo staff di terra avere accesso in real time al monitoraggio dell'aereo in volo, permettendo loro di aver un accesso continuo alle informazioni sulle performance dello stesso. Se, per esempio, ci fosse una falla nel motore, entrerebbe in azione un particolare sistema che eseguirebbe tutte le funzioni necessarie per permette la conclusione del volo in sicurezza.

1.2.4 Internet of Things nell'industria automobilistica

L'automobile, ideata come semplice mezzo di trasporto, sta per essere rivoluzionata da Internet of Things. Secondo una previsione di GSMA, associazione globale di operatori wireless, ogni autovettura avrà entro il 2025 la possibilità di collegarsi ad Internet. Questo estenderà le potenzialità dell'automobile, tra cui:

- Eseguire applicazioni nel sistema di intrattenimento del veicolo
- Associare un telefono al veicolo per averne accesso da remoto
- Visualizzare tutti i dati di diagnostica generati dall'autovettura

Attualmente gli utenti possono già visualizzare lo stato della batteria, trovare il punto esatto in cui hanno parcheggiato e attivare il climatizzatore da remoto. Ma come afferma Liz Kerton, direttore esecutivo dell'*Autotech Council*, un'autovettura è composta dal 90% da software e dal 10% da hardware e, attualmente, abbiamo raggiunto solo il 70%. Molte altre funzionalità, quindi, devono ancora essere sviluppate.

1.2.5 Internet of Things nella produzione industriale

Internet of Things sta rivoluzionando la produzione industriale così come la conosciamo. Le fabbriche e gli impianti, che sono collegati in Internet, sono più efficienti e produttivi rispetto alla controparte non connessa.

I sistemi industriali basati su Internet of Things, infatti, permettono una rapida produzione di nuovi prodotti, una veloce risposta alle richieste di prodotti e un'ottimizzazione real time del processo di sviluppo.

Gli impianti industriali, se connessi con una piattaforma cloud, salvano i dati generati, rendendoli disponibili da remoto e permettendo di avere una panoramica di tutto il sistema e della sua stabilità ed efficienza. Questo porta ad un miglioramento nell'utilizzo e ottimizzazione delle risorse, nonché ad una riduzione dei costi.

1.2.6 Smart City

Una città può essere definita come *Smart City* se gestisce in modo intelligente le attività economiche, la mobilità, le risorse ambientali, le relazioni tra le persone, le politiche dell'abitare e il metodo di amministrazione. In altre parole, una città può essere definita "smart" quando gli investimenti in capitale umano e sociale e nelle infrastrutture tradizionali (trasporti) e moderne (ICT) alimentano uno sviluppo economico sostenibile ed una elevata qualità della vita, con una gestione saggia delle risorse naturali, attraverso un metodo di governo partecipativo.

In questa visione moderna della città ha un ruolo centrale Internet of Things. Introducendo una rete di nodi sensori in grado di misurare molti parametri, è possibile avere una gestione più efficiente della città. I dati sono resi accessibili, in modalità wireless e in tempo reale, ai cittadini o alle autorità competenti.

Per esempio, il traffico veicolare può essere monitorato in modo tale da modificare di conseguenza l'intensità delle luci della città oppure gli automobilisti possono ricevere informazioni real-time riguardo la presenza di parcheggi liberi nelle vicinanze o di strade alternative in caso di forti ingorghi sulla strada.

Allo stesso tempo una parte dei sensori è adibita al controllo dell'inquinamento atmosferico di quella specifica città, mantenendo aggiornati i dati rilevati e facendo scattare un allarme in caso di livelli troppo alti.

Internet of things, quindi, non solo aumenta le funzionalità di una città ma riesce anche a migliorarne la qualità della vita in base alle informazioni raccolte.

Capitolo 2

Ai vertici di Internet of Things: i Sistemi Embedded

Nel capitolo precedente abbiamo visto il concetto di Internet of Things, il grande impatto che avuto oggigiorno e le diverse applicazioni alle quali si può applicare una tale rivoluzione del modo di vedere Internet.

Immaginando di pensare a Internet of Things come un grafo, addentriamoci ora nella spiegazione di tutto ciò che sta ai suoi vertici: i sistemi embedded.

2.1 Definizione e caratteristiche di un sistema embedded

Si definiscono sistemi embedded i sistemi di elaborazione *special-purpose* – ovvero che svolgono una specifica funzione o compito – incorporati in sistemi o dispositivi elettronici di diverse dimensioni. Tipicamente costituiti da una parte hardware e da una software, sono pensati per funzionare per periodi estesi di tempo, processando ripetutamente dati in input e generando dati di output. Sono caratterizzati da risorse hardware limitate ma il loro compito è realizzato secondo una progettazione orientata all'efficienza, ovvero maggior velocità possibile e consumi molto bassi.

Poiché deve svolgere un determinato compito in un ambiente, spesso in situazioni critiche, per un sistema embedded sono richieste le seguenti caratteristiche:

- **Affidabilità**: il sistema deve essere affidabile per un lungo periodo di tempo
- **Adattabilità**: il sistema deve essere facilmente adattabile a diverse situazioni
- **Sicurezza**: il sistema non deve arrecare danni agli utenti o all'ambiente in cui si trova
- **Reattività**: il sistema deve essere celere nel reagire a stimoli provenienti dall'ambiente ed eseguire azioni in real-time senza ritardi

Possiamo suddividere i sistemi embedded in quattro categorie:

- **Sistemi embedded stand-alone**: questi sistemi non necessitano di un sistema host come un computer. Acquisiscono input analogici o digitali dalle proprie porte di input, convertono e processano i dati acquisiti e restituiscono il risultato in output.
- **Sistemi embedded real-time**: questi sistemi restituiscono i dati di output in uno specifico momento oppure rispettano severamente la propria dead line per il completamento del task.
- **Sistemi embedded networked**: questi sistemi sono in relazione ad una rete con opportune interfacce per l'accesso delle risorse. La rete può essere locale (LAN), geografica (WAN) oppure Internet.
- **Sistemi embedded mobile**: questi sistemi sono i dispositivi portatili come cellulari, macchine fotografiche digitali, lettori MP3.

2.2 Il livello hardware di un sistema embedded

Un sistema embedded è composto da una serie di componenti hardware gestiti dalla CPU, che è il cuore dei microprocessori e dei microcontrollori.

Un microcontrollore è un dispositivo elettronico che integra su un singolo chip la memoria, le porte di I/O, i timer e altri componenti. Molti sistemi vengono costruiti su microcontrollori in quanto la velocità di esecuzione delle operazioni integrate è nettamente maggiore rispetto a quelle eseguite via software dai microprocessori.

A questi si aggiungono i microcontrollori *Single Board* che incorporano in un'unica scheda il microcontrollore e tutta la circuteria necessaria per eseguire dei compiti di controllo.

Vediamo ora nel dettaglio le specifiche hardware di questi dispositivi, dall'elettronica di base alla gestione dell'I/O in un sistema embedded.

2.2.1 Elettronica di un sistema embedded

Alla base di un sistema embedded abbiamo i circuiti digitali, ovvero circuiti elettronici il cui funzionamento è basato su un numero finito di livelli di tensione elettrica. La maggior parte dei circuiti digitali lavora con due livelli di tensione:

- 0 volt, detto anche LOW, che rappresenta il valore logico binario 0
- 3-5 volt, detto anche HIGH, che rappresenta il valore logico binario 1

Il componente fondamentale dei circuiti elettronici moderni è il *transistor*. Esso è composto da un materiale semiconduttore al quale sono applicati tre terminali (source, gate, drain) che lo collegano al circuito esterno. L'applicazione di una tensione elettrica superiore alla soglia fa fluire la corrente dalla sorgente al drain: in questo caso il transistor è detto essere ON. Se il gate è OFF, invece, la corrente non transita. Attualmente, per la progettazione

di circuiti integrati, vengono utilizzati i cosiddetti CMOS, una struttura circuitale costituita dalla serie di una rete di *Pull-Up* e di *Pull-Down*, in cui la prima si occupa di replicare correttamente il livello logico alto mentre la seconda gestisce il livello logico basso.

Ogni circuito deve essere alimentato e necessita di due pin a questo scopo:

- **Power pin**: connesso ad un segnale HIGH
- **Ground pin**: connesso ad un segnale LOW

2.2.2 La CPU e la memoria di un sistema embedded

La CPU, il cuore di un sistema embedded, è composta dall'ALU (Arithmetic Logic Unit), dalla CU (Control Unit) e da una serie di registri interni.

L'ALU è responsabile di tutte le operazioni matematiche (somma, differenza, prodotto, quoziente), delle operazioni logiche (AND, OR) e delle operazioni di scorrimento binarie.

La sincronizzazione e la sequenza di tutte le operazioni della CPU sono controllate dalla CU. Questa è responsabile nel dirigere il flusso di istruzioni e dati all'interno della CPU ed esegue step by step le istruzioni del programma.

Il funzionamento di una CPU è definito dalla macchina di Von Neumann, che esplica quello che viene anche chiamato ciclo *fetch–decode–execute*. In un primo momento avviene il *fetch*, ovvero il caricamento dalla memoria dell'istruzione da eseguire; successivamente viene effettuata una decodifica dell'istruzione (*decode*); infine l'istruzione viene eseguita (*execute*), eventualmente aggiornando i registri e scrivendo dati in memoria.

In un sistema embedded, la CPU non si ferma mai ed esegue istruzioni continuamente. La scelta di una CPU, durante la progettazione di sistemi embedded, è influenzata da diversi fattori quali la grandezza massima in bit di un singolo operando per l'ALU (8, 16, 32, 64 bit) e la frequenza di clock.

Un altro aspetto importante di una CPU sono le interruzioni, ovvero il verificarsi di un evento che richiede l'attenzione del processore. Al verificarsi

di un'interruzione la CPU interrompe il programma in esecuzione ed esegue un sottoprogramma, chiamato *routine di interrupt*, allocato ad uno specifico indirizzo di memoria. Terminata la routine d'interrupt, la CPU torna al programma precedentemente interrotto e ne prosegue l'esecuzione. Le richieste di interruzione vengono fatte a dei precisi pin chiamati IRQ (Interrupt Request).

Per quanto riguarda la memoria, in un sistema embedded può essere integrata nel chip o meno. Nel primo caso, la memoria è molto più veloce ma, allo stesso tempo, di capienza nettamente inferiore rispetto al secondo caso. Generalmente i dati vengono salvati nella RAM (Random Access Memory), la memoria principale e volatile, mentre il programma è salvato nella ROM (Read Only Memory), memoria non volatile e utilizzabile solo in lettura.

2.2.3 Gestione dell'I/O in un sistema embedded

Il GPIO (General Purpose Input Output) è un'interfaccia disponibile nei moderni microcontrollori che fornisce un accesso immediato alle proprietà interne dei dispositivi. Generalmente in un microcontrollore sono presenti diversi pin di GPIO che possono essere programmati in input, dove i dati provenienti da una sorgente esterna sono immessi nel sistema per essere manipolati, oppure in output, dove i dati, formattati correttamente, possono essere inviati ai dispositivi esterni. Alcuni pin di GPIO possono essere configurati come linee di interrupt per segnalare alla CPU di eseguire una specifica routine.

I segnali ricevuti su un pin possono essere o digitali o analogici. Nel secondo caso, poiché un sistema di elaborazione lavora solo con valori digitali, è richiesta una conversione. Questa viene effettuata dall'ADC (Analog-to-Digital), che mappa il valore continuo in un valore discreto.

Ai pin di GPIO si aggiungono le interfacce seriali. Fra le più utilizzate nei sistemi embedded ritroviamo:

- USB

- I^2C
- SPI

USB

L' USB (Universal Serial Bus) è uno standard di comunicazione seriale, progettato per connettere più periferiche con un'unica interfaccia standardizzata e permettere il *Plug and Play*, ovvero la possibilità di connettere e disconnettere un device senza spegnere il computer. L'ultima versione disponibile è la 3.0 e permette una velocità di trasferimento teorica pari a 4,8 Gbit/s. USB ha un'architettura master-slave, in cui tutti i dispositivi rispondo ai comandi di un host.

I^2C

I^2C è un sistema di comunicazione seriale bifilare utilizzato tra circuiti integrati. Il protocollo ha una struttura master-slave in cui la comunicazione viene cominciata sempre dal master e gli slave possono solo rispondere. Ogni slave ha un proprio ID in modo tale che possa reagire quando viene richiesta la sua attenzione da parte del master. I^2C utilizza due vie per la comunicazione:

- Un segnale di clock (SCL - Serial Clock Line) per sincronizzare la comunicazione
- Una linea bidirezionale (SDA - Serial Data Line) per inviare e ricevere i dati dal master agli slave

SPI

SPI è un sistema di comunicazione tra un microcontrollore e altri circuiti integrati o tra più microcontrollori. Anch'esso ha un'architettura master-slave, nel quale il master decide quando iniziare e terminare la comunicazione ed è sincrono, per la presenza di un clock che coordina la trasmissione e ricezione dei singoli bit e determina la velocità di trasmissione. A differenza

di I^2C, SPI è di tipo full-duplex, in quanto la comunicazione può avvenire contemporaneamente in trasmissione e ricezione.

2.3 Il livello software di un sistema embedded

Il software in un sistema embedded comprende tutte le linee di codice che permettono di controllare dispositivi o macchine che non sono solitamente pensati come dei computer. Una caratteristica che li distingue è che non tutte le funzioni sono controllate da una persona, bensì da interfacce del dispositivo stesso. Il software sviluppato nei sistemi embedded è, quindi, strettamente legato ai requisiti hardware e alle funzionalità di cui dispone.

Un'altra caratteristica è la possibilità di installare un sistema operativo nei dispositivi compatibili. Si tratta di sistemi compatti e efficienti nell'uso delle risorse, offrendo delle possibilità che i dispositivi non compatibili non possono sfruttare. Ad esempio, il sistema operativo ha la funzione di interfaccia tra l'hardware su cui gira e le applicazioni software, fornendo un livello di astrazione che nasconde alle applicazioni software le specifiche del processore.

2.4 Casi di studio: Arduino e Raspberry Pi

Arduino e Raspberry Pi sono fra le principali piattaforme adibite alla realizzazione di sistemi embedded. Entrambi sono delle *board* con installate una serie di componenti quali pin di GPIO e l'interfaccia seriale USB.

Analizziamo nel dettaglio questi due dispositivi.

2.4.1 Arduino

Arduino è una scheda elettronica di piccole dimensioni, con un microcontrollore e circuteria di contorno. Integra alcuni pin connessi alle porte di I/O, un regolatore di tensione e una interfaccia USB per la comunicazione seriale. Progettato in Italia nel 2005, con lo scopo di creare un dispositivo

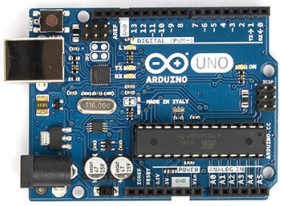

Figura 2.1: Arduino Uno

per il controllo che fosse più economico rispetto ad altri prodotti disponibili sul mercato, nel 2008 contava già più di 50000 dispositivi venduti. A oggi sono stati realizzati 16 versioni dell'hardware di Arduino, ognuna pensata per funzionare in un determinato contesto come Internet of things e i wearable.

Il framework di riferimento per la programmazione in Arduino si chiama *Wiring*, basato sul linguaggio C/C++ con l'aggiunta di alcune librerie per accedere e interagire con l'hardware. È nativamente supportato da Arduino IDE.

Un programma in Wiring contiene due procedure principali:

- **setup()**: funzione che viene eseguita una sola volta, all'avvio del programma, che può essere utilizzata per definire delle impostazioni del programma
- **loop()**: funzione che viene richiamata continuamente, fino allo spegnimento del dispositivo.

```
void setup ()
{
    pinMode(8, OUTPUT);
}

void loop ()
{
    digitalWrite(8, HIGH);
```

```
    delay(500);
    digitalWrite(8, LOW);
    delay(500);
}
```

Listing 2.1: Esempio di codice in Wiring

2.4.2 Raspberry Pi

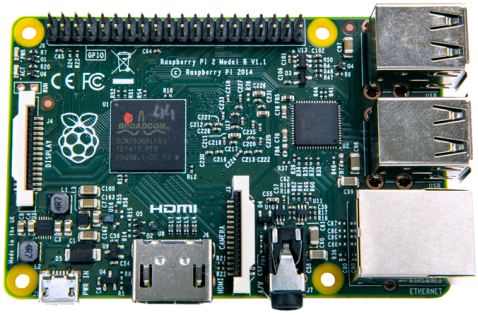

Figura 2.2: Raspberry Pi 2

Raspberry Pi è un *Single-Board-Computer* (calcolatore implementato su una scheda elettronica) sviluppato nel Regno Unito dalla Raspberry Pi Foundation.

Attualmente ne esistono cinque versioni:

- Raspberry Pi A
- Raspberry Pi A+
- Raspberry Pi B
- Raspberry Pi B+
- Raspberry Pi 2

L'ultima versione integra 40 pin di GPIO, programmabili in input e in output, 4 porte USB, per la comunicazione seriale, una porta Ethernet, per il collegamento alla rete, una porta HDMI, un processore Cortex A7 900Mhz e 1GB di memoria RAM.

La caratteristica fondamentale di Raspberry Pi è la possibilità di avviare un sistema operativo tramite micro-SD. Questo permette la prototipazione di sistemi embedded avanzati, nei quali le soluzioni basate su microcontrollori non risultato sufficienti. Raspberry Pi usa principalmente Linux Kernel come OS, in particolare la distribuzione più diffusa è *Raspbian*.

La programmazione I/O su Raspberry Pi può essere effettuata tramite linguaggi di programmazione come C, Java, Python, Bash. Sono state sviluppate, inoltre, delle librerie che facilitano il controllo dei pin su Raspberry Pi.

Ad esempio, Wiring-Pi è una libreria, sviluppata in C, che effettua un porting di Wiring utilizzato in Arduino. Rende, quindi, intuitivo lo sviluppo su Raspberry Pi avendo le conoscenze di Wiring in Arduino.

Un'altra libreria disponibile è Pi4J, che fornisce una serie di API in Java, orientate agli oggetti, che astraggono le funzionalità I/O di Raspberry Pi, consentendone un accesso completo agli sviluppatori.

```
#include <wiringPi.h>
int main (void)
{
  wiringPiSetup ();
  pinMode (0, OUTPUT);
  for (;;)
  {
    digitalWrite (0, HIGH);
    delay (500);
    digitalWrite (0, LOW);
    delay (500);
  }
  return 0;
}
```

Listing 2.2: Esempio di codice in Wiring-pi

```
public static void main(String[] args) throws
   InterruptedException {
    final GpioController gpio = GpioFactory.getInstance
       ();
    final GpioPinDigitalOutput pin = gpio.
       provisionDigitalOutputPin(RaspiPin.GPIO_01,
       PinState.HIGH);
    pin.setShutdownOptions(true, PinState.LOW,
       PinPullResistance.OFF);
    Thread.sleep(10000);
    gpio.shutdown();
}
```

Listing 2.3: Esempio di codice in Pi4J

Capitolo 3

Un caso specifico di sistemi embedded: i Wearable

I wearable rappresentano tutti quei dispositivi elettronici che possono essere indossati da una persona e che hanno la possibilità - diretta o indiretta - di avere accesso a Internet. Smartwatch, occhiali smart, fitness tracker sono tutti esempi di wearable che stanno evolvendo il loro mercato negli ultimi anni. Nel 2014, il numero di dispositivi wearable venduti ammontava a 19 milioni e, secondo Abi Research, questo valore aumenterà fino a 485 milioni entro il 2018.

Molte discipline, da quelle sportive a quelle mediche, per esempio, hanno visto grandi possibilità di innovazione nell'uso di device wearable. Una prova è data dall'uso di braccialetti che monitorano il battito cardiaco e verificano la presenza di anomalie oppure calcolano la distanza percorsa e le calorie consumate. Queste sono solo alcune delle numerose funzionalità che i wearable possono offrirci.

Possiamo dividere i wearable in tre macro categorie:

- **Complex Accessories**: appartengono a questa categoria quei dispositivi che richiedono la connessione ad uno smartphone per essere pienamente operativi. Ne fanno parte, ad esempio, la gran parte dei braccialetti per il controllo dell'attività sportiva.

- **Smart Accessories**: appartengono a questa categoria quei dispositivi che hanno la possibilità di installare app o software di terze parti, potendo integrare ulteriori funzionalità. È comunque necessario il collegamento a uno smartphone o tablet connesso a internet.

- **Smart Wearables**: appartengono a questa categoria quei dispositivi che possono funzionare in piena autonomia, collegandosi a Internet senza la necessità di appoggiarsi a altri device.

3.1 I wearable nel fitness e nello sport

Nello sport e nel benessere della persona i wearable hanno trovato larga scala di utilizzo, per la loro semplicità e praticità.

Figura 3.1: Xiaomi Mi Band

Un esempio è dato da Xiaomi Mi Band, un particolare braccialetto che è in grado di contare i passi, segnalare le calorie bruciate, i chilometri percorsi, il tempo trascorso a passeggiare e a correre; inoltre permette di monitorare il sonno, precisando le ore di sonno profondo e leggero, l'ora in cui ci si è addormentati e quella in cui ci si è svegliati. Il dispositivo comunica poi con un'app dedicata su smartphone, dalla quale l'utente potrà visionare i propri dati e statistiche.

Non solo Xiaomi, ma anche la Nike, famosa in tutta il mondo nella produzione di scarpe, entra nell'ottica dei wearable con il sensore Nike+. Questo sensore è pensato, in particolare, per i corridori e tiene traccia dei tempi, della distanza e altri dati relativi alle corse. Una volta inserito sotto al sot-

Figura 3.2: Sensore Nike+

topiede della scarpa, il sensore è in grado di registrare tutti i dati relativi ad una corsa e inviarli ad uno smartphone associato ad esso.

3.2 I wearable in campo medico

Nell'ambito medico i wearable mirano principalmente alla salvaguardia della salute degli utenti, monitorando, per esempio, i loro parametri vitali.

Figura 3.3: HealthPatch-MD

Un esempio è dato da HealthPatch-MD, un dispositivo che si applica come un cerotto nel petto di un paziente. Il dispositivo è in grado di monitorare il battito cardiaco ed eventuali variazioni, la frequenza respiratoria, la temperatura corporea e la postura. Inoltre, processando migliaia di dati al minuto, algoritmi specifici analizzano i dati biometrici per avere un quadro dettagliato e preciso del paziente. HealthPatch-MD è pensato in un'ottica Internet of Things: infatti, integra al suo interno un chip wireless e uno Bluetooth Low Energy, in modo tale da collegarsi con lo smartphone e inviare ai server tutti i dati da analizzare.

3.3 Gli smart glasses

Gli occhiali smart, meglio noti come *smart glasses*, sono dispositivi wearable che aggiungono informazioni a quello che l'utente che li indossa vede. Solitamente sono costituiti da un display trasparente che permette di proiettare immagini o informazioni attorno alla realtà che circonda l'utente.

Gli smart glasses possono collezionare informazioni tramite i sensori installati in essi e usufruiscono della connettività tramite Bluetooth o Wi-Fi. Inoltre possono tener traccia della posizione tramite un chip GPS.

Fra le aziende che si sono interessate nello sviluppo di smart glasses abbiamo Google e Microsoft, rispettivamente con Google Glass e Microsoft HoloLens.

3.3.1 Google Glass

I Google Glass sono una tipologia di tecnologia wearable tramite la quale l'utente che li indossa visualizza informazioni ed esegue azioni tramite comandi vocali. Inoltre il dispositivo è dotato di un touchpad che permette di controllarlo tramite un'interfaccia *timeline* mostrata nello schermo.

Figura 3.4: Google Glass

Permette di scattare foto, registrare video e di eseguire applicazioni di terze parti. Fra le applicazioni che Google ha reso compatibili con Google Glass sono presenti Google Now, Google Maps, Google+ e Gmail. Molti sviluppatori, in seguito, hanno sviluppato applicazioni per il riconoscimento facciale, manipolazione di foto e pubblicazione sui maggiori social network di successo, come Facebook e Twitter.

3.3.2 Microsoft HoloLens

Microsoft HoloLens è una tipologia di device wearable progettato per sfruttare le potenzialità di Windows Holographic, una piattaforma di realtà aumentata ideata da Microsoft.

Figura 3.5: Microsoft HoloLens

Le lenti utilizzano sensori avanzanti, un sistema di scansione spaziale dei suoni e un display ottico 3D che permettono all'utente di poter interagire con un'interfaccia olografica mediante lo sguardo, la voce o i gesti delle mani.

Le applicazioni di interesse, in via di sviluppo, compatibili con Microsoft HoloLens sono HoloStudio, un'applicazione di modellazione 3D, Skype e On-Sight, realizzato in collaborazione con la NASA, permette di integrare i dati provenienti dalle sonde presenti sul pianeta Marte al fine di poter riprodurre una simulazione dell'ambiente di riferimento, interagendovi.

3.4 Gli smartwatch

Gli smartwatch hanno acquisito un notevole interesse negli ultimi anni da moltissime aziende. Si tratta di orologi che, oltre ad eseguire le funzioni di base, hanno la possibilità di eseguire applicazioni e, nella maggioranza dei casi, di comunicare con il proprio smartphone. Molti di essi dispongono di uno schermo touch, permettendo di interagire con l'orologio e visualizzare le informazioni di interesse.

Gli smartwatch di maggior successo, ad oggi, sono Apple Watch e gli orologi montanti come sistema operativo Android Wear.

3.4.1 Apple Watch

Apple Watch è il primo smartwatch prodotto da Apple e rilasciato al pubblico nell'Aprile del 2015.

Figura 3.6: Apple Watch

Il dispositivo presenta diversi sensori quali accelerometro, giroscopio, barometro e sensore di battito cardiaco. Inoltre, per quanto riguarda la connettività, è dotato di un chip NFC, Bluetooth e Wi-Fi. Può eseguire applicazioni di terze parti (regolarmente approvate su AppStore) e, per usufruire a pieno delle sue funzionalità, richiede una connessione con un iPhone. Permette, inoltre, di effettuare pagamenti con Apple Pay tramite NFC.

Il sistema operativo di cui è dotato prende il nome di *watchOS* , la cui ultima versione è la 2.0 (attualmente ancora in fase di beta-testing).

Per quanto concerne la programmazione su questo dispositivo, Apple ha reso disponibile una libreria chiamata *WatchKit*, integrata nell'IDE di sviluppo Xcode. Il linguaggio utilizzato è, come per iOS, *Swift*.

Tutte le applicazioni, che sono state sviluppate su una versione del sistema operativo watchOS inferiore alla 2.0, presentano la seguente struttura:

- **WatchKit App**: rappresenta l'applicazione che gira su Apple Watch e contiene la storyboard e le risorse associate all'interfaccia utente.

- **WatchKit Extension**: rappresenta una parte dell'applicazione che gira su iPhone e contiene il codice per modificare l'interfaccia utente e per rispondere alle interazioni dell'utente.

Di ogni applicazione, oltre alle interfacce grafiche di cui si compone, è possibile specificare:

- **Glance**: rappresenta una view supplementare per mostrare all'utente le informazioni più importanti dell'applicazione.
- **Notifica**: rappresenta la notifica mostrata dall'applicazione, che è possibile modificare graficamente.

Ogni interfaccia della nostra applicazione eredita dalla classe *WKInterfaceController*, nella quale possiamo aggiungere tutti gli elementi grafici che Xcode ci offre.

```
import WatchKit

class MyInterfaceController: WKInterfaceController {

  @IBOutlet weak var myLabel: WKInterfaceLabel!
  @IBOutlet weak var myButton: WKInterfaceButton!

  override func awakeWithContext(context: AnyObject?) {
    self.myLabel.setText(''Sample text'')
  }

  @IBAction func buttonPressed() {
    self.myLabel.setText(''Text from button'')
  }
}
```

Listing 3.1: Esempio di codice in Swift con WatchKit

Il ciclo di vita di un'applicazione in Apple Watch è definita da tre metodi principali:

- **awakeWitchContext**: questo metodo viene eseguito all'apertura dell'interfaccia per configurarne gli oggetti grafici e caricarne i dati.
- **willActivate**: questo metodo viene eseguito nel momento in cui un'interfaccia è in procinto di essere visibile all'utente.
- **didDeactivate**: questo metodo viene eseguito nel momento in cui l'applicazione è in procinto di porsi in uno stato di inattività.

3.4.2 Smartwatch con Android Wear

Nel 2014 Google rilascia una versione del sistema operativo Android destinato agli smartwatch, dal nome *Android Wear*.

Figura 3.7: Motorola Moto 360

Così come negli smartphone, non c'è un unico dispositivo compatibile con Android Wear ma, al lancio del sistema operativo per smartwatch, molte aziende hanno progettato il loro orologio. E' il caso di Motorola con lo smartwatch Moto 360, oppure Samsung con Samsung Gear Live.

Anche Android Wear, per aver accesso completo alle funzionalità che offre, richiede una connessione con uno smartphone Android.

Per quanto riguarda la programmazione su questi dispositivi, essa è del tutto simile a quella su smartphone Android. Il linguaggio di riferimento è Java, a cui Google aggiunge delle librerie per la realizzazione di interfacce grafiche e per la comunicazione tramite scambio di messaggi tra lo smartwatch e lo smartphone. Una caratteristica tipica nei dispositivi con sistema operativo Android Wear è l'impossibilità di scaricare contenuti dalla rete direttamente dall'orologio ma è necessario l'intervento dello smartphone, che poi provvederà ad inviare il contenuto richiesto allo smartwatch tramite messaggio.

```
public void onDataChanged(DataEventBuffer dataEvents) {
    for (DataEvent event : dataEvents) {
        if (event.getType() == DataEvent.TYPE_CHANGED
           && event.getDataItem().getUri().getPath().
            equals(PATH)) {
                DataMapItem data = DataMapItem.
                    fromDataItem(event.getDataItem());
                Asset profileAsset = data.getDataMap().
                    getAsset(PATH);
                [...]
        }
    }
}
```

Listing 3.2: Esempio di codice Java per la ricezione di messaggi con contenuto su Android Wear.

L'esempio mostra come, ad ogni ricezione di messaggio corrispondente al path di interesse (ad esempio "/image" per le immagini), il contenuto venga prelevato e salvato in una variabile di tipo *Asset*. Successivamente vengono eseguite le operazioni di interesse con il contenuto ricevuto.

Capitolo 4

Il Cloud come componente fondamentale per i sistemi connessi

Abbiamo visto l'importanza di Internet nella progettazione di sistemi embedded, in particolare la possibilità di salvare e condividere dati che difficilmente sarebbe possibile mantenere in locale.

È in quest'ottica di mantenere dati e servizi sempre disponibili che prende spazio il Cloud, permettendo ai sistemi di mantenersi aggiornati e sincronizzati, creando quelle rete di dispositivi descritta da Internet of Things.

Vediamo ora nello specifico cosa sia il Cloud e quali sono le principali piattaforme che lo utilizzano in un'ottica Internet of Things.

4.1 Cos'è il Cloud?

Per Cloud Computing si intende quell'insieme di risorse, quali l'archiviazione, elaborazione e trasmissione dei dati, che vengono rese disponibili all'utente attraverso Internet. In una definizione del National Institute of Standards and Technology (NIST), un'agenzia del governo dedicata alla definizione di standard in grado di guidare i progressi dell'industria e del com-

mercio, si evincono cinque caratteristiche essenziali che il Cloud deve avere per essere definito tale:

- **Self-Service**: l'utente può richiedere i servizi in autonomia, senza l'intervento dei gestori dell'infrastruttura o dei service provider.
- **Accessibilità globale**: i servizi devono essere disponibili in rete e accessibili attraverso meccanismi standard che promuovono l'utilizzo di piattaforme eterogenee, come smartphone, tablet e laptop.
- **Condivisione delle risorse**: le risorse di calcolo del fornitore sono messe in comune per servire molteplici consumatori utilizzando un modello condiviso, con le diverse risorse fisiche e virtuali assegnate e riassegnate dinamicamente in base alla domanda.
- **Rapida elasticità**: le risorse possono essere acquisite e rilasciate elasticamente, in alcuni casi anche automaticamente, per scalare rapidamente verso l'esterno e l'interno in relazione alla domanda.
- **Misurabilità dei servizi**: i sistemi cloud controllano automaticamente e ottimizzano l'uso delle risorse, facendo leva sulla capacità di misurazione ad un livello di astrazione appropriato per il tipo di servizio. L'utilizzo delle risorse può essere monitorato, controllato e segnalato, fornendo trasparenza sia per il fornitore che per l'utilizzatore del servizio.

Inoltre, si identificano tre tipologie di servizi che un sistema cloud è in grado di erogare:

- **Software as a Service (SaaS)**: viene fornita agli utenti la possibilità di utilizzare le applicazioni del fornitore, funzionanti su un'infrastruttura Cloud.
- **Platform as a Service (PaaS)**: viene fornita agli utenti la possibilità di distribuire sull'infrastruttura cloud applicazioni create in pro-

prio oppure acquisite da terzi, utilizzando linguaggi di programmazione, librerie, servizi e strumenti supportati dal fornitore.

- **Infrastructure as a Service (IaaS)**: viene fornita agli utenti la possibilità di acquisire elaborazione, memoria, rete e altre risorse fondamentali di calcolo, inclusi sistemi operativi e applicazioni.

4.2 Servizi cloud

Il Cloud è oggetto di interesse di numerose aziende, in quanto incrementa la quantità di servizi che possono offrire ai propri utenti. In particolare nell'ambito mobile gli usi del Cloud sono innumerevoli: dal salvataggio di foto alla memorizzazione in remoto dei contatti della rubrica fino al ritrovamento di uno smartphone smarrito.

Vediamo ora una descrizione di tre servizi cloud fra i tanti disponibili ad oggi: iCloud di Apple, Google Drive e Cloudinary.

4.2.1 iCloud

iCloud è un insieme di servizi di cloud computing sviluppato da Apple. Integrato in OS X e iOS, permette di gestire le proprie e-mail, memorizzare la propria rubrica, utilizzare 5GB di spazio gratuito per salvare i propri dati, trovare i propri dispositivi in caso di smarrimento e, infine, utilizzare applicazioni quali Pages, per creazione di documenti, Numbers, per la creazione di fogli elettronici e Keynote, per la creazione di presentazioni. Integra, inoltre, funzionalità per la sincronizzazione nel cloud delle proprie note, degli eventi del calendario e dei promemoria memorizzati nei propri dispositivi.

Tramite un unico account Apple, è possibile mantenere i propri dati memorizzati nel cloud per poi poterli gestire da un qualsiasi dispositivo associato a quell'account.

4.2.2 Google Drive

Google Drive è il servizio cloud messo a disposizione da Google che permette di usufruire dello spazio offerto di 15GB per memorizzare e condividere dati e, inoltre, permette la modifica collaborativa di documenti. Recentemente è stato aggiunto il servizio Google Foto, che permette l'hosting gratuito e illimitato di foto nei server cloud.

Una caratteristica che lo contraddistingue è la presenza di applicazioni cloud, come *Google Docs*, che permette la creazione di documenti e la modifica contemporanea di questi fra più utenti.

Tutte le funzionalità sono accessibili da qualsiasi dispositivo associato all'account di Google.

4.2.3 Cloudinary

Cloudinary è una piattaforma cloud rivolta in particolare all'hosting di immagini, che fornisce numerosi servizi per la loro manipolazione e condivisione. È caratterizzato dal disporre di diversi SDK disponibili sia in ambito mobile (Android e iOS) sia per linguaggi come Java, Ruby, PHP e i linguaggi .NET. Questi kit di sviluppo permettono di effettuare operazioni come l'upload, l'editing e la cancellazione di un'immagine direttamente nel software che si intende realizzare.

Una volta completata la registrazione al portale, Cloudinary fornisce due chiavi segrete (definite come *API Key* e *API Secret*) che permettono l'accesso alle funzionalità di amministratore. Queste sono necessarie se si intende implementare gli SDK di Cloudinary nella propria applicazione.

Per esempio, per l'upload di un'immagine sui server di Cloudinary tramite SDK, occorrerà definire:

- Un'astrazione a livello di linguaggio del nostro spazio cloud su Cloudinary
- Attributi da associare alla nostra immagine, come nome, tag, sottocartella.

```
Cloudinary cloudinary = new Cloudinary(ObjectUtils.
    asMap(''cloud_name'', ''your_cloud_name'',
 ''api_key'', ''your_api_key'',
 ''api_secret'', ''your_api_secret''));

  File fileToUpload = new File(''file_path'');

  cloudinary.uploader().upload(fileToUpload,
      ObjectUtils.asMap(''public_id'', ''file_name''));
```

Listing 4.1: Esempio di upload su Cloudinary in Java

Una volta terminato l'upload di un'immagine, Cloudinary ci restituirà il link corrispondente. Questo può essere utilizzato in tutte le applicazioni che richiedono quell'immagine.

Conoscendo il nome del file memorizzato in Cloudinary, tramite SDK è possibile ottenere il link corrispondente oppure eliminare il file.

```
class CloudinaryRequester:  CLUploaderDelegate {

  override func awakeWithContext(context: AnyObject?) {
    var cloudinary = CLCloudinary(url: ''cloudinary://
        api_key:api_secret@cloud_name'')
    var uploader = CLUploader(cloudinary, delegate:
        self)
    uploader.explicit(''public_id'', options: [''type''
        :''upload''], withCompletion:
        onCloudinaryCompletion, andProgress:
        onCloudinaryProgress)
  }
```

```
    func onCloudinaryCompletion(successResult:[NSObject :
        AnyObject]!, errorResult:String!, code:Int,
        idContext:AnyObject!) {
      println(successResult[''secure_url''])
    }
}
```

Listing 4.2: Esempio di richiesta di un link a Cloudinary in Swift

4.3 Piattaforme cloud orientate a Internet of Things

Con l'avvento di Internet of Things, si sono sviluppate numerose piattaforme cloud specializzate nel raccogliere dati da sistemi embedded e renderli disponibili all'utente su richiesta. Questo ha semplificato la creazione di software progettati in un'ottica IoT.

Segue una descrizione delle principali piattaforme.

4.3.1 Carriots

Carriots è una *Platform as a Service* sviluppata per progetti di Internet of Things e Machine to Machine. Permette di salvare i dati generati dai dispositivi nei propri server, offrendo inoltre delle API per comunicare con la piattaforma e creare applicazioni proprie.

Dopo aver effettuato la registrazione nel sito di riferimento, Carriots offre due API key (una di sola lettura, l'altra con permessi di scrittura e lettura), necessarie per poter accedere ai dati nella piattaforma o per pubblicarli.

L'accesso o il caricamento dei dati avviene tramite richieste HTTP ai server di Carriots tramite una specifica sintassi. Carriots supporta richieste HTTP di tipo:

- **GET**: per ottenere le informazioni di un elemento

- **POST**: per creare un nuovo elemento
- **PUT**: per aggiornare un elemento esistente
- **DELETE**: per eliminare un elemento

Per esempio, per l'upload di un certo dato da parte di un device, la richiesta sarà la seguente:

```
POST /streams HTTP/1.1
Host: api.carriots.com
Accept: application/json
User-Agent: User-Agent
Content-Type: application/json
carriots.apikey: Your-Apikey
Content-Length: Content-length
Connection: close

Data
{
''protocol'' : ''v2'',
''device'' : ''Device-ID'',
''at'' : ''now'',
''data'' : ''data in JSON format''
}
```

Listing 4.3: Esempio di richiesta HTTP POST a Carriots

Poniamo la nostra attenzione su *Data*. Esso è formato da quattro campi obbligatori:

- **Protocol**: la versione di protocollo (di default la v2)
- **Device**: l'ID del device registrato su Carriots

- **At**: indica il momento in cui è stata fatta la richiesta HTTP
- **Data**: il contenuto dei dati che vogliamo inviare, in formato JSON

Ogni dato che il device vorrà inviare, quindi, dovrà essere in formato JSON.

Carriots offre, inoltre, altri due servizi nella sua piattaforma: i *listener* e i *trigger*.

Un listener, come dice il nome stesso, si mette in ascolto di un evento e, in base ad esso, esegue un'azione. Ogni azione è sviluppata in linguaggio Groovy. Per esempio, supponiamo di voler inviare un'email se la temperatura che riceviamo dal nostro dispositivo supera i 30 gradi.

```
{
  ''temperature'' : 31
}
```

Listing 4.4: Esempio di dato ricevuto da Carriots

La prima cosa che la piattaforma esegue è controllare la *If expression* specificata.

```
context.data.temperature > 30
```

Listing 4.5: Esempio di If expression in Carriots

Se la if expression è verificata, viene eseguita la *Then expression*.

```
import com.carriots.sdk.utils.Email;

def email = new Email ();
email.to = ''<your email>'';
email.subject =''Temperatura maggiore di 30 gradi'';
```

```
email.message = ``La temperatura rilevata ha superato
   la soglia limite di 30 gradi'';
email.send();
```

Listing 4.6: Esempio di Then expression in Carriots

Un trigger permette di inviare ad un servizio esterno un dato ricevuto. Per esempio, supponiamo di aver ricevuto un JSON contenente la temperatura e l'umidità.

```
{
  ``temperature'' : 31
  ``humidity'' : 65
}
```

Listing 4.7: Secondo esempio di dato ricevuto da Carriots

Volendo inviare solamente la temperatura ad un servizio esterno, nel pannello della piattaforma di Carriots occore specificare il trigger con i seguenti dati:

- **Name**: il nome che si vuole assegnare al trigger
- **Description**: una descrizione per il trigger
- **Max Retries**: numero massimo di tentativi (di default 2)
- **Push frequency**: la frequenza di invio (di default 5)
- **Enabled**: per attivare il trigger (di default: checked)
- **Id Service**: l'id del servizio
- **URL**: l'URL del servizio
- **Verb**: la tipologia di invio del dato al servizio esterno (di default POST)

- **User**: il nome utente (facoltativo)
- **Password**: la password (facoltativa)
- **Payload**: il dato da voler inviare al servizio esterno. Nell'esempio della temperatura, la sintassi sarà la seguente: Temp=%%%temperature%%%

4.3.2 TempoIQ

TempoIQ è un servizio cloud per il monitoraggio, il salvataggio e l'analisi di dati ricevuti da sensori e dispositivi. Fornisce numerose API e offre kit di sviluppo in diversi linguaggi come Python, Java, C# e Ruby. Permette di organizzare i propri sensori e i propri dispositivi, associando ad ognuno eventuali attributi. Per esempio, volendo creare un dispositivo che permetta di misurare la temperatura e l'umidità in un determinato luogo, si può procedere nel seguente modo:

- Si effettua una connessione tra il client e i server di TempoIQ
- Si utilizzano le API fornite per la creazione e il salvataggio di nuovi device nella piattaforma

```
import com.tempoiq.*;

//Connessione con il server
InetSocketAddress host = new InetSocketAddress(``my-
    company.backend.tempoiq.com'', 443);
Credentials credentials = new Credentials(``my-key'',
    ``my-secret'');
Client client = new Client(credentials, host, ``https''
    );

//Creazione degli attributi del device
```

```
Map<String, String> attributes = new HashMap<String,
    String>();
attributes.put(``model'', ``v1'');

//Creazione dei sensori
Sensor sensor1 = new Sensor(``temperature'');
Sensor sensor2 = new Sensor(``humidity'');
List<Sensor> sensors = new ArrayList<Sensor>();
sensors.addAll(Arrays.asList(sensor1, sensor2));

//Creazione del device
Device device = new Device(``thermostat.0'', ````,
    attributes, sensors);

//Salvataggio del device in TempoIQ
Result<Device> result = client.createDevice(device);
```

Listing 4.8: Esempio di creazione di un device nei server di TempoIQ in Java

Ogni device, una volta registrato, può memorizzare i valori che riceve dai sensori e renderli disponibili ai device che ne fanno richiesta. Per ogni dato salvato, è richiesta, oltre al valore del sensore, la data di riferimento.

```
import java.util.*;
import com.tempoiq.*;
import org.joda.time.*;

//Creazione dei dati da salvare
DateTime dt1 = new DateTime(2015, 1, 1, 0, 0, 0, 0,
    DateTimeZone.UTC);
Map<String, Number> points1 = new HashMap<String,
    Number>();
points1.put(``temperature'', 68);
```

```
points1.put(``humidity'', 71.5);
MultiDataPoint mp1 = new MultiDataPoint(dt1, points1);

DateTime dt2 = dt1.plus(Period.minutes(5));
Map<String, Number> points2 = new HashMap<String,
    Number>();
points2.put(``temperature'', 67.5);
points2.put(``humidity'', 70.0);
MultiDataPoint mp2 = new MultiDataPoint(dt2, points2);

//Salvataggio dei dati su TempoIQ
Device device = new Device(``thermostat.0'');
Result<WriteResponse> result = client.writeDataPoints(
    device, Arrays.asList(mp1, mp2));
```

Listing 4.9: Esempio di salvataggio dei valori ricevuti dai sensori in TempoIQ

```
import java.util.*;
import com.tempoiq.*;
import org.joda.time.*;

//Seleziono il range di tempo
DateTime start = new DateTime(2015, 1, 1, 0, 0, 0, 0,
    DateTimeZone.UTC);
DateTime end = new DateTime(2015, 1, 2, 0, 0, 0, 0,
    DateTimeZone.UTC);

//Seleziono il device
Device device = new Device(``thermostat.0'');
Selection selection = new Selection()
  .addSelector(Selector.Type.DEVICES, Selector.key(
      device.getKey()));
```

```
//Recupero le informazioni
Cursor<Row> cursor = client.read(selection, start, end)
    ;
for(Row row : cursor) {
  System.out.format(``timestamp \%s, temperature: \%f,
      humidity: \%f'',
          row.getTimestamp(),
          row.getValue(``thermostat.0'', ``temperature'')
              ,
          row.getValue(``thermostat.0'', ``humidity'')).
              println();
}
```

Listing 4.10: Esempio di lettura di dati di un determinato periodo da TempoIQ

4.3.3 Kaa

Kaa è una piattaforma middleware open-source pensata per Internet of Things. Permette la realizzazione di soluzioni IoT end-to-end, raccogliendo e analizzando i dati ricevuti in real-time. Kaa offre kit di sviluppo in linguaggio Java, C e C++ che includono la comunicazione client-server, l'autenticazione, l'ordinamento e il criptaggio dei dati. Inoltre permette la creazione di applicazioni che funzionano su ogni tipo di connessione alla rete, consentendo di scegliere un protocollo di trasporto esistente o crearne uno personalizzato.

Basato su uno schema a log, i server di Kaa sviluppano il binding degli oggetti e API, fornendo un SDK personalizzato. L'utente utilizza queste API per istruire Kaa ad inviare record di log al sistema back-end o sistemi warehouse per la manipolazione dei dati.

```
public class SmartCar {
  public static void Main(String[] args) {
    DesktopKaaPlatformContext kaaContext = new
       DesktopKaaPlatformContext();
    KaaClient kaaClient = Kaa.newClient(kaaContext,
       null);

    kaaClient.setProfileContainer(new ProfileContainer
       () {
        @Override
        public Car getProfile() {
          return new Car(``Marca'', ``Modello'',
                      ``Colore'', ``Targa'');
        }
    });

    kaaClient.start();
    kaaClient.stop();
  }
}
```

Listing 4.11: Esempio di codice Java con Kaa SDK

4.3.4 Canopy

Canopy è una piattaforma cloud open-source specializzata in Internet of Things. Fornisce un servizio per il salvataggio dei dati, fornendo all'utente API per l'accesso a tutte le funzionalità. Offre, inoltre, una libreria che porta le funzioni del cloud nei firmware embedded e alcune librerie per lo sviluppo di app IoT.

La piattaforma permette la creazione di device e di proprietà ad essi definite tramite l'utilizzo delle *Rest-API* fornite.

```
POST /api/create_devices
{
    ''quantity'' : <QUANTITY>
    ''friendly_names'' : [<FRIENDLY_NAME>, ...]
}
```

Listing 4.12: Esempio di creazione device tramite Rest-API in Canopy

Permette, inoltre, la richiesta di dati, specificato un determinato periodo di tempo. Ad ogni richiesta effettuata, il server invierà una risposta contenente i dati, se richiesti.

```
GET /api/device/d5c827d2-1ba7-4da6-b87b-0864f0969fa8/
    temperature?start_time='2015-03-19 12:00:00'
```

Listing 4.13: Esempio di richiesta di un dato a Canopy tramite Rest-API

```
{
    ''result'' : ''ok'',
    ''current_clock_us'' : 1426772581829000,
    ''current_clock_utc'' : ''2015-03-19T22:47:31Z'',
    ''count'' : 3,
    ''samples'' : [
        {
            ''t'': ''2015-03-18T16:57:36Z'',
            ''v'': 24.821886
        }
    ]
}
```

Listing 4.14: Esempio di risposta di Canopy

Capitolo 5

Quando la casa incontra IoT: la domotica

Nel primo capitolo abbiamo introdotto Internet Of Things, evidenziando i principali campi a cui si applica una tale innovazione.

In questo capitolo ci focalizzeremo su un campo in particolare: la domotica. Essa, infatti, risulta un caso di studio molto interessante perché mostra come possono interagire e come possono essere controllati oggetti domestici tramite Internet e come la casa risulti più sicura grazie ai diversi sensori e dispositivi in comunicazione fra loro.

Procediamo, quindi, ad introdurre il concetto di domotica e i principali sistemi e piattaforme che, ad oggi, sono disponibili.

5.1 Introduzione alla domotica

La domotica è la tecnologia che studia l'automazione della casa. Il termine deriva dal neologismo francese *domotique*, a sua volta contrazione della parola greca *domos* (casa, costruzione) e di *automatique* (automatica; secondo altri *informatique*, informatica): quindi, letteralmente, "casa automatica". Con tale termine si identificano tutte quelle tecnologie integrate tra loro che

consentono di rendere automatiche una serie di operazioni all'interno delle mura domestiche.

Tra le numerose funzioni caratteristiche di un ambiente domotico, abbiamo:

- Accensione e spegnimento di luci
- Accensione e spegnimento di impianti di riscaldamento o condizionamento
- Controllo di porte e cancelli
- Controllo di elettrodomestici
- Controllo di parametri ambientali e atmosferici
- Gestione di un sistema di allarme

I vantaggi di automatizzare un ambiente domestico sono molteplici:

- **Maggiore sicurezza**: grazie all'automazione è possibile aumentare i livelli di sicurezza dell'abitazione, sia nel caso di ingresso di intrusi, grazie ad un sistema di monitoraggio della casa, sia nel caso di incendi o fughe di gas, grazie a degli opportuni sensori adibiti a questo compito.
- **Maggiore comfort all'interno dell'abitazione**: è possibile rendere l'habitat quotidiano più accogliente e gradevole per chi lo vive.
- **Maggiore versalità**: un impianto domotico garantisce una maggiore versalità rispetto ad un impianto tradizionale.
- **Maggiore risparmio nella gestione dell'impianto**: il controllo sull'energia presente nell'abitazione consente di risparmiare sui costi di gestione dell'impianto.

Il settore della domotica ha destato un tale interesse da parte di numerose compagnie, ognuna delle quali ha proposto un proprio sistema per la gestione automatizzata della casa.

5.2 Sistemi domotici

I principali sistemi domotici, in continuo sviluppo da parte delle aziende, sono:

- **HomeKit** di Apple
- **SmartThings** di Samsung
- **Nest** di Google

5.2.1 HomeKit

HomeKit è un framework prodotto da Apple per controllare e comunicare con accessori connessi nella casa dell'utente. Ogni accessorio, abilitato per HomeKit, è controllabile dal proprio dispositivo iOS ed è possibile gestirlo tramite l'assistente vocale Siri.

HomeKit permette di creare un layout per una o più case. Ogni abitazione è definita dal tipo *HMHome*, che può contenere un numero variabile di stanze, definite dal tipo *HMRoom*. Analogamente ogni stanza può contenere diversi accessori, definiti da *HMAccessory* e una serie di servizi, definiti da *HMService*, che corrispondono alle azioni che gli accesori sono in grado di svolgere. Per aver accesso a tutti gli oggetti HomeKit, si utilizza la classe di riferimento *HMHomeManager*.

```
self.homeManager = [[HMHomeManager alloc] init];
self.homeManager.delegate = self;
HMHome *home = self.homeManager.primaryHome;
HMHome *room;

for (room in home.rooms) {
    //do something
}
```

Listing 5.1: Esempio di codice in Objective-C con framework HomeKit

È possibile suddividere le stanze in gruppi, chiamate zone e definite dal tipo *HMZone*. Allo stesso modo è possibile specificare gruppi di servizi, tramite il tipo *HMServiceGroup*.

Per ogni casa è memorizzato un database, sincronizzato con i dispositivi iOS admin e ospiti che hanno ricevuto l'autorizzazione per quella specifica casa. Ogni app, per mantenere il database aggiornato, deve essere osservatore dei cambiamenti alla base di dati.

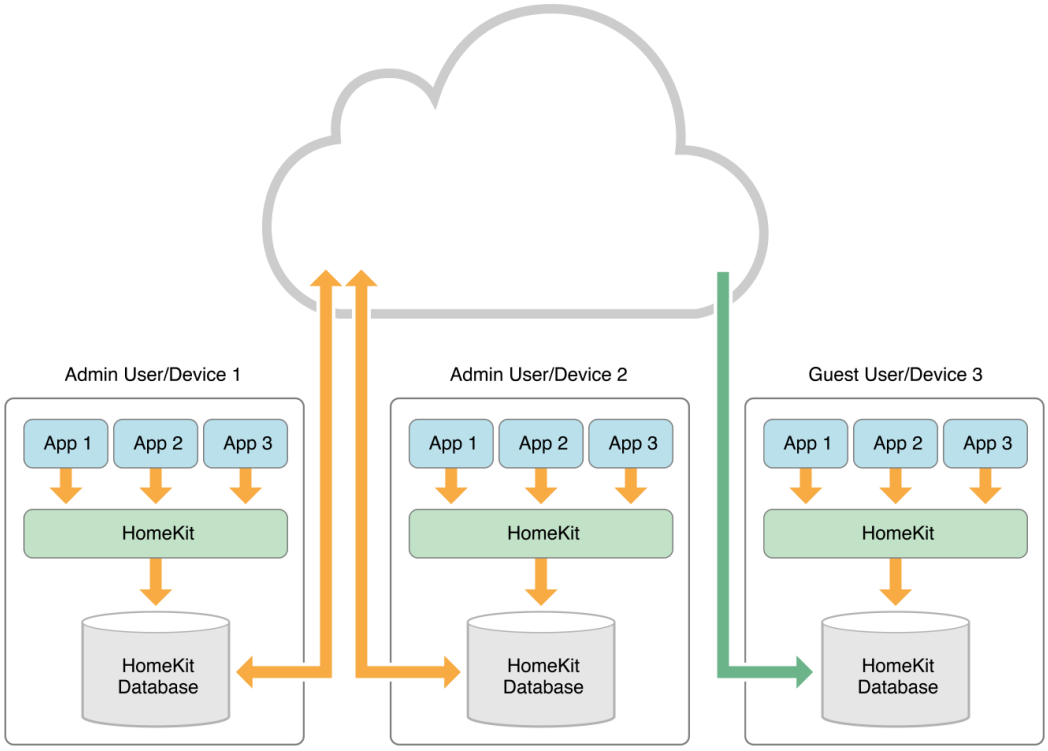

Figura 5.1: Gestione del database in HomeKit

5.2.2 SmartThings

SmartThings è il sistema domotico realizzato da Samsung per il controllo e la sicurezza della propria casa.

È strutturato da tre componenti principali:

- L'applicazione mobile disponibile per Android, iOS e Windows Phone.

- Lo **SmartThings Hub**, il cuore del sistema, che controlla tutti i prodotti connessi ad esso.
- Gli accessori compatibili.

Figura 5.2: Samsung SmartThings Hub

Lo SmartThings Hub è collegato a Internet tramite Ethernet e contiene unità Z-Wave, progettate appositamente per la domotica e per interagire con i dispositivi connessi.

Tramite l'applicazione mobile è possibile ricevere notifiche dalla casa domotica, controllare i device e impostare le proprie preferenze.

L'architettura del sistema realizzato da Samsung astrae i dettagli di uno specifico device, permettendo allo sviluppatore di focalizzarsi solamente nelle funzionalità e azioni supportate da quel device.

A livello concettuale, l'architettura si presenta così:

- Applicazioni web e mobile
- SmartApp e SmartService
- Astrazione dei device e delle funzionalità
- Connettività al Cloud
- Device

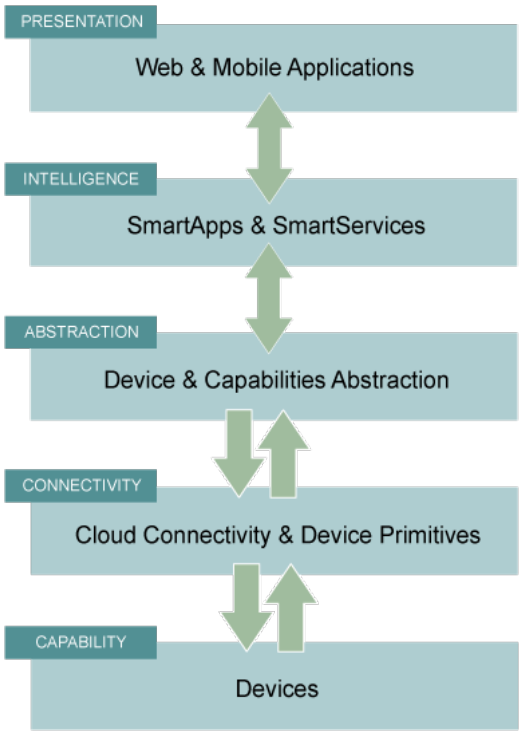

Figura 5.3: Architettura concettuale di SmartThings

Da questo schema possiamo notare un componente fondamentale nel sistema di Samsung: il Cloud. In esso, infatti, vengono salvati i diversi eventi verificatosi in modo tale che le applicazioni mobile riflettano lo stato corrente della casa. Inoltre, alcune smart app richiedono di essere eseguite nel Cloud per funzionare anche su quei dispositivi non collegati all'hub di Samsung, così come le stesse smart app necessitano di alcuni servizi e, per richiamarli, il Cloud è un'ottima alternativa in quanto permette un monitoraggio semplificato degli errori e assicura la privacy degli utenti.

Un'ulteriore suddivisione che possiamo fare all'interno della piattaforma SmartThings è la seguente:

- **Gli account**: in una scala gerarchica occupano il posto più elevato e rappresentano gli utenti che utilizzano la piattaforma.

- **I luoghi**: rappresentano locazioni di interesse come, ad esempio, la casa oppure l'ufficio.
- **I gruppi**: rappresentano le stanze o spazi fisici di uno specifico luogo.
- **I dispositivi**: tutti i device organizzati all'interno di un gruppo.

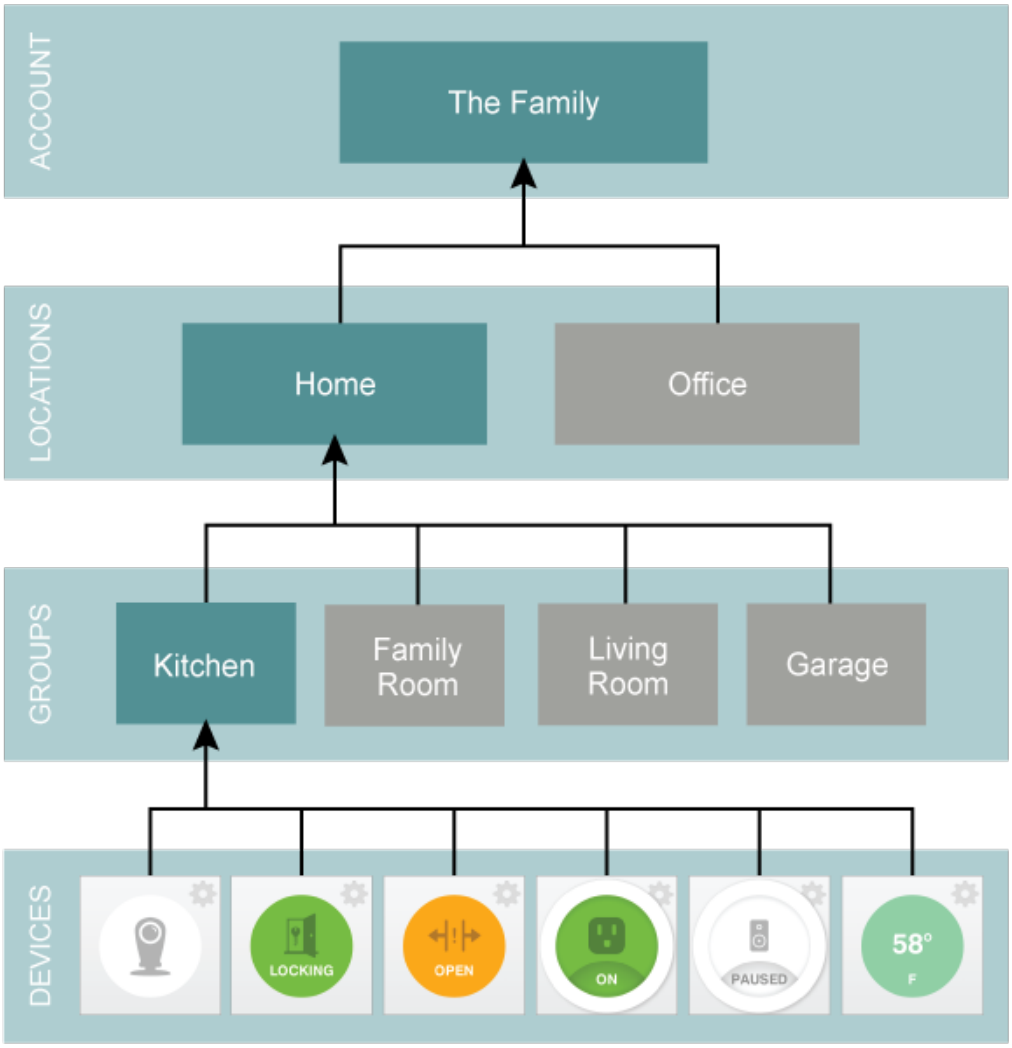

Figura 5.4: Suddivisione gerarchica dei contenitori di elementi in SmartThings

5.2.3 Nest

Google, nel gennaio del 2014, acquista l'azienda Nest Labs, che si occupa di produrre dispositivi per l'automazione della casa. Ne sono esempi il termostato Nest, il rilevatore di fumo Nest Protect e la Nest Cam.

Il termostato Nest è in grado di studiare la nostra casa e le nostre preferenze, programmandosi automaticamente con la giusta temperatura. È in grado di rilevare se l'utente è fuori casa, entrando in modalità di risparmio energetico.

Il rilevatore di fumo Nest Protect, invece, utilizza un sensore fotoelettrico che, tramite due diverse lunghezze d'onda della luce, riesce ad identificare diverse tipologie di fumo e avvisare l'utente in caso di pericolo.

Infine, la Nest Cam permette di monitorare la nostra casa da remoto.

Figura 5.5: I prodotti Nest: termostato Nest, Nest Protect e Nest Cam

Ogni dispositivo sopra illustrato è in grado di interagire con gli altri ed è controllabile a distanza dal proprio smartphone.

Google ha avviato, inoltre, un programma dal nome *Works with Nest* per permettere a dispositivi di terze parti di comunicare con i prodotti Nest.

Ne è un esempio *Kevo Smart Lock*, un dispositivo in cui inserire le chiavi di casa che comunica direttamente con il termostato Nest. Questo disposi-

tivo riconosce chi si trova a casa e istruisce il termostato affinché regoli la temperatura secondo le abitudini dell'utente.

Un altro esempio è dato da *Rachio*, un dispositivo che comunica direttamente con Nest Protect e, in caso di rilevamento di un incendio, accende gli irrigatori attorno casa.

Architettura di Nest

Il servizio Nest fornisce i dati che rappresentano il modello della casa. I dispositivi e le applicazioni leggono questi dati per eseguire delle azioni opportune, come per esempio la modifica e l'aggiornamento delle informazioni analizzate. Il modello è costituito da un documento memorizzato nel servizio, condiviso tra i vari dispositivi e strutturato gerarchicamente con un JSON, dove in cima abbiamo i device e le strutture mentre subito sotto abbiamo specifiche tipologie di dispositivi.

Anche lo stato del sistema è memorizzato in un documento JSON, al quale si sottoscrivono i vari dispositivi in modo tale da essere notificati in caso di cambiamenti dello stesso.

Infine Nest implementa i protocolli di Firebase, provider di servizi cloud, che fornisce delle API di alto livello per la sincronizzazione in real time dei dati, semplificando il carico di lavoro dello sviluppatore.

Capitolo 6

Smart Home: un caso di studio

Smart Home è il titolo assegnato al progetto sviluppato come caso di studio, in cui le tecnologie discusse nei capitoli precedenti sono utilizzate in modo integrato nella realizzazione di un sistema domotico.

Il progetto concilia, infatti, la domotica in un'ottica IoT e la estende sia ai wearable sia al Cloud.

Il capitolo procede, inizialmente, con una descrizione e analisi del sistema complessivo per focalizzarsi, in seguito, nel dettaglio di ciascun sotto-sistema che lo compone.

6.1 Descrizione e analisi dei requisiti del progetto

6.1.1 Funzionalità del sistema

Il progetto Smart Home crea un sistema di automazione della casa, con il quale l'utente è in grado di interagire con uno smartwatch, nello specifico l'Apple Watch.

Il sistema permette di:

- Rilevare la presenza di intrusi all'interno della casa

- Fotografare la stanza in cui è stato rilevato un movimento non autorizzato
- Visualizzare la foto scattata più recente per ogni stanza dall'orologio
- Accendere e spegnere le luci di casa tramite orologio
- Visualizzare la temperatura di casa tramite orologio
- Attivare e disattivare l'allarme di casa con la password desiderata (sia dallo smartwatch sia dal keypad disponibile nella casa)
- Inviare una email con la foto scattata in caso di rilevamento di intrusi
- Inviare un sms all'utente, avvisandolo della presenza di un intruso nell'abitazione

Si possono distinguere tre macro casi d'uso all'interno del nostro sistema:

- Gestione allarme di casa
- Gestione impianto di illuminazione
- Visualizza informazioni sul sistema

Gestione allarme di casa

Questo caso d'uso specifica tutte quelle operazioni svolte dall'utente che permettono di interagire con il sistema di allarme della casa.

Esso può essere suddiviso nei seguenti casi d'uso:

- **Attiva allarme da casa**: terminato l'inserimento della password tramite il keypad installato nella casa domotica, il sistema di allarme è attivato.
- **Disattiva allarme da casa**: se la password inserita tramite il keypad installato nella casa domotica è identica alla password memorizzata in precedenza, il sistema di allarme viene disattivato.

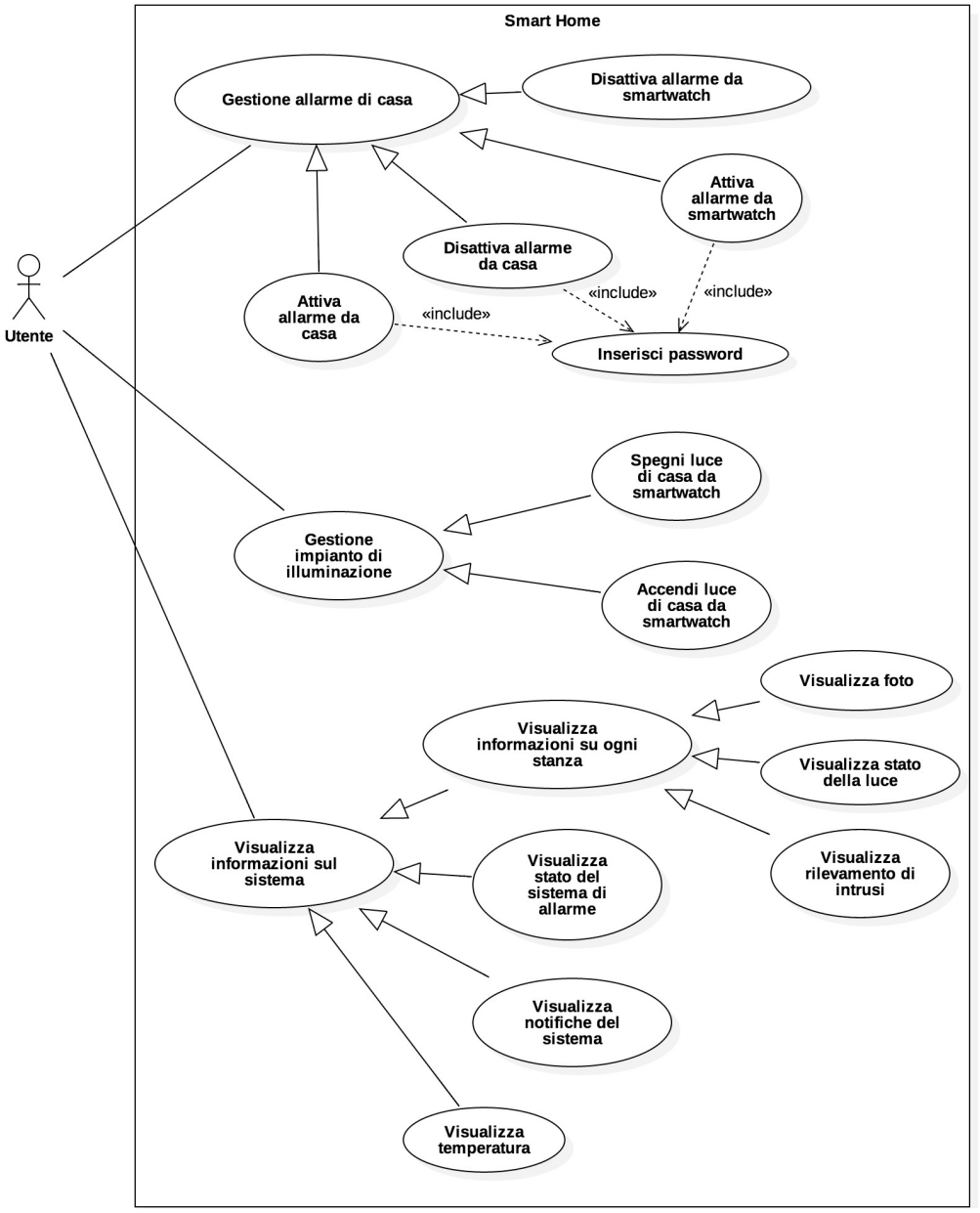

Figura 6.1: Diagramma dei casi d'uso del sistema

- **Attiva allarme da smartwatch**: tramite il proprio smartwatch, l'utente è in grado di attivare l'allarme di casa, entrando nel menù dedicato e inserendo la password desiderata.
- **Disattiva allarme da smartwatch**: a differenza dell'attivazione dell'allarme, la disattivazione tramite smartwatch è *one-click*: basterà premere l'apposito pulsante per disabilitare il sistema di allarme.

I primi tre casi dipendono dal caso d'uso *Inserisci password*, che permette all'utente di digitare una password tramite keypad o tramite smartwatch.

Gestione impianto di illuminazione

Questo caso d'uso specifica tutte quelle operazioni svolte dall'utente che permettono di interagire con l'impianto di illuminazione della casa domotica.

Esso può essere suddiviso nei seguenti casi d'uso:

- **Accendi luce di casa da smartwatch**: l'utente è in grado di accendere le luci di ogni stanza tramite il proprio smartwatch, semplicemente selezionando la stanza di interesse e procedendo all'accensione.
- **Spegni luce di casa da smartwatch**: l'utente è in grado di spegnere le luci di ogni stanza tramite il proprio smartwatch, semplicemente selezionando la stanza di interesse e procedendo allo spegnimento.

Visualizza informazioni sul sistema

Questo caso d'uso comprende tutte quelle operazioni che permettono all'utente di visualizzare informazioni riguardo al sistema.

Esso può essere suddiviso nei seguenti casi d'uso:

- **Visualizza informazioni su ogni stanza**: l'utente è in grado di avere una panoramica per ogni stanza della casa, in particolare se le luci sono accese o spente e se è stata rilevata la presenza di un intruso all'interno della stessa. Inoltre, l'utente può visualizzare, se presente, l'ultima foto scattata all'intruso dalla fotocamera del sistema.

- **Visualizza stato del sistema di allarme**: l'utente è in grado di visionare (sia da smartwatch sia dal display fornito nella casa domotica) lo stato del sistema di allarme, in particolare se il sistema è attivo o disattivato.

- **Visualizza notifiche del sistema**: l'utente è in grado di visualizzare le notifiche inviate dal sistema, quali sms ed email. Nel primo caso si avvisa l'utente che è stata rilevata una presenza non autorizzata all'interno dell'abitazione mentre, nel secondo caso, viene inviata la foto scattata all'intruso.

- **Visualizza temperatura**: L'utente è in grado, tramite il proprio smartwatch, di visualizzare la temperatura di casa.

6.1.2 Dispositivi e materiali utilizzati

Per la realizzazione del progetto sono stati utilizzati i seguenti dispositivi:

- Arduino Uno

- Raspberry Pi B+

- Apple Watch

Inoltre, per la realizzazione della struttura rappresentante un modello di una casa domotica sono stati utilizzati i seguenti materiali:

- **PIR HC-SR501**: sensore ad infrarosso per il rilevamento di intrusi nella casa.

- **LCD HD44780**: schermo per mostrare all'utente informazioni riguardo lo stato di allarme del sistema.

- **Potenziometro B10K**: utilizzato per regolare la luminosità dello schermo LCD.

- **Keypad**: per l'inserimento della password per attivare e disattivare l'allarme.
- **Raspberry Pi Camera**: fotocamera utilizzata per scattare le foto agli intrusi.
- **Servo Motore SG90**: utilizzato per il posizionamento della fotocamera.
- **Buzzer**: utilizzato per emettere un suono di allarme.
- **Sensore di temperatura TMP36**: per la misurazione della temperatura di casa.
- **Led**: per l'illuminazione delle stanze della casa.
- **Shift Register 74HC595**: registri di scorrimento che convertono input sequenziale in output parallelo. Utilizzato per utilizzare un minor numero di pin in Arduino.

6.1.3 Piattaforme, librerie e IDE di sviluppo

Essendo il progetto realizzato con dispositivi diversi, sono stati utilizzati diversi ambienti di sviluppo:

- **Arduino IDE** per la programmazione su microcontrollori Arduino, tramite il framework Wiring, basato su linguaggio C/C++
- **Eclipse** per la programmazione in Java del codice da eseguire su Raspberry Pi
- **Xcode** per la programmazione su Apple Watch, con linguaggio Swift

Sono state utilizzate, inoltre, due piattaforme cloud per il salvataggio dei dati:

- **Carriots**: per salvare lo stream dei dati generati dai sensori

- **Cloudinary**: per salvare le foto scattate dalla fotocamera

Oltre alle librerie fornite da queste due piattaforme, sono state utilizzati:

- **PigPiod**: utilizzata per il controllo di un servo motore in Raspberry Pi
- **JavaMail API**: utilizzata per l'invio di email in linguaggio Java
- **RXTXcomm**: utilizzata per la comunicazione seriale tra Raspberry Pi e Arduino
- **Keypad.h**: per la gestione di un keypad in Arduino
- **LiquidCrystal.h**: per utilizzare e controllare un LCD in Arduino
- **QueueList.h**: per gestire una coda FIFO in Arduino
- **Timer.h**: per la gestione del timer in Arduino

6.2 Progettazione architetturale del sistema (visione d'insieme)

Effettuiamo ora la progettazione architetturale del sistema. Si procederà con un'analisi globale del sistema, per poi analizzare ogni singola parte separatamente nelle sezioni successive.

6.2.1 Diagramma delle classi

Il diagramma delle classi raffigurato rappresenta un possibile modello generico per il sistema da sviluppare.

La casa domotica è costituita da un determinato numero di stanze, ognuna delle quali contiene un sistema per il rilevamento di movimento (per esempio un sensore ad infrarosso) e uno o più dispositivi di illuminazione (un led per esempio). La casa, inoltre, utilizza:

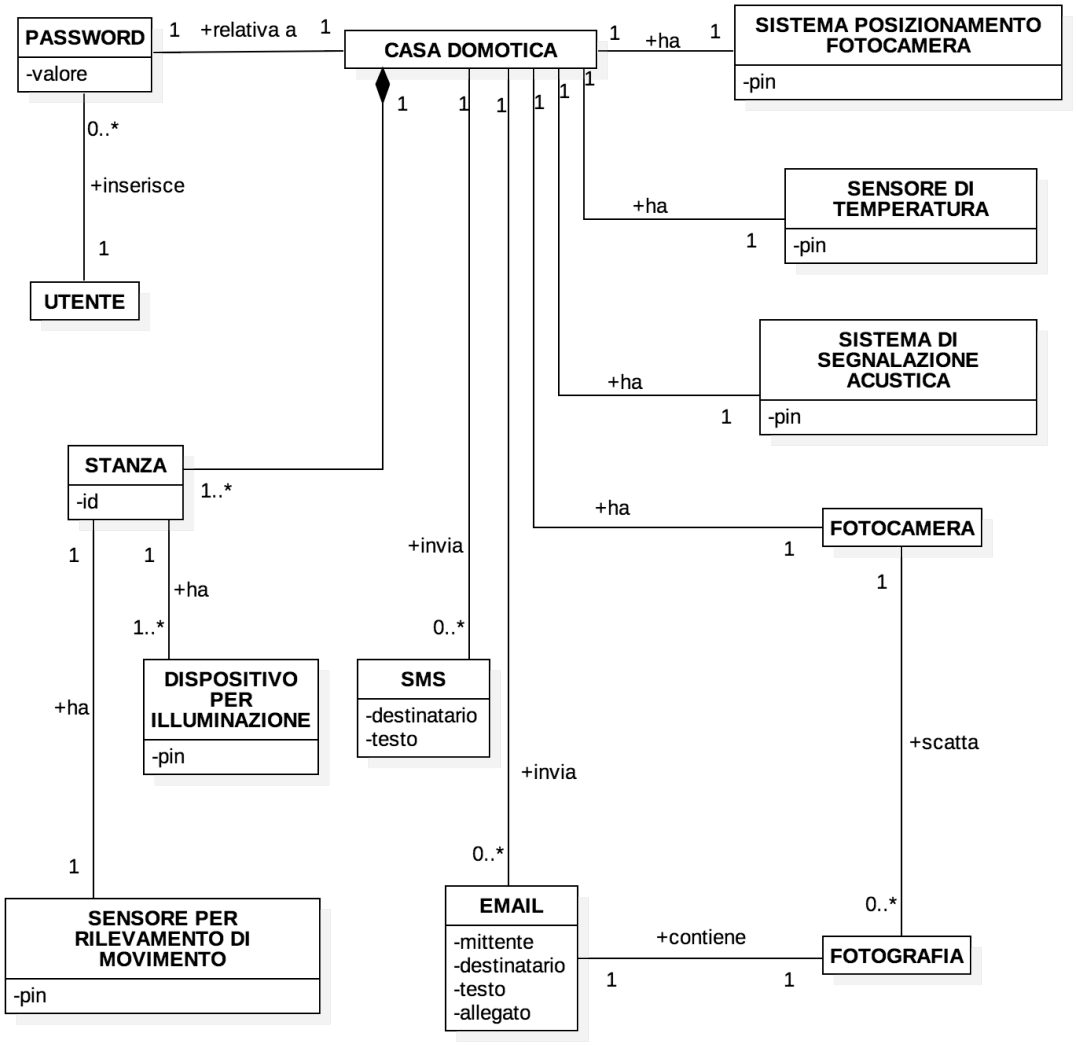

Figura 6.2: Diagramma delle classi rappresentante il modello del sistema

- Un sistema per posizionare la fotocamera nella giusta posizione (per esempio: un servo motore)
- Un sensore per la misurazione della temperatura
- Un sistema per segnalare acusticamente il rilevamento di un intruso all'interno dell'abitazione (per esempio: un buzzer)
- Una fotocamera per scattare la fotografia nella stanza oggetto di rilevamento

La casa domotica deve essere, inoltre, in grado di:

- Inviare un'email contenente la fotografia scattata
- Inviare un sms di allarme all'utente

6.2.2 Diagrammi degli stati

Possiamo rappresentare il nostro sistema attraverso tre diagrammi degli stati, che modellano:

- Lo stato del sistema di allarme
- Lo stato dell'impianto di illuminazione
- Il calcolo della temperatura

Diagramma degli stati del sistema di allarme

Il diagramma focalizza l'attenzione su tre stati principali:

- **Allarme spento**: in questo stato il sistema di allarme è disattivato e l'utente è in grado di attivare il sistema, inserendo la password tramite il keypad o tramite smartwatch. Se vi erano delle luci accese, queste vengono spente.

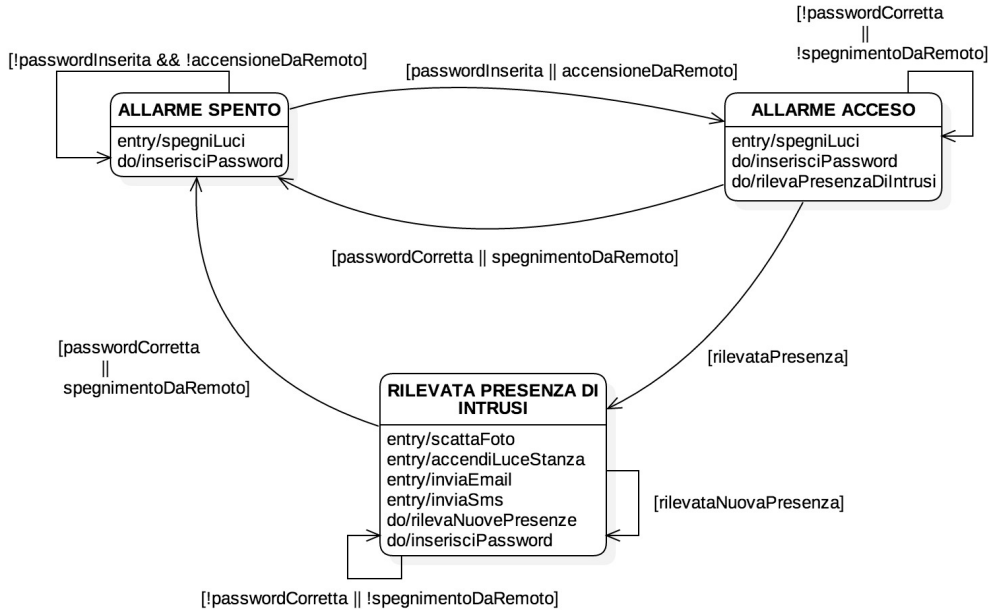

Figura 6.3: Diagramma degli stati del sistema di allarme

- **Allarme acceso**: in questo stato il sistema di allarme è attivato e si pone in attesa di eventuali rilevamenti di intrusi. L'utente è in grado di disattivare l'allarme, inserendo la password nel keypad o utilizzando il proprio smartwatch.

- **Rilevata presenza di intrusi**: in questo stato il sistema ha rilevato una presenza non autorizzata all'interno dell'abitazione. Dopo aver scattato una fotografia nella stanza oggetto di movimento, invia un sms e un'email all'utente. L'utente è in grado di disattivare l'allarme, inserendo la password nel keypad oppure utilizzando il proprio smartwatch.

Diagramma degli stati dell'impianto di illuminazione

Il diagramma mostra gli stati del sistema durante l'accensione o lo spegnimento di una luce da parte dell'utente.

In particolare si distinguono tre stati:

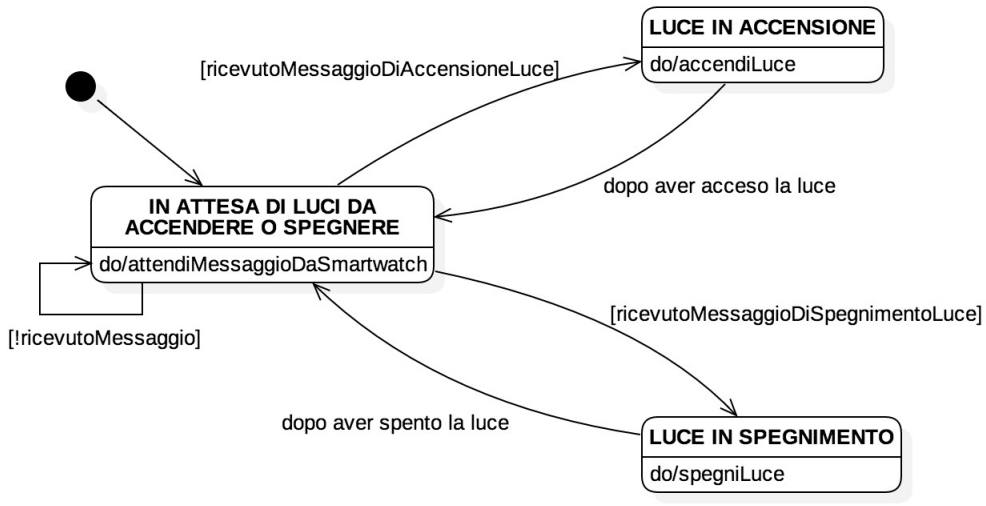

Figura 6.4: Diagramma degli stati dell'impianto di illuminazione

- **In attesa di luci da accendere o spegnere**: in questo stato il sistema attende che l'utente invii un comando dallo smartwatch per accendere o spegnere una luce.
- **Luce in accensione**: questo stato indica che è arrivato un messaggio di accensione di una luce da parte dello smartwatch e che la stessa sta per essere accesa. Effettuata questa operazione, il sistema ritornerà nello stato di attesa.
- **Luce in spegnimento**: questo stato indica che è arrivato un messaggio di spegnimento di una luce da parte dello smartwatch e che la stessa sta per essere spenta. Effettuata questa operazione, il sistema ritornerà nello stato di attesa.

Le operazioni di spegnimento e accensione delle luci sono permesse solo quando l'allarme è disattivato.

Diagramma degli stati del calcolo della temperatura

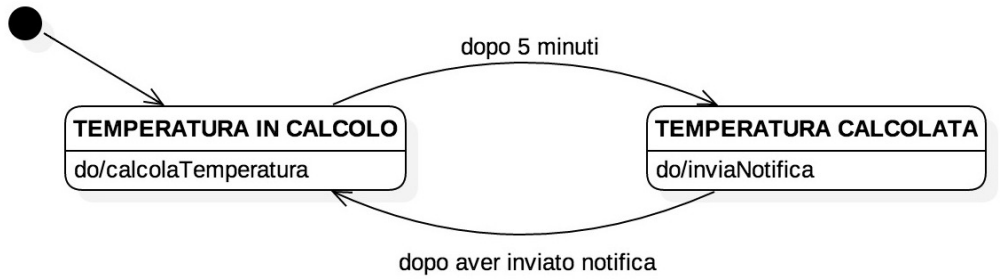

Figura 6.5: Diagramma degli stati del calcolo della temperatura

Il diagramma mostra gli stati che si alternano durante il calcolo della temperatura nel sistema.

Nello specifico, si distinguono due stati:

- **Temperatura in calcolo**: in questo stato il sistema calcola per cinque minuti la temperatura. Dopo questo periodo, raccoglie tutti i dati raccolti e ne effettua una media matematica, transitando nello stato successivo.
- **Temperatura calcolata**: in questo stato viene ricevuto il nuovo valore di temperatura e si procede alla notifica di Raspberry Pi. Dopo aver inviato la notifica contenente il valore aggiornato di temperatura, il sistema transita nello stato iniziale.

6.2.3 Struttura del sistema

Il sistema si compone di quattro componenti principali:

- Raspberry Pi
- Arduino Uno
- Apple Watch
- Cloud

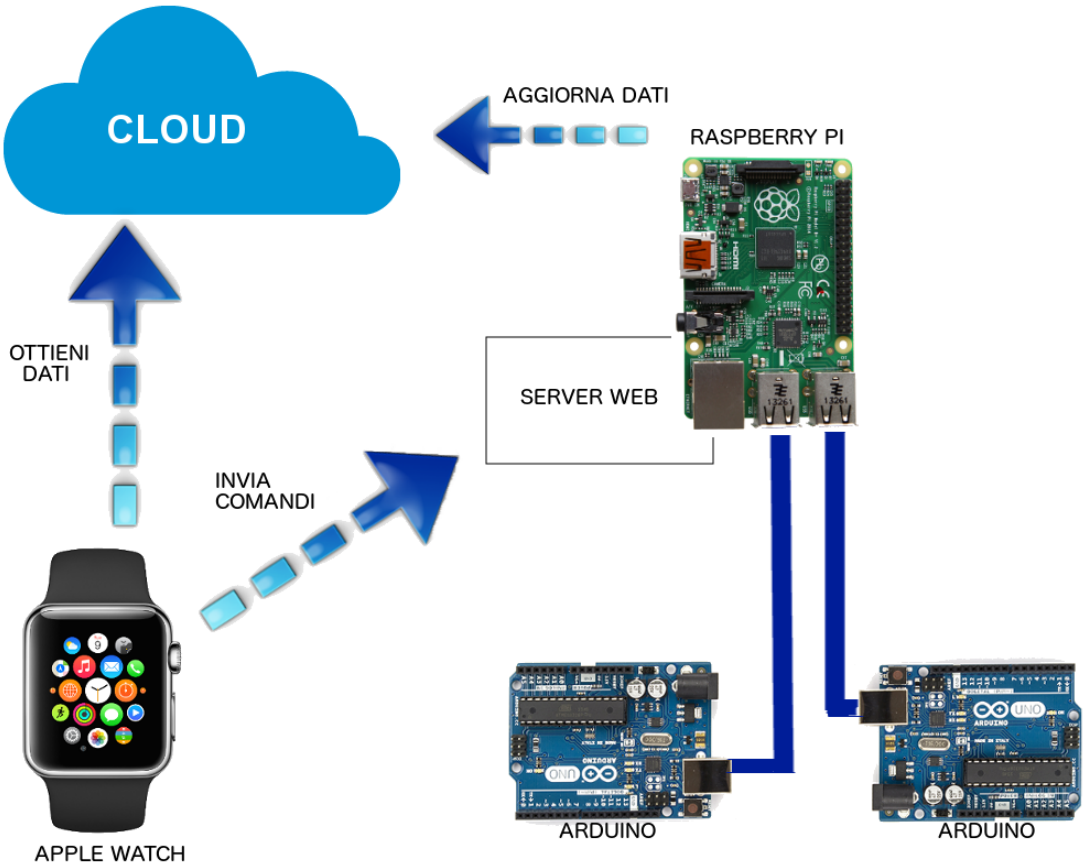

Figura 6.6: Struttura d'insieme del sistema

Raspberry Pi

Raspberry Pi è il cuore del sistema: in esso gira un server web dal quale può ricevere ed eseguire comandi da remoto. Inoltre, la connessione seriale ai due Arduino gli permette di ricevere e inviare aggiornamenti di stato sul sistema, aggiornando di conseguenza i dati memorizzati nel cloud.

Arduino

Il sistema comprende due Arduino: il primo, rivolto all'interazione con l'utente, permette di inserire una password tramite keypad per attivare o disattivare il sistema di allarme e di visualizzare informazioni sul sistema nel display installato; il secondo, rivolto all'analisi dei dati, utilizza un sensore

a infrarosso per ogni stanza per il rilevamento di intrusi e un sensore di temperatura per il calcolo di quest'ultima.

Apple Watch

Apple Watch è in grado di interagire con il sistema inviando richieste al server web di Raspberry Pi o prelevando le informazioni dal Cloud. Per esempio, è in grado di accendere e spegnere le luci di casa oppure attivare e disattivare l'allarme mandando una richiesta al server web. Può, invece, visualizzare lo stato del sistema effettuando una richiesta al Cloud.

Cloud

Il Cloud è un elemento chiave nella progettazione del sistema: esso, infatti, permette di salvare tutti i dati relativi alla casa domotica quali temperatura, stato di luci e allarme e eventuali foto scattate. Il Cloud fornisce, quindi, una panoramica dello stato della casa ai dispositivi che lo richiedono come Apple Watch.

6.2.4 Formalismi di comunicazione tra i componenti del sistema

Poiché il sistema è strutturato con diversi dispositivi, è stato necessario implementare un meccanismo di comunicazione che fosse comune a tutti. Sono stati, quindi, implementati due formalismi:

- Un formalismo per la comunicazione con il Cloud
- Un formalismo per la comunicazione all'interno del sistema

Comunicazione con il Cloud

La piattaforma cloud utilizzata, Carriots, utilizza il JSON come formato per il salvataggio dei dati nei propri server. Ogni volta che Raspberry Pi

vorrà aggiornare un'informazione presente in Carriots, dovrà inviargli una richiesta HTTP contenente i dati aggiornati in formato JSON.

Comunicazione all'interno del sistema

All'interno del sistema è stato implementato, invece, un diverso meccanismo di comunicazione. Ogni dispositivo comunica con un altro tramite scambio di messaggi, dove ogni messaggio viene inviato sotto forma di stringa seguendo questo template:

MITTENTE;DESTINATARIO;CONTENUTO:VALORE

Il mittente, il destinatario e il contenuto del messaggio sono separati da un punto e virgola; al contenuto può essere associato un valore, in tal caso quest'ultimo è preceduto dai due punti.

Il dispositivo che riceverà un messaggio in questo formato farà uso del *Parser*, implementato in esso, per tradurlo in un messaggio e allo stesso tempo potrà invocarlo per convertire un messaggio nella stringa da inviare.

6.3 Analisi di dettaglio

Procediamo nella fase di analisi dettagliata dei singoli dispositivi che compongono il sistema.

6.3.1 Arduino 1

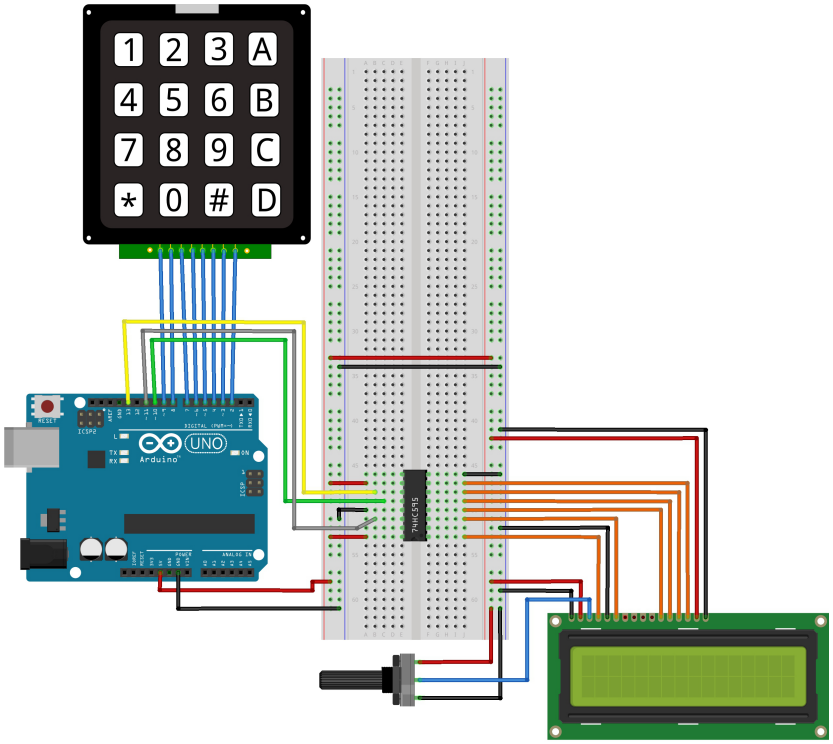

Figura 6.7: Schema dettagliato di Arduino 1 e i suoi componenti

Come detto in precedenza, il primo Arduino si occupa dell'interazione con l'utente, permettendogli di attivare o disattivare l'allarme tramite il keypad installato e di visualizzare informazioni sullo stato del sistema in un display LCD.

Per il keypad sono stati utilizzati otto pin di Arduino (dal 2 al 9), quattro per le righe e quattro per le colonne, formando una matrice 4x4.

Per lo schermo LCD, invece, è stato utilizzato uno shift register per diminuire l'utilizzo complessivo dei pin, che si riduce a tre (pin 10, 11, 13).

Sottosistemi in esecuzione su Arduino 1

Il sistema complessivo in esecuzione su Arduino 1 presenta diverse funzionalità:

- Attivazione e disattivazione dell'allarme
- Aggiornamento delle informazioni visualizzate sul display
- Lettura di un'eventuale rilevamento di intrusi
- Ricezione di messaggi da seriale
- Invio di messaggi sulla seriale

Ognuna di queste funzioni può essere rappresentata da un sottosistema, ognuno in esecuzione con una macchina a stati di dimensioni inferiori rispetto a quella corrispondente a un sistema monolitico.

Definiamo in Arduino 1 i seguenti sottosistemi:

- Alarm Setter
- Lcd Updater
- Message Receiver
- Message Sender
- Detect Presence

Alarm Setter

Alarm Setter è il sottosistema che si occupa della gestione dell'attivazione e della disattivazione dell'allarme.

Il suo stato iniziale è *Alarm Off*, in cui il sistema di allarme risulta disattivato.

Quando viene inserita la password tramite il keypad o viene ricevuto un messaggio dalla seriale in cui si richiede l'attivazione dell'allarme, il sottosistema transita allo stato *Alarm On*, in cui il sistema di allarme risulta abilitato.

Finché l'utente non inserisce la password corretta oppure il sistema non riceve un messaggio dalla seriale in cui si richiede la disattivazione dell'allarme, il sottosistema permane nello stato attuale. Se, invece, si verifica una delle condizioni precedenti, il sottosistema transita allo stato iniziale *Alarm Off*.

Lcd Updater

Lcd Updater è il sottosistema che si occupa dell'aggiornamento delle informazioni sullo stato del sistema.

Il suo stato iniziale è *Wait For Lcd Update*, in cui il sottosistema attende che venga notificato in caso di aggiornamenti del sistema.

Se è necessario aggiornare le informazioni sul display (per esempio l'allarme è stato attivato oppure è stata rilevata una presenza non autorizzata nell'abitazione) il sottosistema viene notificato e transita nello stato *Lcd updating* nel quale, dopo aver effettuato l'aggiornamento, ritorna allo stato iniziale.

Message Receiver

Message Receiver è il sottosistema che si occupa della ricezione di messaggi dalla seriale.

Il suo stato iniziale è *Waiting For Messages*, in cui il sottosistema attende eventuali aggiornamenti dalla seriale.

Se un nuovo messaggio è disponibile, il sottosistema transita nello stato di *Message Received* e ne legge il contenuto. Se un altro messaggio è disponibile, il sottosistema rimane nello stato attuale, altrimenti il sistema transita nello stato iniziale.

Message Sender

Message Sender è il sottosistema che si occupa dell'invio di messaggi sulla seriale.

In Arduino 1, il passaggio dallo stato di allarme attivo a quello disattivo e viceversa viene notificato a Raspberry Pi. Partendo da uno stato iniziale *Wait For Alarm State Changes*, in caso di cambiamento dello stato di allarme il sottosistema transita nello stato *Alarm State Changed*.

Dopo aver inviato il messaggio, il sottosistema ritorna nello stato iniziale in attesa di nuovi cambiamenti dello stato di allarme.

Detect Presence

Detect Presence è il sottosistema che si mette in ascolto di eventuali notifiche dal Message Receiver riguardo il rilevamento di una presenza non autorizzata all'interno della casa.

Il suo stato iniziale è *No Presence*, nel quale non ha ancora ricevuto nessuna notifica da parte del Message Receiver.

Se quest'ultimo riceve un messaggio di rilevamento presenza, notifica il sottosistema che transita nello stato di *Presence Detected*. Se vengono rilevate nuove presenze, il sottosistema rimane nello stato attuale, altrimenti transita allo stato iniziale in attese di nuove notifiche da parte del Message Receiver.

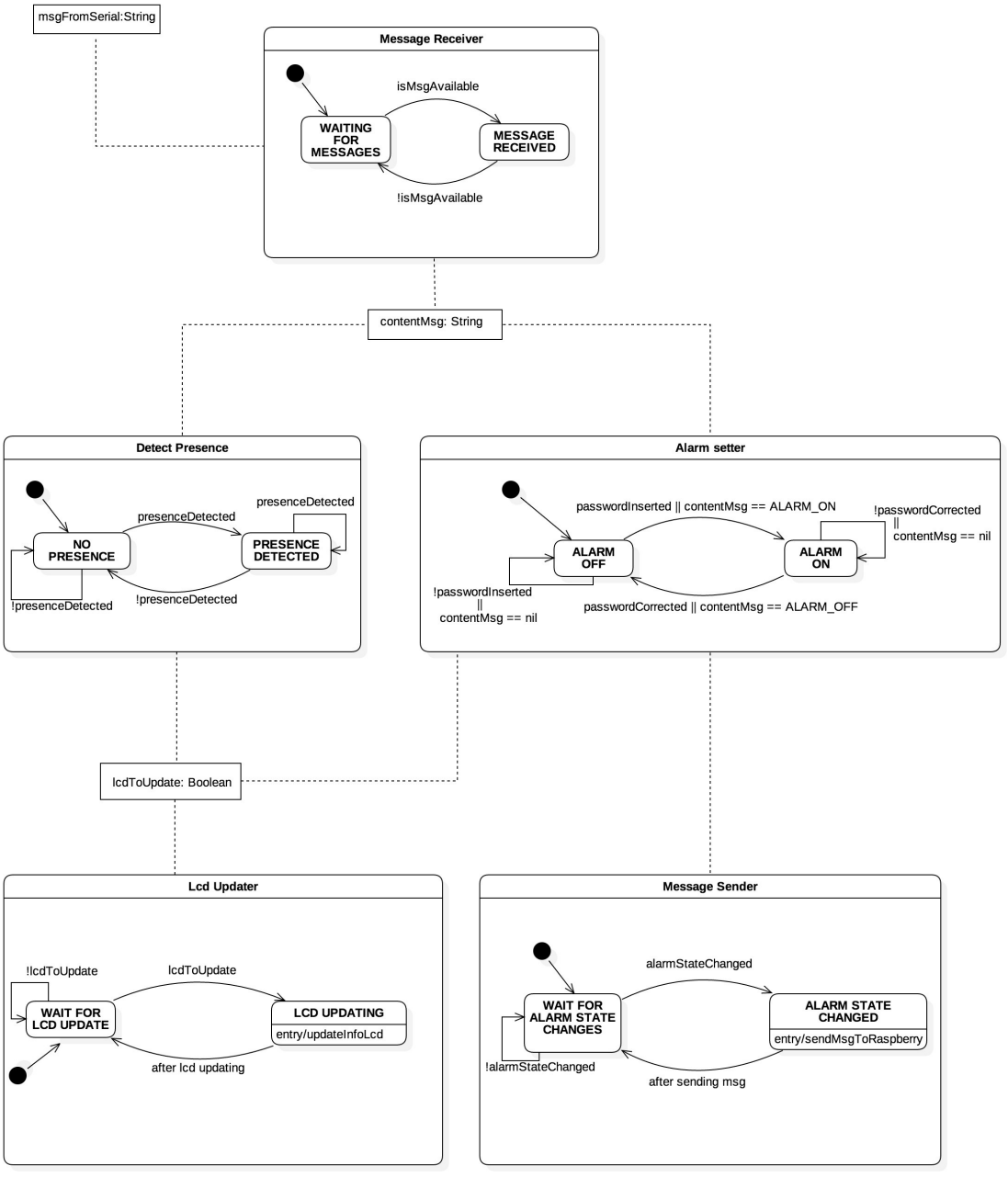

Figura 6.8: Interazione tra i sottosistemi in esecuzione su Arduino 1

Diagramma delle classi

Come è visibile nel diagramma delle classi, il sistema in esecuzione su Arduino 1 è suddiviso in task. Ogni task ha un metodo di inizializzazione e un metodo *tick* con il quale esegue il proprio compito.

Si distinguono cinque specifici task:

- **AlarmSetterTask**: si occupa dell'implementazione del sottosistema per la gestione del sistema di allarme. Intercetta i caratteri inseriti tramite il keypad, definito dalla classe *Keypad4x4* per memorizzare o confrontare la password, permettendo l'attivazione o la disattivazione dell'allarme.
- **MessageReceiverTask**: si occupa dell'implementazione del sottosistema per la ricezione di messaggi sulla seriale.
- **MessageSenderTask**: si occupa dell'implementazione del sottosistema per l'invio di messaggi sulla seriale.
- **DetectPresenceTask**: si occupa dell'implementazione del sottosistema che ha il compito di verificare se il *MessageReceiverTask* ha ricevuto un messaggio di rilevamento di una presenza non autorizzata all'interno dell'abitazione. Inoltre utilizza la classe *RoomsMap* per decifrare il codice della stanza ricevuto e convertirlo in un nome tramite l'apposito metodo.
- **LcdUpdaterTask**: si occupa dell'implementazione del sottosistema che ha il compito di aggiornare le informazioni nel display a fronte di eventi quali il cambiamento dello stato di allarme o il rilevamento di un intruso all'interno della casa.

Ogni task comunica con gli altri attraverso delle variabili condivise, definite all'interno della classe *Context*.

Inoltre, ogni task viene eseguito da una classe specifica *Scheduler*, che utilizza un periodo specificato tramite la classe *Timer*, che astrae il concetto di timer in Arduino.

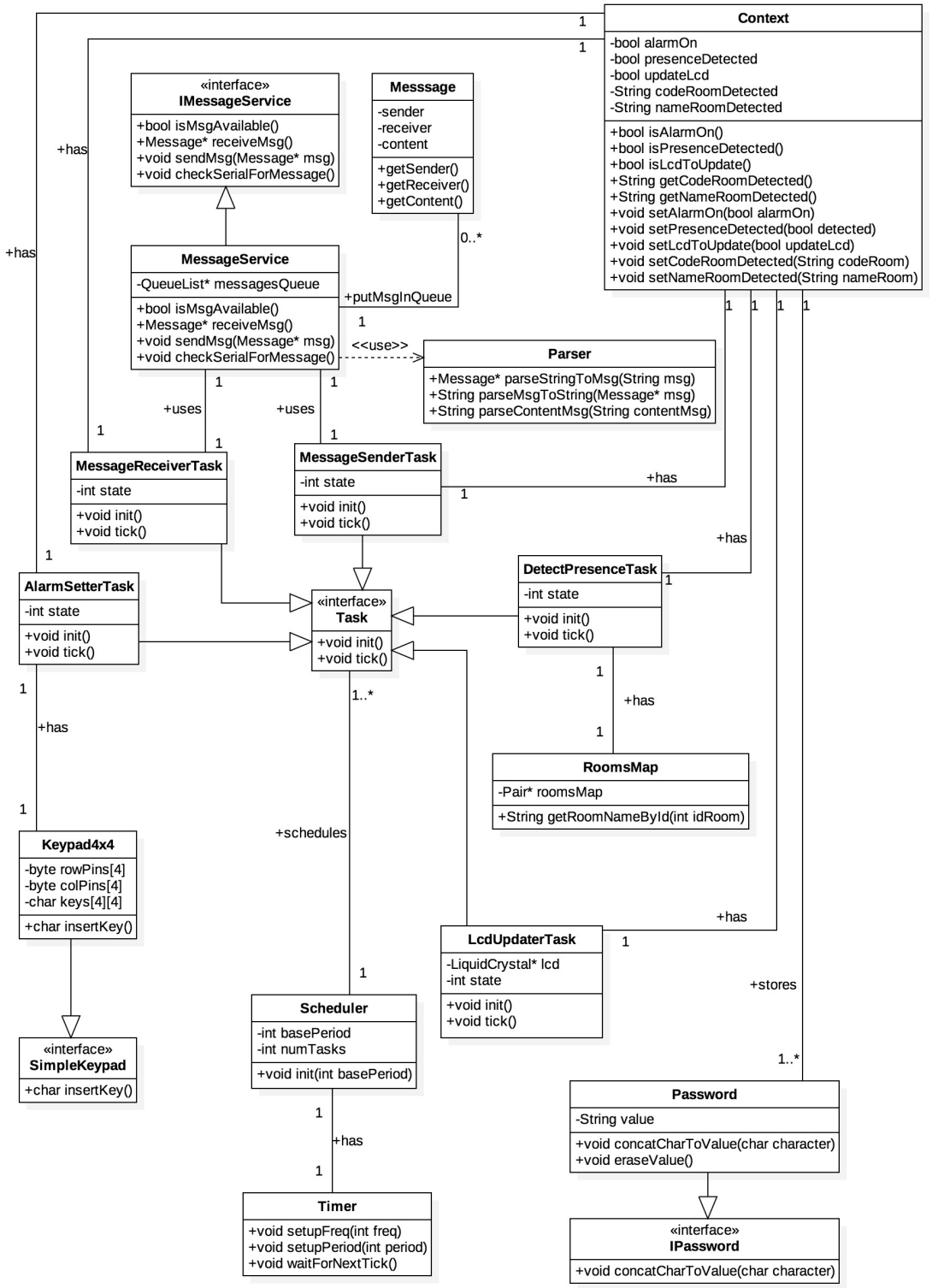

Figura 6.9: Diagramma delle classi del sistema in Arduino 1

Di rilievo è la classe *MessageService*, che incapsula i metodi per la ricezione e l'invio di un messaggio, che vengono utilizzati dal *MessageReceiverTask* e dal *MessageSenderTask*.

Per la traduzione dei messaggi viene utilizzata la classe *Parser* che permette di convertire un messaggio in stringa e viceversa e permette di analizzare il contenuto di ognuno.

6.3.2 Arduino 2

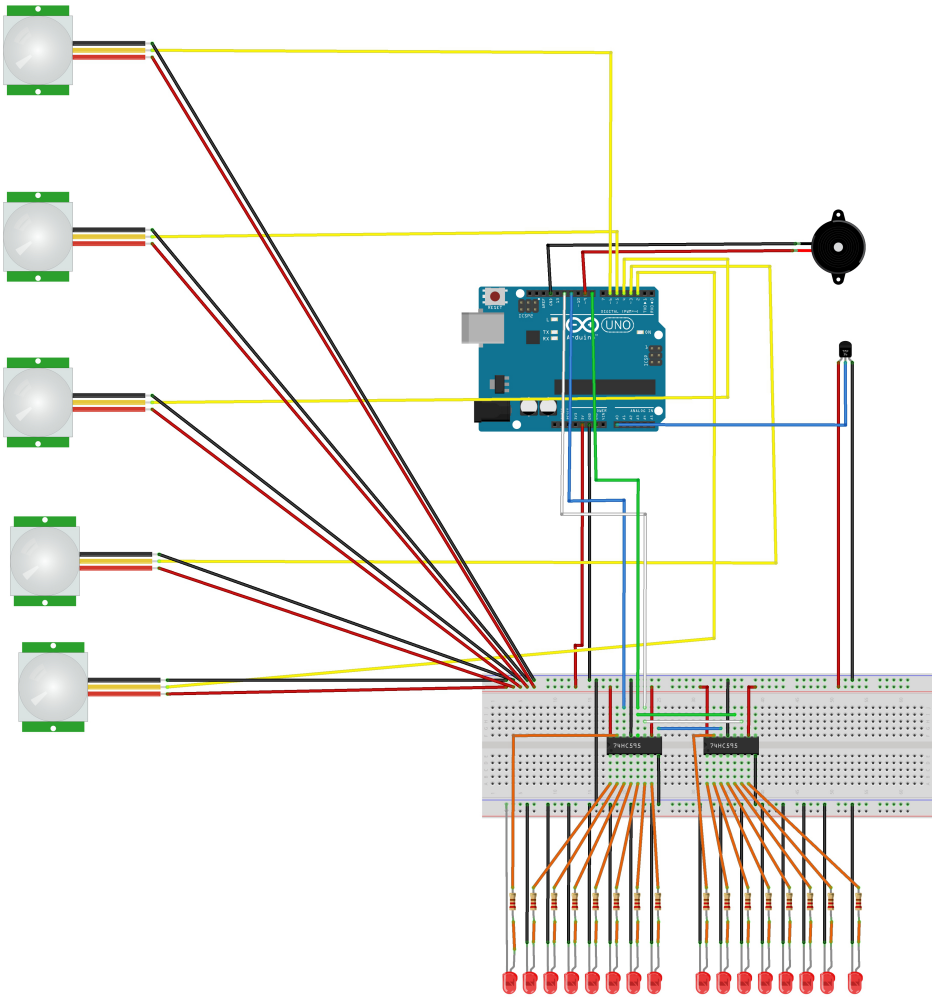

Figura 6.10: Schema dettagliato di Arduino 2 e i suoi componenti

Il secondo Arduino si occupa dell'acquisizione e gestione dei dati da parte dei sensori installati, nonché del controllo dell'impianto di illuminazione.

È costituito da cinque sensori a infrarosso (installati nei pin 2,4,5,6 e 7) che permettono di rilevare presenze non autorizzate all'interno dell'abitazione. Inoltre utilizza un buzzer (pin 9) per la segnalazione acustica dell'allarme e un sensore per il rilevamento della temperatura (pin analogico A0).

Per quanto riguarda l'illuminazione, vengono utilizzati tredici led collegati tramite shift register ad Arduino: tre di essi vengono utilizzati per mostrare il funzionamento del sistema di allarme mentre gli altri dieci sono adibiti all'illuminazione delle stanza (due per ognuna di esse).

Sottosistemi in esecuzione su Arduino 2

Così come in Arduino 1, anche nel secondo Arduino possiamo distinguere diverse funzionalità che è in grado di compiere:

- Rilevamento di una presenza non autorizzata tramite i sensori installati
- Gestione delle luci della casa
- Ricezione di messaggi sulla seriale
- Invio di messaggi sulla seriale
- Calcolo della temperatura di casa

Definiamo anche in Arduino 2 dei task specifici, ognuno dei quali svolge una funzionalità sopra descritta:

- Message Receiver
- Detect Presence
- Turn Light On Off
- Calculate Temperature
- Message Sender

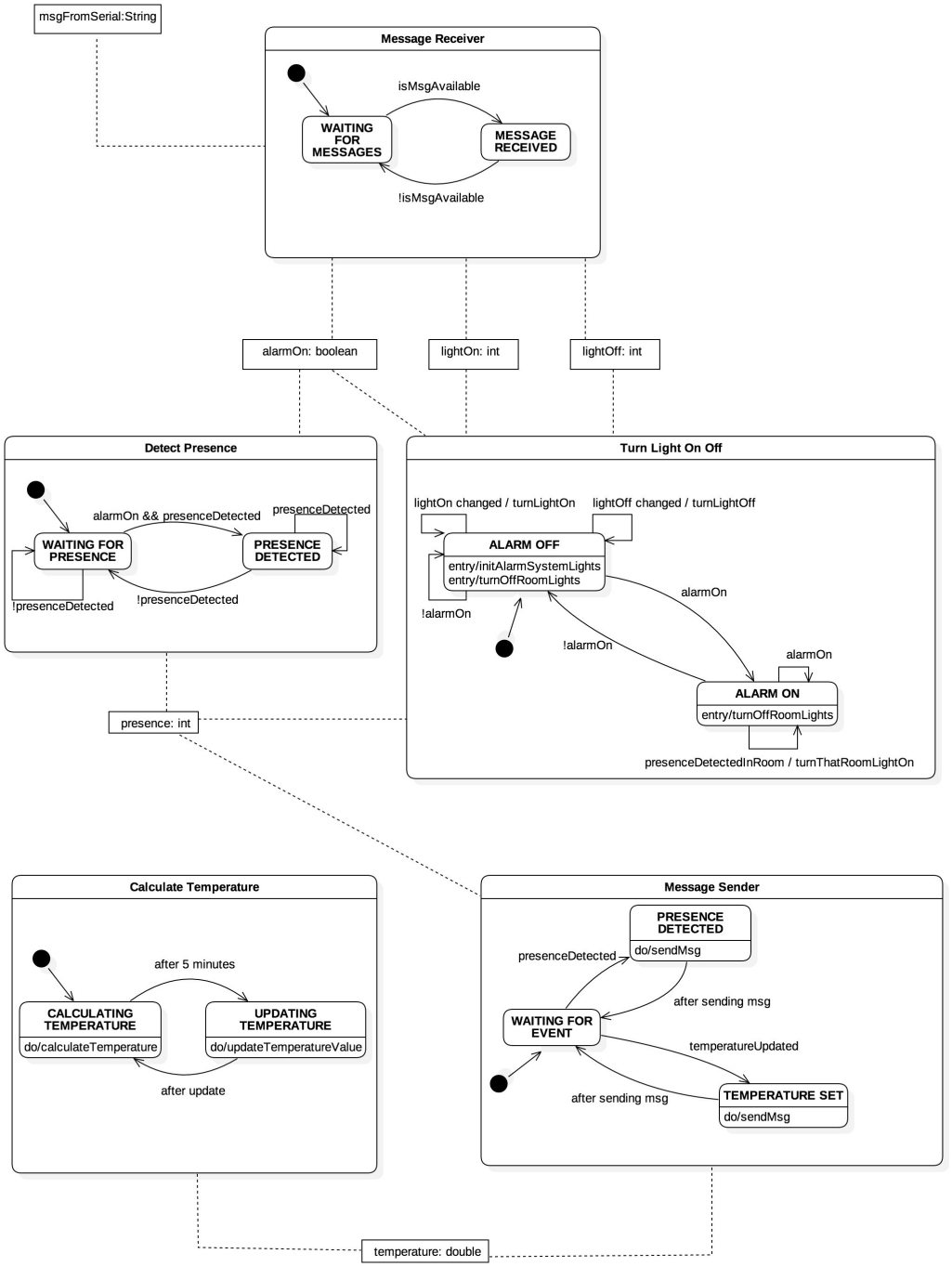

Figura 6.11: Interazione tra i sottosistemi in esecuzione su Arduino 2

Message Receiver

Message Receiver è il sottosistema che, come in Arduino 1, si occupa della ricezione di messaggi dalla seriale.

Il suo stato iniziale è *Waiting For Messages*, in cui il sottosistema attende eventuali aggiornamenti dalla seriale.

Se un nuovo messaggio è disponibile, il sottosistema transita nello stato di *Message Received* e ne legge il contenuto.

Il contenuto del messaggio può riguardare l'attivazione o disattivazione dell'allarme e l'accensione o lo spegnimento di una luce della casa.

Detect Presence

Detect Presence è il sottosistema che si occupa del rilevamento di intrusi all'interno dell'abitazione.

Il suo stato iniziale è *Waiting For Presence* nel quale il sottosistema attende il rilevamento di intrusi.

Se il sistema di allarme è attivo e il sensore ha rilevato una presenza all'interno dell'abitazione, il sottosistema transita nello stato *Presence Detected*, dove il codice della stanza oggetto di rilevamento viene salvato in una variabile condivisa.

Terminato il salvataggio, il sistema transita nuovamente nello stato iniziale in attesa di nuovi rilevamenti.

Turn Light On Off

Turn Light On Off è il sottosistema che si occupa dell'accensione e dello spegnimento delle luci.

Il suo stato iniziale è *Alarm Off* nel quale l'utente può accendere e spegnere le luci di casa dal proprio smartwatch. Se infatti al *Message Receiver* è giunto un messaggio dal contenuto *Light On*, la luce specificata verrà accesa; se invece è giunto un messaggio dal contenuto *Light Off*, la luce specificata nel messaggio verrà spenta.

Se l'allarme viene attivato, il sottosistema transita nello stato *Alarm On* dove l'utente non ha più il controllo delle luci. L'eventuale accensione di una luce, infatti, è delegata al sistema di sicurezza che, ogni volta che rileva un intruso, salva in una variabile condivisa il codice della stanza che permette a questo sottosistema di accendere la relativa luce.

Se l'allarme viene disattivato, il sottosistema transita nello stato iniziale.

Calculate Temperature

Calculate Temperature è il sottosistema che si occupa di prelevare i dati dal sensore di temperatura ed effettuarne il calcolo preciso.

Il suo stato iniziale è *Calculating Temperature*, nel quale il sottosistema calcola i valori della temperatura leggendo i dati dal sensore.

Dopo cinque minuti il sottosistema transita nello stato *Updating Temperature*, dove riceve una media matematica dei valori precedentemente calcolati e salva il nuovo valore nella variabile condivisa.

Una volta terminato il salvataggio, ritorna allo stato iniziale, procedendo ad un nuovo calcolo.

Message Sender

Message Sender è il sottosistema che si occupa di inviare messaggi sulla seriale a fronte di eventi.

Il suo stato iniziale è *Waiting For Event*, nel quale il sottosistema attende che si verifichi uno specifico evento. Ne possono accadere due:

- Viene rilevata una presenza non autorizzata: in questo caso il sottosistema transita nello stato *Presence Detected* e invia il messaggio sulla seriale.

- Viene aggiornato il valore della temperatura: in questo caso il sottosistema transita nello stato *Temperature Set* e invia il messaggio sulla seriale.

Diagramma delle classi

Come si evince dal diagramma delle classi, anche il sistema in esecuzione su Arduino 2 è suddiviso in task. Ogni task ha un metodo di inizializzazione e un metodo *tick* con il quale esegue il proprio compito.

Si distinguono cinque task specifici:

- **DetectPresenceTask**: si occupa dell'implementazione del sottosistema per il rilevamento di una presenza non autorizzata all'interno dell'abitazione. In particolare, utilizza sensori ad infrarossi definiti dalla classe *Pir* che verificano il rilevamento di intrusi tramite il metodo *checkPresence()* e un buzzer, definito dalla classe *Buzzer*, che implementa metodi per la segnalazione acustica.

- **TurnLightOnOffTask**: si occupa dell'implementazione del sottosistema per l'accensione e lo spegnimento delle luci della casa. Utilizza la classe *ShiftOutManager* per scrivere su shift register e accendere i led di interesse.

- **CalculateTemperatureTask**: si occupa dell'implementazione del sottosistema per il calcolo della temperatura. Utilizza un sensore, definito dalla classe *TemperatureSensor*, che tramite il metodo *getTemperature()* restituisce il valore di temperatura letto.

- **MessageReceiverTask**: come in Arduino 1, si occupa dell'implementazione del sottosistema per la ricezione di messaggi sulla seriale.

- **MessageSenderTask**: come in Arduino 1, si occupa dell'implementazione del sottosistema per l'invio di messaggi sulla seriale.

Anche in Arduino 2 i task comunicano tramite variabili condivise, definite dalla classe *Context* e vengono eseguiti da uno scheduler definito dalla classe *Scheduler*.

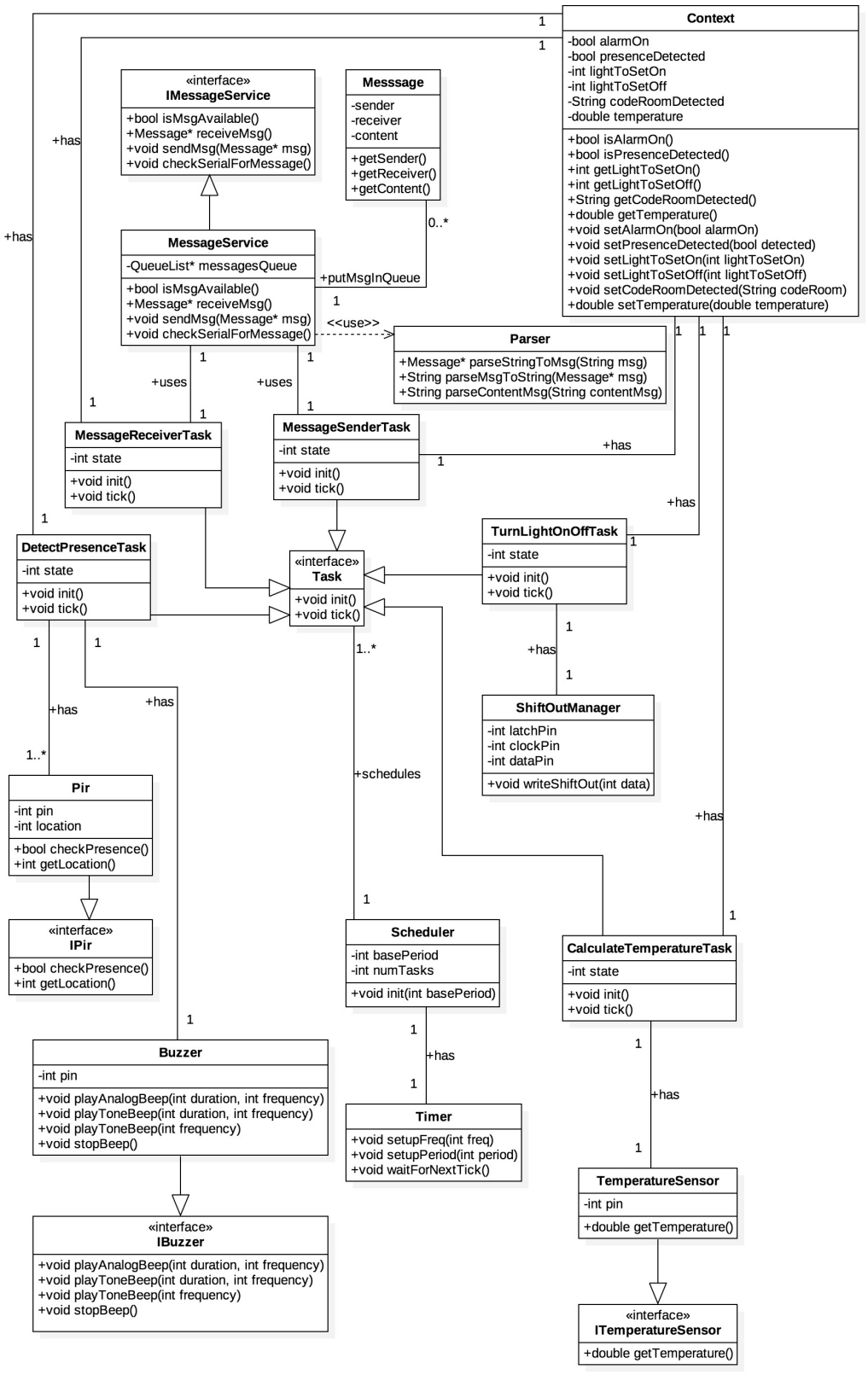

Figura 6.12: Diagramma delle classi del sistema in Arduino 2

6.3.3 Raspberry Pi

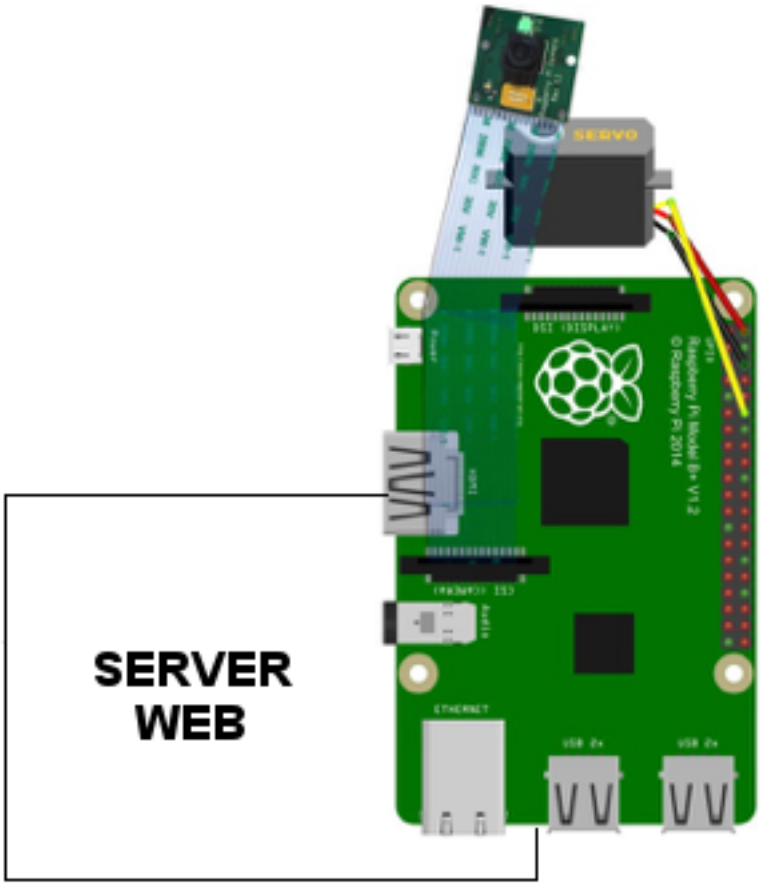

Figura 6.13: Schema dettagliato di Raspberry Pi

Analisi delle funzionalità

Raspberry Pi implementa diverse funzionalità: una delle principali è quella di inoltrare correttamente i messaggi che riceve ai destinatari specificati.

Come abbiamo visto in precedenza, infatti, il sistema comunica tramite scambio di messaggi, dove di ognuno si specifica mittente, destinatario e contenuto. Raspberry Pi preleva il destinatario dal messaggio e invia il contenuto al dispositivo specificato.

I dispositivi possono comunicare con Raspberry Pi tramite seriale (nel progetto, i due Arduino) oppure tramite il server web in esecuzione su di esso (per esempio, Apple Watch inviando comandi da remoto).

Raspberry Pi, inoltre, utilizza un servo motore per il posizionamento della fotocamera (nel progetto, Raspberry Pi Camera) nella stanza in cui è stato rilevato movimento. Una volta scattata la foto, questa viene inviata via email all'utente.

Inoltre Raspberry Pi salva tutti i dati ricevuti e trasmessi nel Cloud, in modo che siano accessibili da diversi dispositivi come, ad esempio, Apple Watch.

Diagramma delle classi

Una delle classi di maggiore importanza nel sistema in esecuzione su Raspberry Pi è *MsgService*. Questa classe implementa i metodi necessari per la ricezione e l'invio di messaggi tramite thread e, inoltre, ne analizza il contenuto, eseguendo delle ulteriori azioni se necessarie.

Essa contiene due classi innestate:

- **Receiver**: estende da *Thread* e si occupa della ricezione di messaggi dalla seriale
- **Sender**: anch'essa estende da *Thread* e si occupa dell'invio di un messaggio sulla seriale

La seriale è definita dalla classe *SerialCommChannel*, che implementa l'interfaccia *CommChannel*, nella quale sono implementati i metodi per la lettura e la scrittura di dati sulla seriale.

MsgService, inoltre, delega alcuni funzioni ad altre classi e nello specifico:

- **CarriotsRequester**: tramite il metodo *sendStreamToCarriots(String dataToStream)* esegue un nuovo thread che salva nei server cloud di Carriots lo stream di dati

- **EmailService**: tramite il metodo *sendEmailWithAttachment(String destination, String filePath)* esegue un nuovo thread che invia all'utente un'email con in allegato la foto della stanza in cui è stato rilevato un intruso

- **PhotoUploader**: tramite il metodo *uploadImageToCloud(String path, String name)* esegue un nuovo thread che salva nel Cloud le foto scattate dal sistema.

- **FileNamer**: tramite il metodo *getFileName()* restituisce un nome da assegnare al file corrispondente alla foto scattata dal sistema.

- **Parser**: implementa i metodi per l'analisi e la comprensione dei messaggi ricevuti dai dispositivi. Permette tramite il metodo *parseStringToMsg(String msg)* di convertire una stringa nel suo messaggio corrispondente; tramite il metodo *parseMsgToString(Msg msg)* di convertire un messaggio nella stringa corrispondente; tramite il metodo *parseContentMsg(String contentMsg)* di analizzare il contenuto del messaggio.

Il software in esecuzione su Raspberry Pi è formato da una schermata principale, definita dalla classe *MainFrame*, i cui eventi vengono intercettati dalla classe *MainFrameController*. Tramite il metodo *initSession()* questa classe inizializza una connessione con i due Arduino ed esegue il thread *ServerRequestReceiver* per ricevere eventuali richieste da remoto.

Per ogni richiesta ricevuta, viene eseguito il thread *ServerRequestReceiverThread* che preleva l'eventuale messaggio ricevuto e richiama il *MsgService* per analizzarlo.

Il modello del software è costituito dal servo motore, definito dalla classe *Servo*, che tramite il metodo *rotate(int position)* permette di ruotare la fotocamera, definita dalla classe *Camera*, nella giusta posizione. In seguito la fotocamera, una volta posizionata, scatta una fotografia tramite il metodo *takePhoto()*.

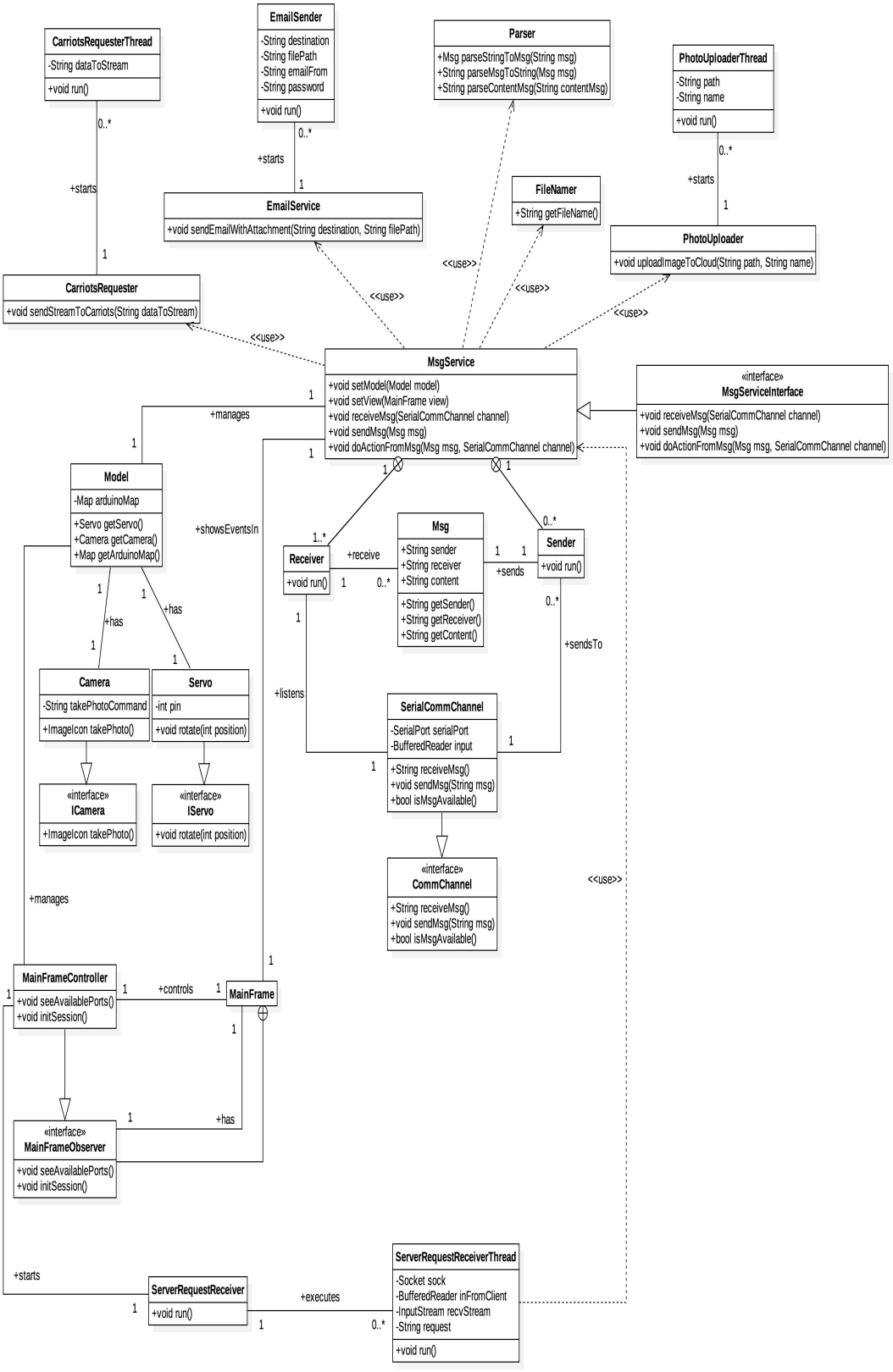

Figura 6.14: Diagramma delle classi del software in esecuzione su Raspberry

6.3.4 Apple Watch

Figura 6.15: Applicazione di progetto in esecuzione su Apple Watch

Analisi delle funzionalità

L'applicazione in esecuzione su Apple Watch permette l'interazione dell'utente con la casa domotica, permettendo di controllare aspetti come il sistema di allarme, l'impianto di illuminazione e lo stato attuale dell'abitazione.

Ad ogni richiesta, l'applicazione interroga il Cloud affinché questo gli invii le informazioni di interesse.

L'applicazione è, inoltre, in grado di inviare richieste al server web in esecuzione su Raspberry Pi in modo tale da soddisfare i comandi dell'utente.

Fra i comandi principali vi sono l'accensione e lo spegnimento dell'allarme e delle luci di casa mentre fra le informazioni disponibili vi sono lo stato dell'allarme, lo stato delle luci, la temperatura e la foto di un eventuale intruso all'interno della nostra abitazione.

Diagramma delle classi

Il diagramma mostra i *controller* che si occupano di gestire le varie view (in Apple Watch definite come *Interface*) che compongono l'applicazione.

Si evidenziano:

- **MainInterfaceController**: è l'interfaccia principale che viene visualizzata dall'utente, mostrando il menù con le varie opzioni. Da questa interfaccia l'utente è in grado di raggiungere tutte le altre.
- **AlarmInterfaceController**: è l'interfaccia che permette di visualizzare lo stato del sistema di allarme e di attivarlo o disabilitarlo a seconda delle esigenze.
- **InsertPasswordInterfaceController**: è l'interfaccia che permette di inserire una password nel caso in cui si voglia attivare il sistema di allarme.
- **RoomInterfaceController**: è l'interfaccia che mostra all'utente le informazioni principali riguardo ad una stanza della casa e permette di accendere e spegnere le luci di essa.
- **ImageViewerInterfaceController**: è l'interfaccia che permette di visualizzare l'ultima foto scattata in una stanza dove è stato rilevato movimento non autorizzato. È richiamata dall'interfaccia precedente.
- **TemperatureInterfaceController**: è l'interfaccia che permette di visualizzare la temperatura di casa.

Tutte queste classi descritte ereditano da *WKInterfaceController*, che rappresenta una generica interfaccia grafica all'interno di Apple Watch.

Inoltre parte di queste classi effettuano richieste al Cloud per ottenere le informazioni necessarie al corretto funzionamento dell'applicazione.

Queste richieste vengono delegate ad un'altra classe, definita come *CarriotsRequester*.

Questa classe utilizza i seguenti metodi:

- **makeHttpRequestToCarriots(String url)**: dato l'url corretto, questo metodo effettua una richiesta Http ai server di Carriots per ottenere il JSON corrispondente a quella richiesta.
- **getJSONFromId(int id)**: dato l'id del JSON memorizzato sui server di Carriots, questo metodo crea l'URL corrispondente e fa una richiesta Http per ottenerlo.
- **isRoomLightOn(JSON json)**: questo metodo analizza un JSON, ottenuto da una precedente richiesta, per verificare se la luce di una specifica stanza è accesa o meno.
- **isAlarmOn(JSON json)**: questo metodo analizza un JSON, ottenuto da una precedente richiesta, per verificare se l'allarme è abilitato o meno.
- **isPresenceDetected(JSON json**: questo metodo analizza un JSON, ottenuto da una precedente richiesta, per verificare se, in una specifica stanza, è stato rilevato un intruso o meno.
- **getTemperature(JSON json)**: questo metodo preleva da un JSON, ottenuto in precedenza, il valore aggiornato della temperatura all'interno dell'abitazione.

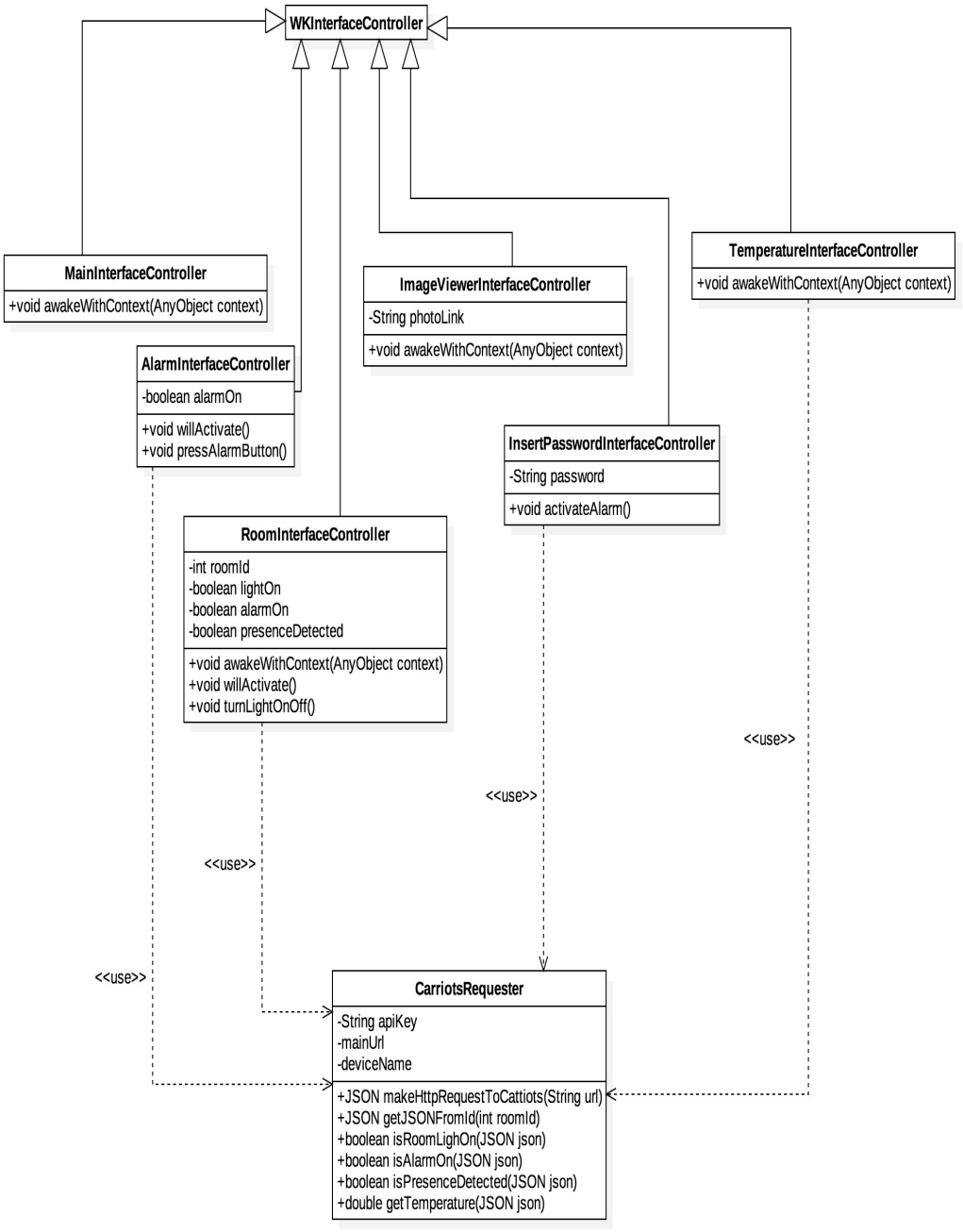

Figura 6.16: Diagramma delle classi del software realizzato per Apple Watch

6.4 Progettazione

Segue la progettazione delle principali operazioni dei dispositivi che compongono il sistema.

6.4.1 Arduino 1

Attivazione del sistema di allarme da parte dell'utente

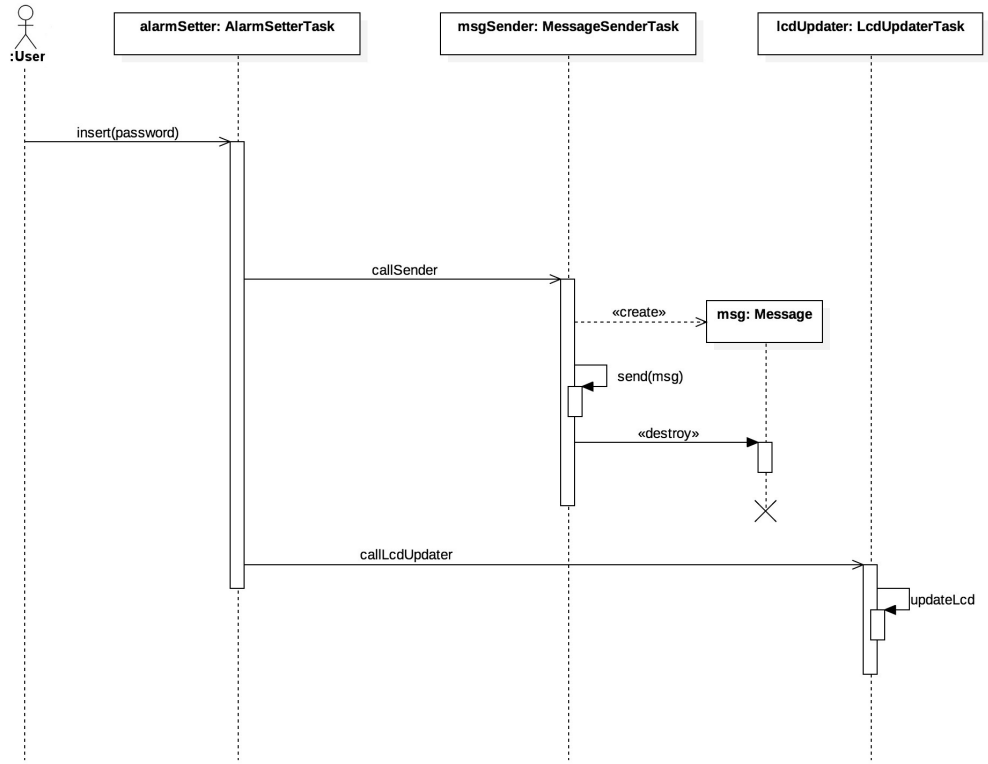

Figura 6.17: Diagramma di sequenza che mostra l'attivazione del sistema di allarme da parte dell'utente

Quando l'utente inserisce una password, viene richiamato *AlarmSetterTask* che provvede a creare un nuovo messaggio, inviandolo a *MessageSenderTask* che si occupa dell'invio sulla seriale. Inoltre, invia un messaggio a *LcdUpdaterTask* che aggiorna le informazioni nel display installato.

Disattivazione del sistema di allarme da parte dell'utente

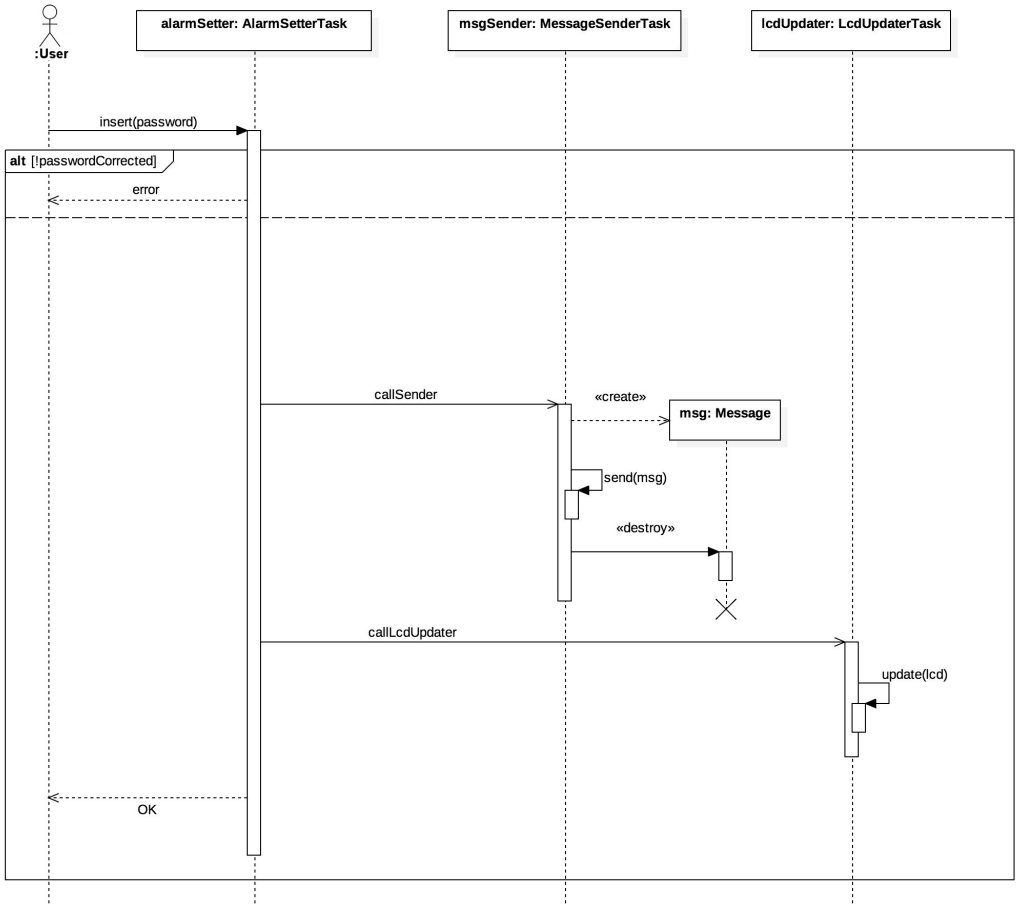

Figura 6.18: Diagramma di sequenza che mostra la disattivazione del sistema di allarme da parte dell'utente

Il diagramma presenta molte caratteristiche in comune con quello precedente. Si differenzia in quanto viene effettuato un controllo sulla password inserita:

- Se non è corretta, viene restituito un errore all'utente.
- Se è corretta, si procede con lo stesso meccanismo descritto nel diagramma precedente.

Reazione del sistema a fronte di una ricezione di messaggio

Il diagramma mostra come il sistema reagisce all'arrivo di un messaggio nella seriale.

Si distinguono quattro casi:

- **Ricevuto messaggio di spegnimento allarme**: in questo caso è giunta da remoto una richiesta di disattivare il sistema di allarme. Il sistema procede segnalando ad *AlarmSetterTask* di effettuare la disattivazione il quale, a sua volta, richiama *MessageSenderTask* e *LcdUpdaterTask*. Il primo crea un messaggio con cui segnalare l'evento al secondo Arduino e Raspberry Pi mentre il secondo procede all'aggiornamento delle informazioni sul display installato.

- **Ricevuto messaggio di impostazione password**: questo messaggio precede quello di accensione allarme e permette di impostare una nuova password per il sistema di allarme. Una volta letto il contenuto da parte del *MessageReceiverTask*, viene impostata la variabile corrispondente nel contesto condiviso tra i vari task.

- **Ricevuto messaggio di accensione allarme**: in questo caso, come per lo spegnimento, è giunta da remoto una richiesta di attivare il sistema di allarme. Il sistema segnala ad *AlarmSetterTask* di procedere con l'attivazione il quale, a sua volta, richiama *MessageSenderTask*, che segnala l'evento al secondo Arduino e Rasberry Pi tramite messaggio e *LcdUpdaterTask* che procede all'aggiornamento delle informazioni sul display installato.

- **Ricevuto messaggio di rilevamento intruso**: questo messaggio segnala la presenza di un intruso all'interno dell'abitazione. Viene richiamato il *DetectPresenceTask* che, dopo aver decodificato l'id ricevuto con il nome della stanza corrispondente, richiama *LcdUpdaterTask* per procedere all'aggiornamento delle informazioni nel display.

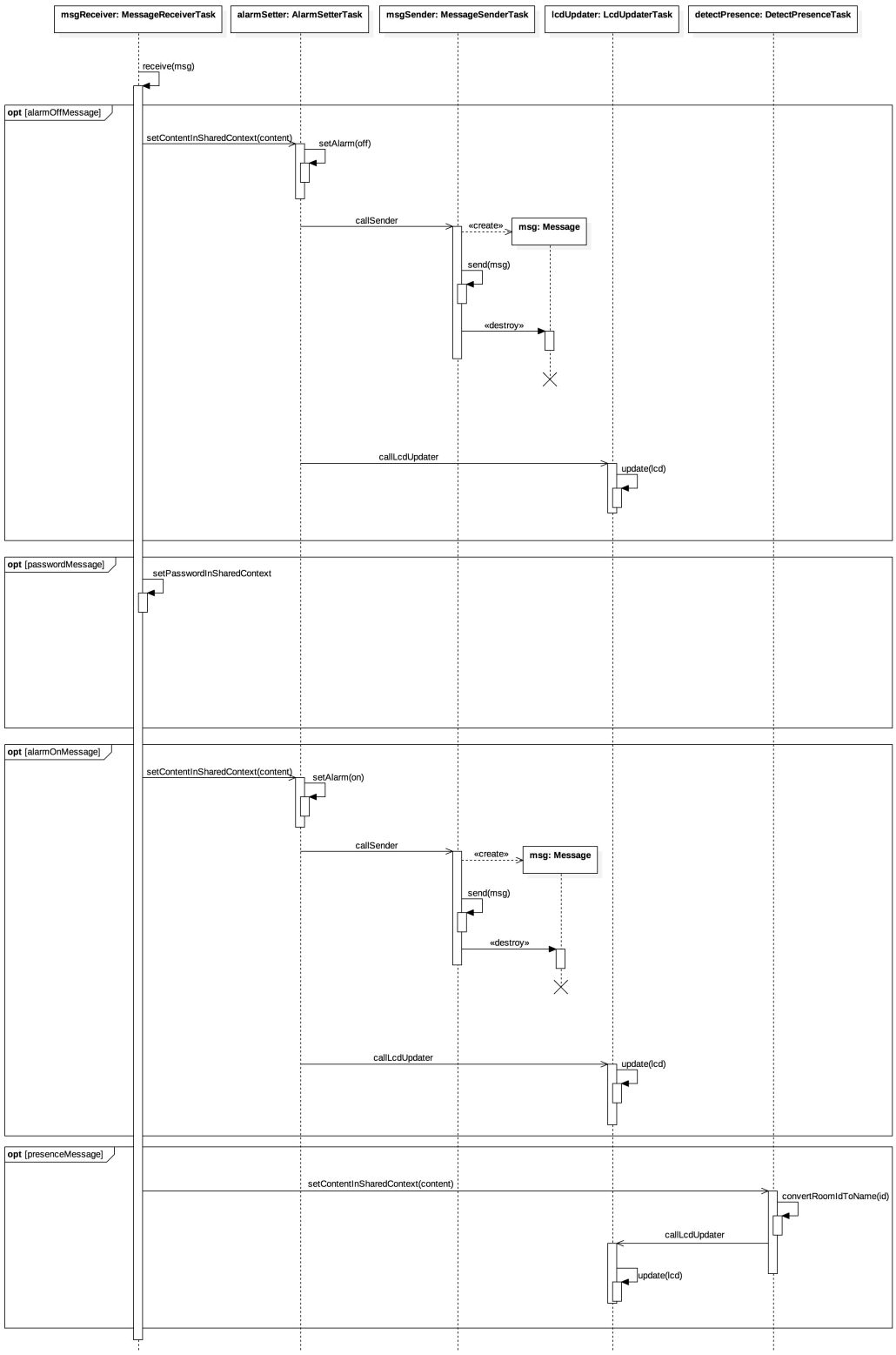

Figura 6.19: Diagramma di sequenza che mostra il comportamento del sistema alla ricezione di un messaggio

6.4.2 Arduino 2

Rilevamento di un intruso

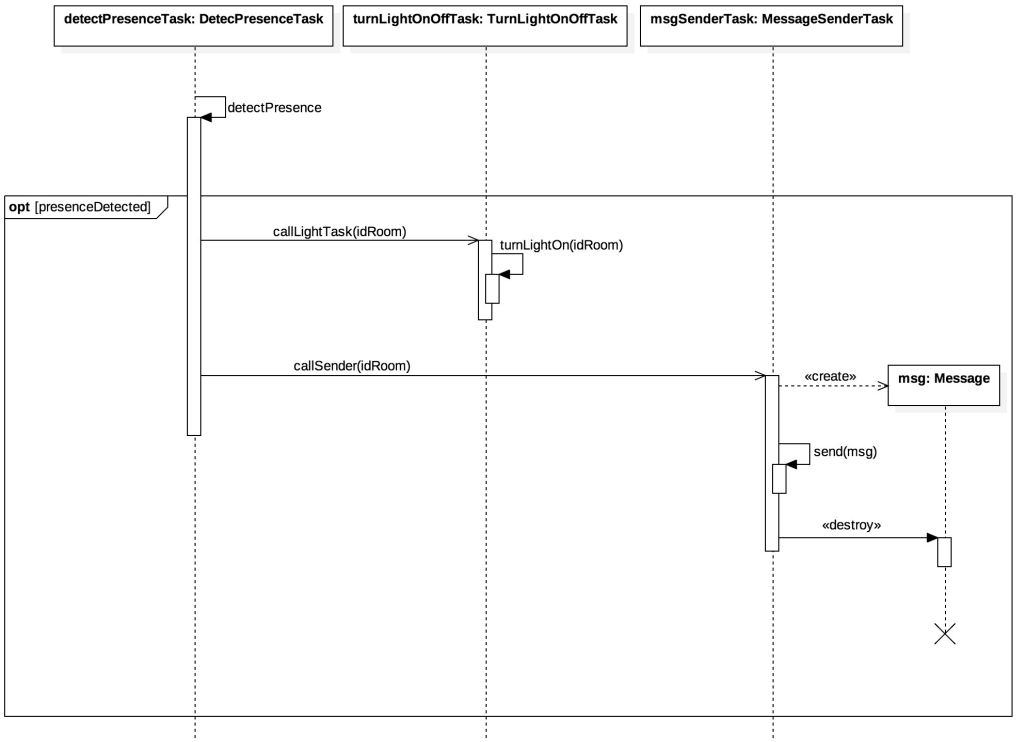

Figura 6.20: Diagramma di sequenza che mostra il comportamento del sistema al rilevamento di un intruso

Quando *DetectPresenceTask* rileva un intruso all'interno dell'abitazione, viene richiamato *TurnLightOnOffTask* che accende le luci alla stanza corrispondente all'id della stessa segnalato dal *DetectPresenceTask*.

Inoltre viene richiamato *MessageSenderTask* affinché invii l'id della stanza al secondo Arduino, notificandolo, quindi, di aver rilevato la presenza di un intruso.

Calcolo della temperatura

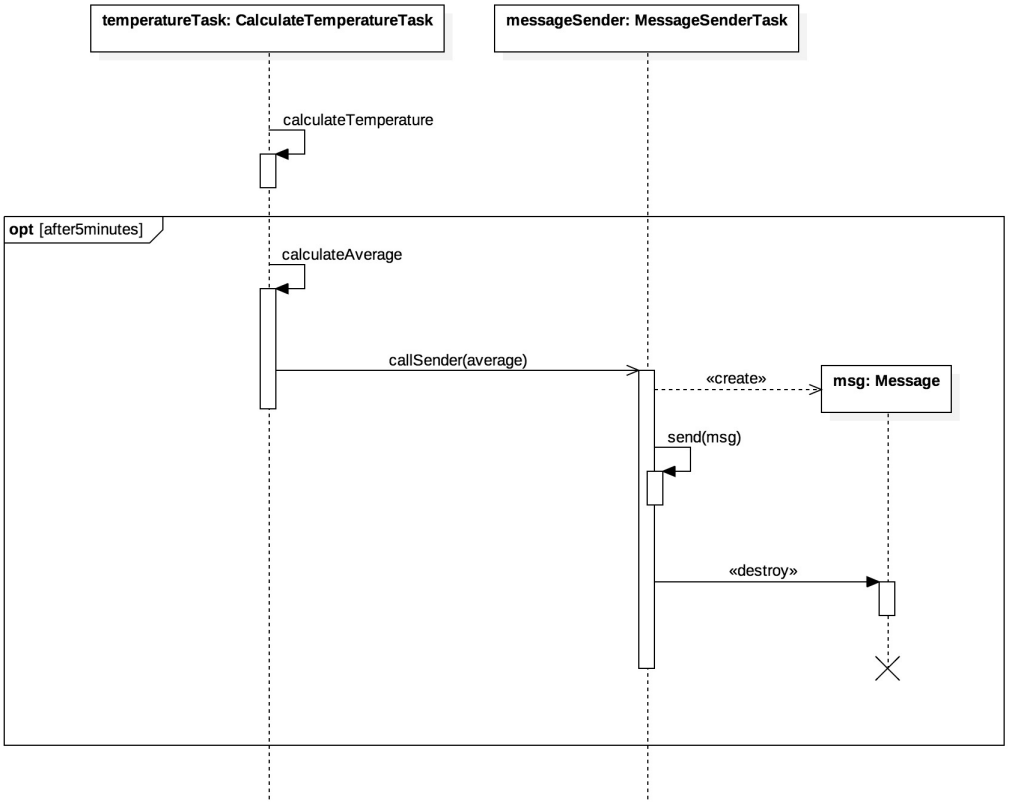

Figura 6.21: Diagramma di sequenza che mostra il comportamento del sistema nel calcolo della temperatura

TemperatureTask calcola in loop i valori di temperatura che riceve dal sensore. Dopo cinque minuti, il sistema calcola una media dei valori riscontrati e richiama il *MessageSender*. Quest'ultimo crea un nuovo messaggio e lo invia sulla seriale, che in seguito verrà letto da Raspberry Pi che procede al salvataggio del valore nel Cloud.

Reazione del sistema a fronte di una ricezione di messaggio

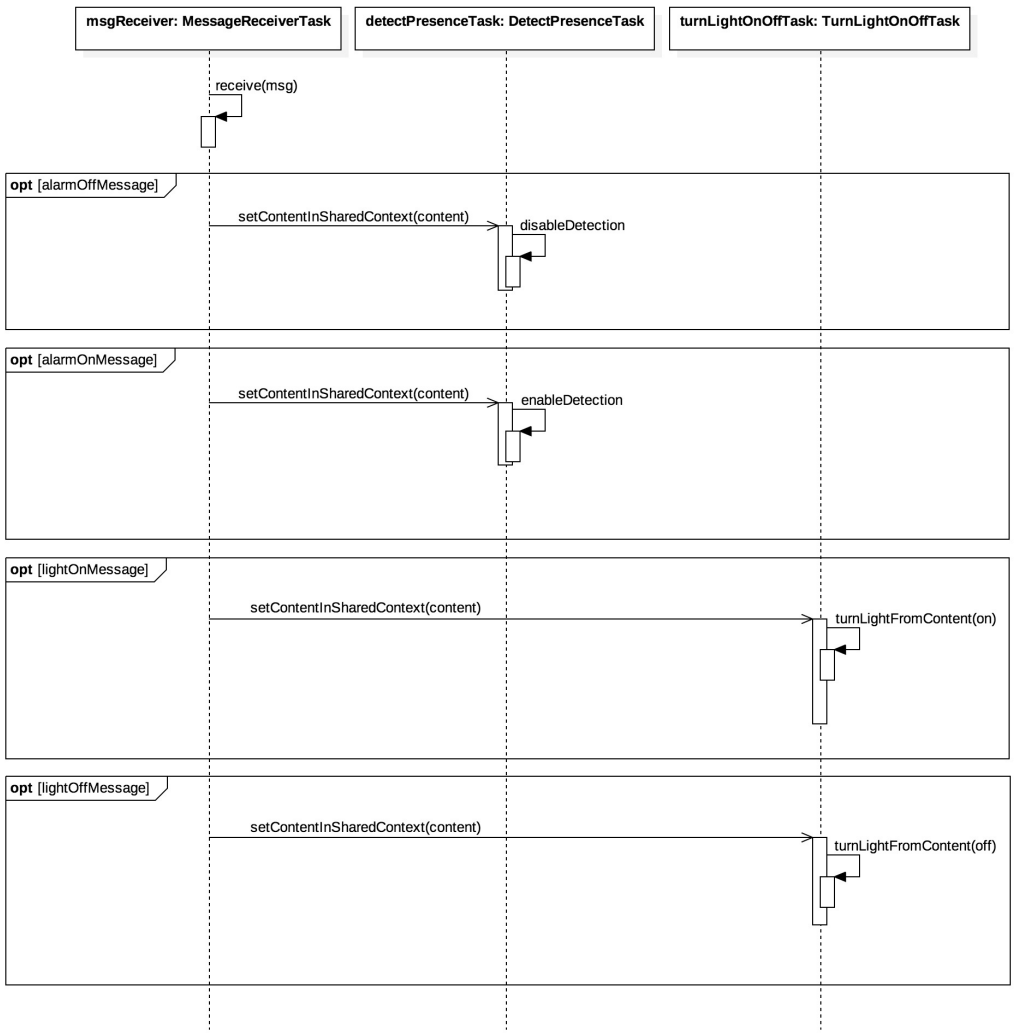

Figura 6.22: Diagramma di sequenza che mostra il comportamento del sistema alla ricezione di un messaggio

Il diagramma mostra come il sistema reagisca all'arrivo di un messaggio nella seriale.

Anche in Arduino 2 si distinguono quattro casi:

- **Ricevuto messaggio di spegnimento allarme**: ricevuto questo messaggio, *MessageReceiverTask* aggiorna il valore della variabile, as-

sociata allo stato dell'allarme, nel contesto condiviso. Viene, in seguito, richiamato *DetectPresenceTask* che disabilita il sistema di rilevamento di intrusi.

- **Ricevuto messaggio di accensione allarme**: ricevuto questo messaggio, *MessageReceiverTask* aggiorna il valore della variabile condivisa associata allo stato dell'allarme. Viene poi richiamato *DetectPresenceTask* che abilita il sistema al rilevamento di intrusi all'interno dell'abitazione.

- **Ricevuto messaggio di accensione luce**: ricevuto questo messaggio, *MessageReceiverTask* aggiorna il valore della variabile condivisa, impostando l'id della stanza di cui vanno accese le luci. Successivamente viene richiamato *TurnLighOnOffTask* che procede all'accensione delle luci della corrispondente stanza.

- **Ricevuto messaggio di spegnimento luce**: ricevuto questo messaggio, *MessageReceiverTask* imposta il valore della variabile condivisa con l'id della stanza di cui si vogliono spegnere le luci. Successivamente viene richiamato *TurnLightOnOffTask* che procede allo spegnimento delle luci della stanza corrispondente.

6.4.3 Raspberry Pi

Reazione del sistema a fronte di una ricezione di messaggio

Il diagramma di sequenza mostra il funzionamento del sistema in esecuzione su Raspberry Pi.

Come detto in precedenza, Raspberry Pi ha la funzione principale di veicolare i messaggi da un dispositivo ad un altro, analizzandone il contenuto e, dove necessario, eseguire ulteriori azioni.

Si distinguono sei casi:

- **Ricevuto un messaggio di inizializzazione**: quando il *Receiver* riceve questo messaggio, richiama il *MsgService* che ne legge il con-

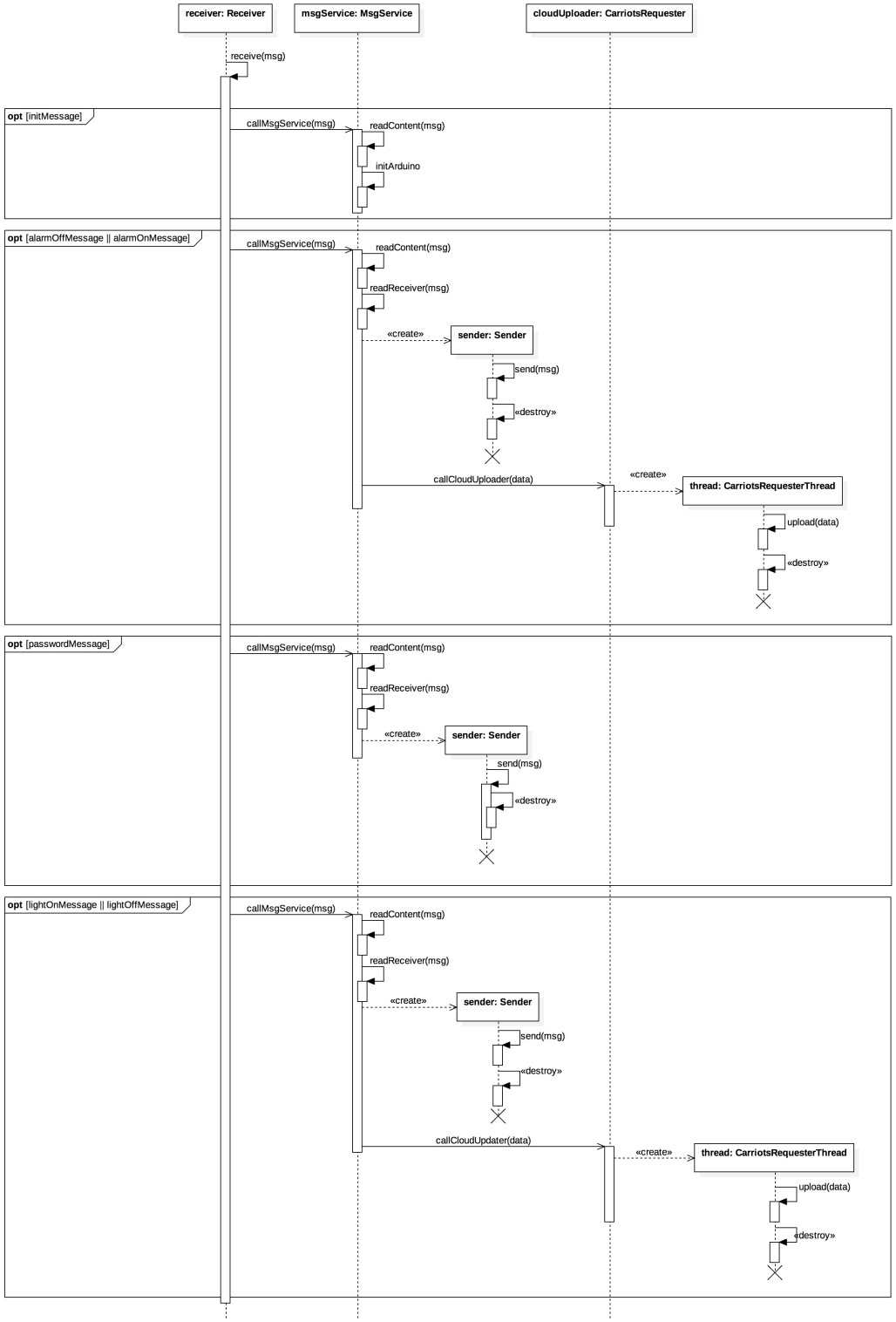

Figura 6.23: Diagramma di sequenza che mostra il comportamento del sistema alla ricezione di un messaggio (Parte 1)

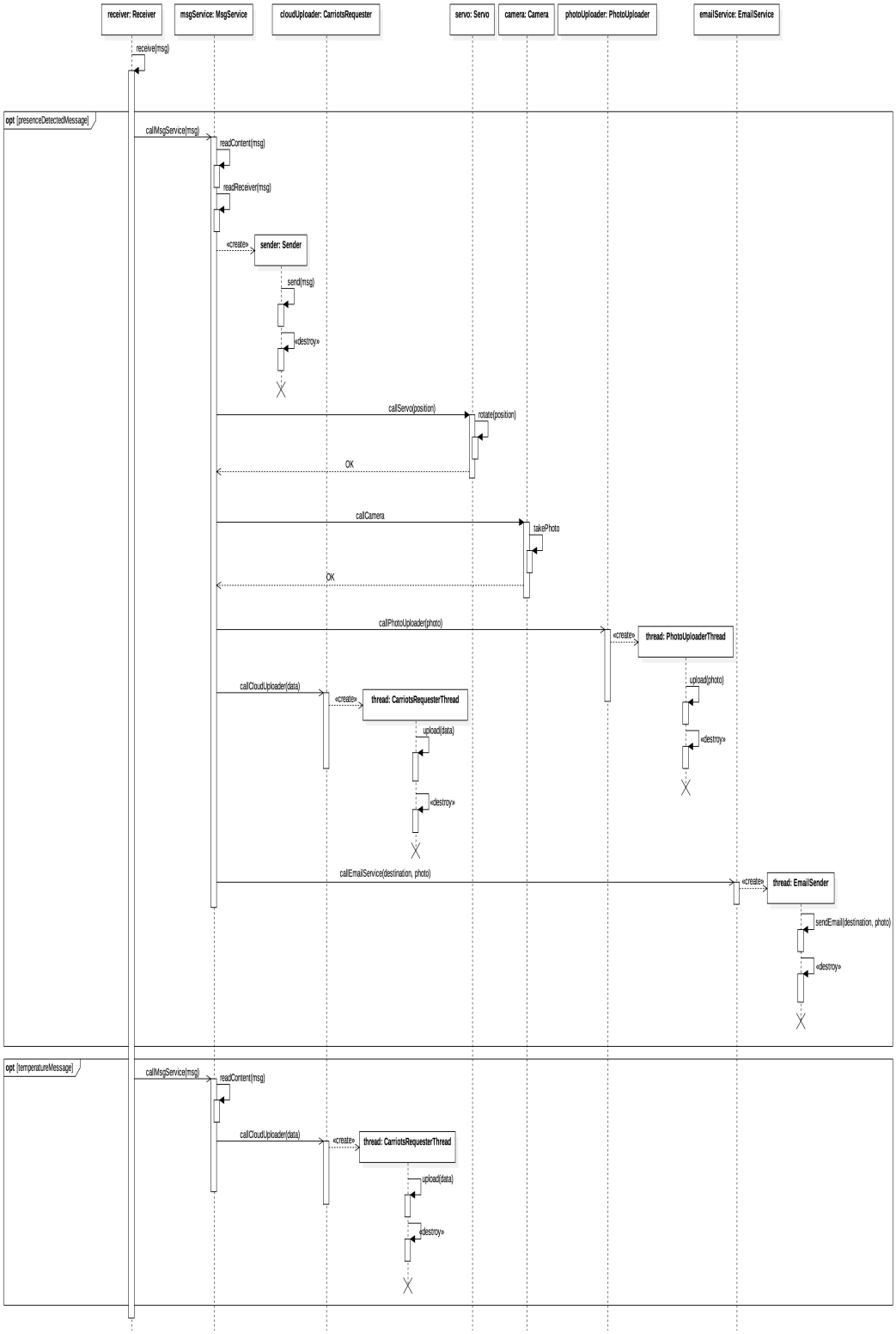

Figura 6.24: Diagramma di sequenza che mostra il comportamento del sistema alla ricezione di un messaggio (Parte 2)

tenuto e effettua il salvataggio di Arduino con la sua corrispondente porta seriale.

- **Ricevuto un messaggio di accensione o spegnimento allarme**: quando il *Receiver* riceve questo messaggio, richiama il *MsgService* che, dopo averne letto il contenuto, effettua la lettura del destinatario e crea un nuovo thread *Sender* che procede all'invio sulla seriale del messaggio. Successivamente viene richiamato *CarriotsRequester* che si occupa del salvataggio nel Cloud dei dati corrispondenti al messaggio appena ricevuto. Esso procede creando un thread *CarriotsRequesterThread* a cui delega il compito dell'upload dei dati aggiornati.

- **Ricevuto un messaggio di accensione o spegnimento luce**: quando il *Receiver* riceve questo messaggio, viene richiamato il *MsgService* che, dopo aver letto il contenuto e il destinatario del messaggio, crea un nuovo thread *Sender* a cui delega il compito dell'inoltro di tale messaggio sulla seriale. Successivamente viene richiamato *CarriotsRequester* che tramite la creazione di un thread *CarriotsRequesterThread* effettua il salvataggio dei dati aggiornati nel Cloud.

- **Ricevuto un messaggio di impostazione password**: alla ricezione di questo messaggio, *Receiver* richiama il *MsgService* il quale, dopo averne letto contenuto e destinatario, procede alla creazione di un nuovo thread *Sender* che procede all'invio di suddetto messaggio.

- **Ricevuto un messaggio di aggiornamento temperatura**: alla ricezione di questo messaggio, *Receiver* richiama il *MsgService* che, dopo averne letto il contenuto, richiama *CarriotsRequester* affinchè aggiorni il valore della temperatura nel Cloud. *CarriotsRequester* procede creando un nuovo thread *CarriotsRequesterThread* che effettua l'aggiornamento richiesto.

- **Ricevuto un messaggio di rilevamento intruso**: alla ricezione di questo messaggio, *Receiver* richiama il *MsgService* che ne analizza

destinatario e contenuto. Dopo aver verificato che si tratta di un rilevamento di un intruso, procede ad avvisare il dispositivo destinatario del messaggio tramite un nuovo thread *Sender*. Successivamente avvisa il *Servo* di posizionare la fotocamera nella posizione corrispondente alla stanza oggetto di rilevamento. Quando il *Servo* termina il posizionamento, avvisa il *MsgService* che provvede a notificare la *Camera* che può procedere allo scatto della fotografia. Quando il *MsgService* riceve la conferma del completamento dello scatto della fotografia, notifica *CarriotsRequester*, *PhotoUploader* e *EmailService* che eseguono tre thread e, rispettivamente, *CarriotsRequesterThread*, *PhotoUploaderThread* e *EmailSender*. Il primo, come già spiegato, procede all'aggiornamento dei dati nel Cloud; il secondo preleva la fotografia scattata e salva quest'ultima in un server Cloud dedicato; il terzo procede alla composizione di un'email, contenente la fotografia scattata, e la invia all'utente.

6.4.4 Apple Watch

Visualizzazione dello stato di allarme

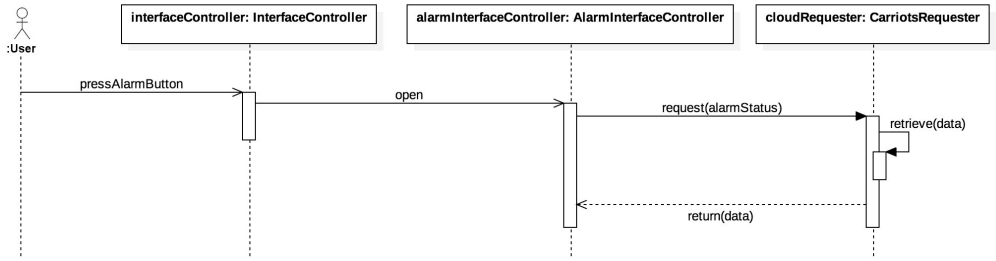

Figura 6.25: Diagramma di sequenza che mostra il funzionamento del sistema durante la visualizzazione dello stato di allarme

Quando l'utente preme il bottone specifico alla visualizzazione dello stato di allarme, il controllore dell'interfaccia principale *InterfaceController* pro-

cede all'apertura di una nuova interfaccia, i cui eventi sono gestiti da *AlarmInterfaceController*.

Questa classe effettua una richiesta al Cloud tramite *CarriotsRequester* il quale, dopo una ricerca del dato specificato, lo restituisce ad *AlarmInterfaceController*.

Visualizzazione della temperatura

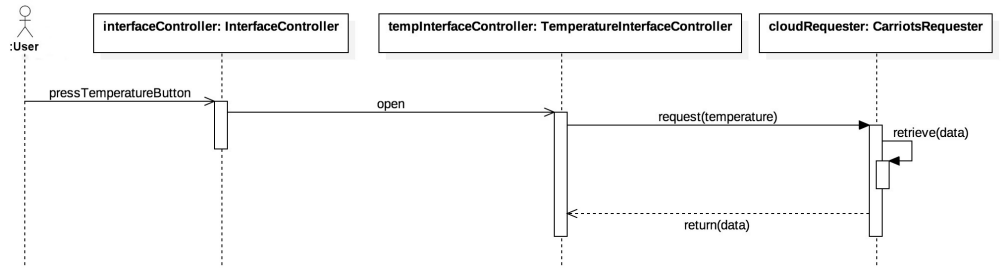

Figura 6.26: Diagramma di sequenza che mostra il funzionamento del sistema durante la visualizzazione della temperatura di casa

Così come per lo stato di allarme, quando l'utente preme il bottone specifico alla visualizzazione della temperatura, *InterfaceController* procede all'apertura dell'interfaccia gestita da *TemperatureInterfaceController*.

Anche questa classe effettua una richiesta al Cloud tramite *CarriotsRequester* il quale le restituisce il dato richiesto.

Visualizzazione informazioni sulla stanza

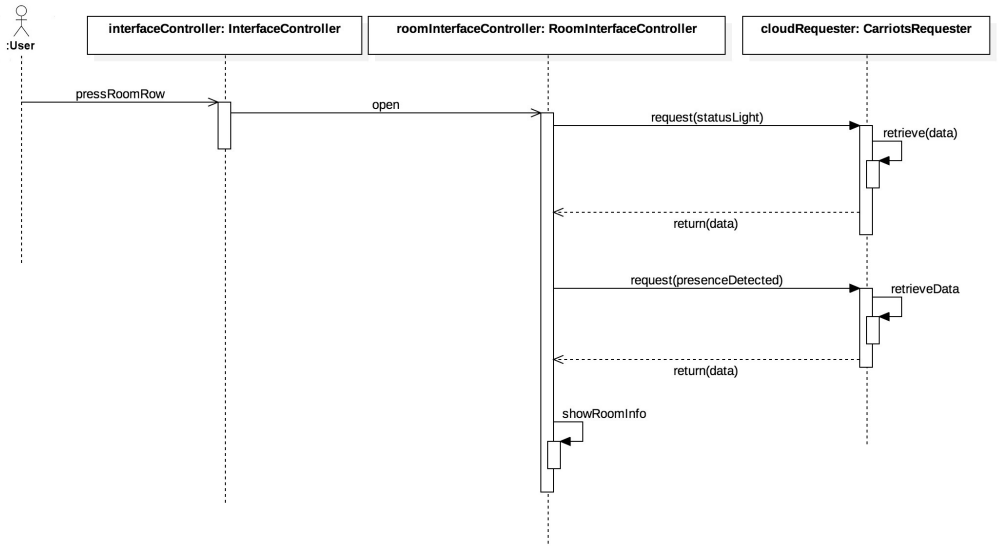

Figura 6.27: Diagramma di sequenza che mostra il funzionamento del sistema durante la visualizzazione delle informazioni sulla stanza richiesta

Anche per questa operazione l'iter è molto simile a quelli visti in precedenza: dopo aver premuto la riga con la stanza desiderata, *InterfaceController* procede all'apertura di *RoomInterfaceController* che effettua due richieste al Cloud: la prima sullo stato della luce (accesa o spenta), la seconda sul rilevamento di un intruso (rilevato o non rilevato). Una volta ricevuto il responso da *CarriotsRequester*, *RoomInterfaceController* mostra i dati richiesti.

Visualizzazione foto dell'intruso

Quando l'utente preme il bottone relativo alla visualizzazione dell'ultima foto scattata, *RoomInterfaceController* effettua una richiesta a Cloudinary, passandogli l'id della stanza. Se Cloudinary non trova una corrispondenza, restituisce un messaggio di errore. Se, invece, trova una corrispondenza con una foto, la restituisce a *RoomInterfaceController* che provvede a visualizzarla.

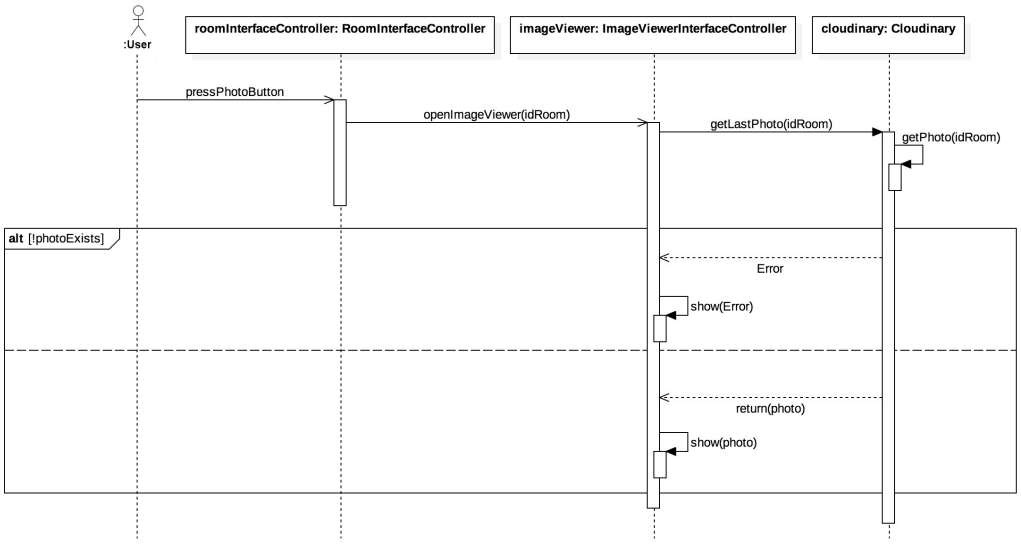

Figura 6.28: Diagramma di sequenza che mostra il funzionamento del sistema durante la visualizzazione dell'ultima foto scattata all'intruso

Attivazione e disattivazione del sistema di allarme

Quando l'allarme è disattivato, dopo che il relativo bottone è stato premuto dall'utente, *AlarmInterfaceController* apre *InsertPasswordInterfaceController* dove l'utente è chiamato ad inserire una password per l'attivazione del sistema di allarme.

Una volta inserita, *InsertPasswordInterfaceController* effettua una richiesta di attivazione allarme a Raspberry Pi e attende l'aggiornamento dei dati. Una volta scaricati i dati aggiornati, li mostra nell'interfaccia.

Se invece l'allarme è attivo, dopo la pressione del bottone viene subito inviata una richiesta di disattivazione allarme a Raspberry Pi. *AlarmInterfaceController* effettua poi una chiamata di richiesta dei dati aggiornati al Cloud tramite *CarriotsRequester*.

Una volta ottenuti i dati aggiornati, li mostra nell'interfaccia.

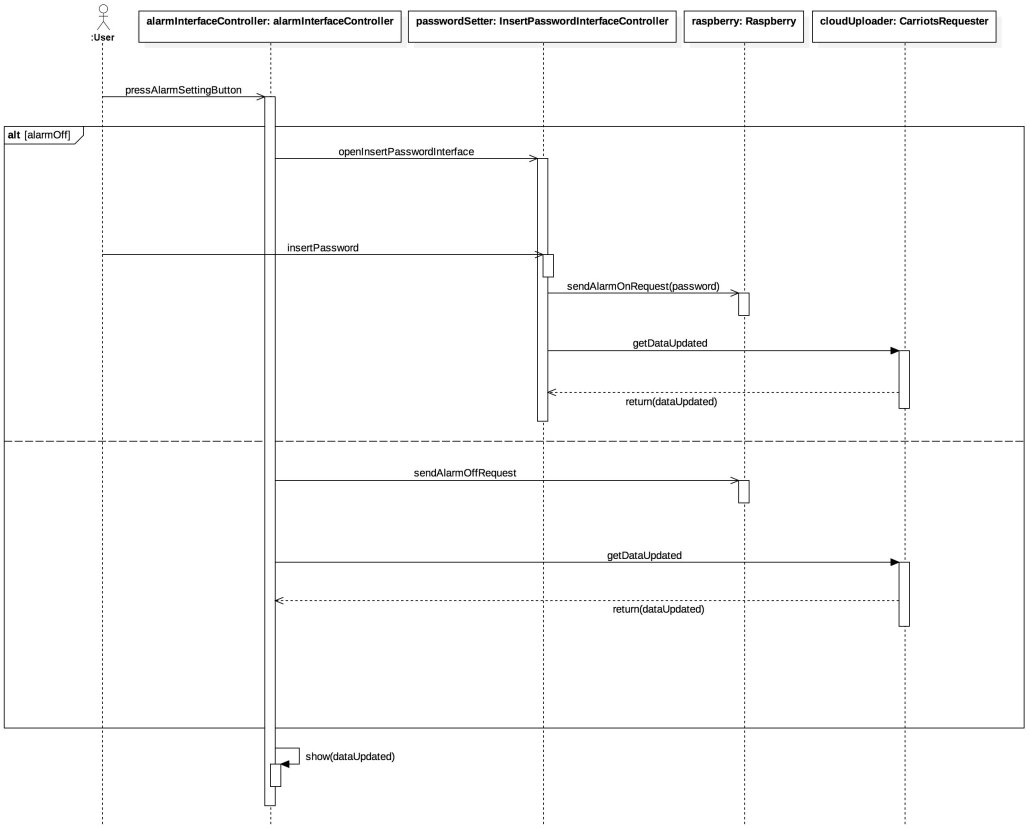

Figura 6.29: Diagramma di sequenza che mostra il funzionamento del sistema durante l'attivazione o la disattivazione dell'allarme da Apple Watch

Accensione e spegnimento luci di una stanza

Quando l'utente preme il bottone per l'accensione o spegnimento di una luce, *RoomInterfaceController* manda una richiesta a Raspberry Pi: se la luce era accesa, manda una richiesta di spegnimento, altrimenti il contrario.

Successivamente *RoomInterfaceController* attende i dati aggiornati facendo una richiesta al Cloud tramite *CarriotsRequester* che provvede a inviarglieli.

Una volta ricevuti i dati, *RoomInterfaceController* li mostra nell'interfaccia.

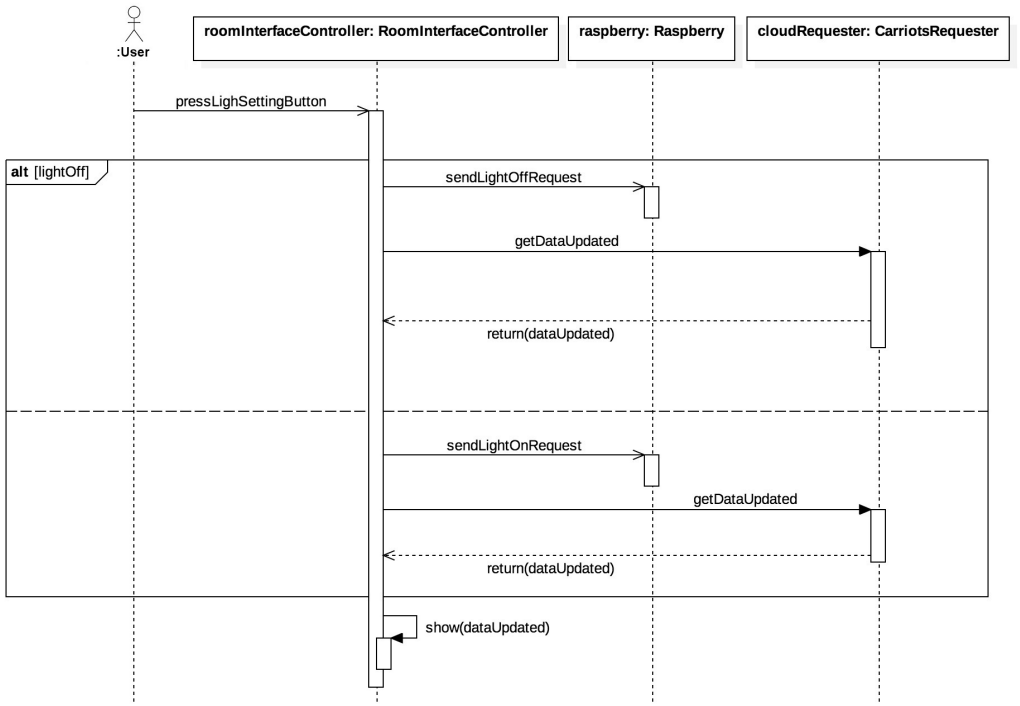

Figura 6.30: Diagramma di sequenza che mostra il funzionamento del sistema durante l'accensione o spegnimento della luce di una stanza

6.5 Implementazione

Procediamo alla fase di implementazione dei dispositivi che compongono il sistema, mostrando le principali scelte implementative ed estratti di codice del sistema.

6.5.1 Arduino 1

Inserimento e verifica password

L'inserimento della password avviene con l'inserimento sequenziale di più caratteri, disponibili nel keypad.

Se viene premuto un tasto nel keypad, il carattere corrispondente sarà inserito nella password. Per completare l'inserimento, occorre premere il

carattere "#" che viene interpretato dal sistema come carattere di fine password.

Ad ogni carattere inserito, viene richiesto al display Lcd di aggiornarsi, impostando la variabile condivisa *updateLcd*.

```
bool AlarmSetterTask::isPasswordInserted()
{
  Password* pass = state == ALARM_OFF ? context->
     getPassword() : context->getPassToCompare();
  char character = this->keyp->insertKey();
  if (character != NULL && character != '#')
  {
    pass->concatCharToValue(character);
    context->setLcdToUpdate();
  }
  else if (character == '#')
  {
     return true;
  }
  return false;
}
```

Listing 6.1: Estratto di codice di Arduino 1 adibito all'inserimento della password

La verifica di una password viene utilizzata per la disattivazione del sistema di allarme.

Inserita la password, il metodo la confronta con una precedente memorizzata e, se è identica, restituisce *true*, altrimenti imposta le variabili condivise *passwordWrong* e *updateLcd* – in modo tale che il display mostri che il codice è errato – e restituisce *false*.

```
bool AlarmSetterTask::isPasswordCorrected()
{
  if (this->isPasswordInserted())
  {
    if ((context->getPassToCompare()->getValue()).
        equals(context->getPassword()->getValue()))
    {
      return true;
    }
    else
    {
      context->setPasswordWrong(true);
      context->setLcdToUpdate();
    }
  }
  return false;
}
```

Listing 6.2: Estratto di codice di Arduino 1 adibito alla verifica della password

Cambiamento di stato del sistema di allarme

Il cambiamento di stato del sistema di allarme avviene controllando il valore che assume la variabile *alarmOn* durante l'esecuzione del sistema e controllando se una password è inserita dall'utente.

Nel primo caso, l'utente ha inviato una notifica remota per l'attivazione dell'allarme; nel secondo caso, l'utente ha premuto il carattere di fine password.

In presenza di questi due casi, il sistema transita nello stato *ALARM_ON*. In questo stato viene verificato se l'utente disattiva il sistema di allarme,

tramite il metodo *isPasswordCorrected()*, che verifica se la password inserita è corretta e tramite il metodo *isAlarmOn()*, che verifica se la variabile ha assunto il valore *false* durante l'esecuzione del sistema.

In presenza di queste due condizioni, viene impostata la variabile condivisa per l'aggiornamento del display, vengono cancellate le password memorizzate e si transita nuovamente allo stato *ALARM_OFF*.

```
void AlarmSetterTask::tick()
{
  switch(state)
  {
    case ALARM_OFF:
      if (context->isAlarmOn())
      {
        this->state = ALARM_ON;
        context->setLcdToUpdate();
      }

      if (this->isPasswordInserted())
      {
        this->state = ALARM_ON;
        context->setAlarmOn();
        context->setLcdToUpdate();
      }
      break;

    case ALARM_ON:
      if (this->isPasswordCorrected() || !context->
          isAlarmOn())
      {
        this->state = ALARM_OFF;
        context->setAlarmOff();
```

```
            context->setLcdToUpdate();
            context->getPassword()->eraseValue();
            context->getPassToCompare()->eraseValue();
        }
    }
}
```

Listing 6.3: Estratto di codice di Arduino 1 adibito al controllo dei cambiamenti di stato nel sistema di allarme

La classe MessageService e il parsing dei messaggi

```
MessageService::MessageService()
{
  this->msgQueue = new QueueList<Message*>();
}

bool MessageService::isMsgAvailable()
{
  return !(this->msgQueue->isEmpty());
}

Message* MessageService::receiveMsg()
{
  if (!msgQueue->isEmpty())
  {
    return this->msgQueue->pop();
  }
  else
  {
    return NULL;
  }
```

```
}

void MessageService::sendMsg(Message* msg)
{
  String message = Parser::getInstance().
     parseMsgToString(msg);
  Serial.println(message);
}

void MessageService::checkSerialForMessage()
{
  String msgFromSerial = "";

  while (Serial.available() > 0)
  {
    char charFromSerial = (char) Serial.read();
    msgFromSerial = msgFromSerial + charFromSerial;
    delay(1);
  }

  if (msgFromSerial != "")
  {
    this->msgQueue->push(Parser::getInstance().
       parseStringToMsg(msgFromSerial));
  }
}
```

Listing 6.4: Codice della classe MessageService

La classe *MessageService* contiene i metodi per la ricezione e l'invio di messaggi nella seriale.

Per la memorizzazione temporanea dei messaggi viene utilizzata una coda

(definita dalla libreria *QueueList*) che permette le tipiche operazioni di push e di pop.

Possono essere eseguite quattro operazioni:

- **isMsgAvailable()**: questo metodo verifica la presenza di messaggi, controllando se la coda che li memorizza è vuota o meno. Restituisce *true* in caso di messaggi disponibili, *false* viceversa.
- **receiveMsg()**: se la coda non è vuota, questo metodo restituisce il primo messaggio disponibile; viceversa, restituisce *null*.
- **sendMsg(Message* msg)**: preso in ingresso un messaggio, questo metodo si occupa di scriverlo sulla seriale sotto forma di stringa.
- **checkSerialForMessage()**: questo metodo verifica se la seriale contiene un eventuale messaggio e, se presente, procede alla memorizzazione di questo nella coda.

Possiamo notare dal codice che, per il parsing dei messaggi, viene utilizzata la classe *Parser*. Questa classe contiene i metodi per convertire un messaggio in una stringa e viceversa e per estrapolarne il contenuto.

```
String parseMsgToString(Message* msg)
{
    String sender = msg->getSender();
    String receiver = msg->getReceiver();
    String content = msg->getContent();

    String message = sender + ';' + receiver + ';' +
       content;

    return message;
}
```

Listing 6.5: Estratto di codice del Parser

Per la conversione da messaggio a stringa, si prelevano da esso il mittente, il destinatario e il contenuto tramite gli opportuni *getter* e si concatenano in una stringa, separati da un punto e virgola.

Per la conversione da stringa a messaggio, si effettua uno *split* sulla stringa di interesse separando le sottostringhe delimitate dal punto e virgola e, infine, si restituisce un messaggio contenente come mittente, destinatario e contenuto le tre sottostringhe ottenute.

Per l'estrapolazione del contenuto in una stringa, si effettua uno *split* sulla stringa di interesse separando le sottostringhe delimitate dal punto e virgola e, infine, si restituisce la seconda sottostringa ottenuta.

Invio di messaggi a fronte di eventi

```
void MessageSenderTask::tick()
{
  switch(state)
  {
    case WAIT_FOR_ALARM_STATE_CHANGES:
      if (actualAlarmState != context->isAlarmOn())
      {
        state = ALARM_STATE_CHANGED;
        actualAlarmState = context->isAlarmOn();
      }
      break;

    case ALARM_STATE_CHANGED:
      Message* alarmMessage;
      if (context->isAlarmOn())
      {
        alarmMessage = new Message(``ARDUINO1'',``
            ARDUINO2'', ``ALARM_ON'');
        this->msgService->sendMsg(alarmMessage);
```

```
        }
        else
        {
          alarmMessage = new Message(``ARDUINO1'', ``
              ARDUINO2'', ``ALARM_OFF'');
          this->msgService->sendMsg(alarmMessage);
        }
        delete alarmMessage;
        this->state = WAIT_FOR_ALARM_STATE_CHANGES;
        break;
    }
}
```

Listing 6.6: Estratto di codice che mostra l'invio di un messaggio a seguito del cambiamento dello stato di allarme

Il *MessageSenderTask* è un task in esecuzione su Arduino 1 che si occupa dell'invio di messaggi sulla seriale a fronte di un evento.

In particolare, l'evento che ascolta è il cambiamento dello stato di allarme. Infatti, nello stato iniziale *WAIT_FOR_ALARM_STATE_CHANGES*, *MessageSenderTask* mantiene una variabile con lo stato attuale e lo confronta con la variabile condivisa *alarmOn*.

Se il valore della variabile della classe è diversa da quello della variabile condivisa, il sistema transita nello stato *ALARM_STATE_CHANGED*. In questo stato, in base al valore di *alarmOn*, procede all'invio di un messaggio di attivazione o disattivazione allarme.

Terminato l'invio, ritorna nello stato iniziale, in attesa di nuovi cambiamenti di valore della variabile condivisa *alarmOn*.

Inizializzazione ed esecuzione dei task

Il main del programma in esecuzione su Arduino 1 è costituito da un metodo *setup()* dove avviene l'inizializzazione dei diversi task e un metodo

loop() dove questi vengono mandati in esecuzione.

```
void setup()
{
  Serial.begin(BAUD_RATE);
  sched.init(SCHED_PERIOD);
  Context* context = new Context();

  MessageSenderTask* msgSender = new MessageSenderTask(
      context);
  msgSender->init();
  sched.addTask(msgSender);

  LcdUpdaterTask* lcdUpdater = new LcdUpdaterTask(10,
      context);
  lcdUpdater->init();
  sched.addTask(lcdUpdater);

  AlarmSetterTask* alarmSetter = new AlarmSetterTask(
      rowKeypadPins, colKeypadPins, context);
  alarmSetter->init();
  sched.addTask(alarmSetter);

  MessageReceiverTask* msgReceiver = new
      MessageReceiverTask(context);
  msgReceiver->init();
  sched.addTask(msgReceiver);

  DetectPresenceTask* detectPresence = new
      DetectPresenceTask(context);
  detectPresence->init();
  sched.addTask(detectPresence);
```

```
}

void loop()
{
  sched.schedule();
}
```

Listing 6.7: Metodi setup() e loop() di Arduino 1

I task vengono mandati in esecuzione da una classe che ha il ruolo di Scheduler. Essa viene impostata con un periodo *SCHED_PERIOD*, che definisce la cadenza con il quale vengono eseguiti i metodi *tick* dei vari task.

```
void Scheduler::schedule()
{
  timer.waitForNextTick();
  for (int i = 0; i < nTasks; i++)
  {
    taskList[i]->tick();
  }
}
```

Listing 6.8: Estratto di codice della classe *Scheduler* che manda in esecuzione i task in essa memorizzati

6.5.2 Arduino 2

Rilevamento di un intruso

Il rilevamento di un intruso avviene grazie ai sensori infrarossi installati nelle varie stanze della casa e avviene solo se il sistema di allarme è attivato.

```
void DetectPresenceTask::tick()
{
  switch(state)
  {
    case WAITING_FOR_PRESENCE:

      if (!context->isAlarmOn())
      {
        buzzer->stopBeep();
      }

      if (context->isPresenceDetected())
      {
        context->setPresenceDetected(false);
      }

      if(context->isAlarmOn())
      {
        for (int i = 0; i < PIR_NUMBER; i++)
        {
          pirTimers[i]++;
          if (pir[i]->checkPresence() && pirTimers[i] >
              PIR_TIMER)
          {
            context->setCodeRoomDetected(pir[i]->
                getLocation());
```

```
                pirTimers[i] = 0;
                state = PRESENCE_DETECTED;
            }
          }
        }
        break;

      case PRESENCE_DETECTED:
        context->setPresenceDetected(true);
        buzzer->toneBeep(440);
        state = WAITING_FOR_PRESENCE;
        break;
    }
}
```

Listing 6.9: Estratto di codice che mostra il rilevamento di un intruso nel sistema in esecuzione su Arduino 2

Il primo stato in cui il sistema si trova è *WAITING_FOR_PRESENCE* che verifica se un sensore a infrarosso ha rilevato una presenza o meno. In caso affermativo, viene impostata la variabile condivisa con l'id della stanza e il sistema transita nello stato *PRESENCE_DETECTED*.

Affinché non vi siano multiple rilevazioni contemporanee di una stessa stanza, ad ogni sensore infrarosso è associato un timer che viene azzerato ogni volta che quel determinato sensore rileva un movimento. In questo modo, eventuali notifiche di quel PIR vengono ignorate se il suo timer non supera la soglia definita in precedenza dalla variabile *PIR_TIMER*.

Nello stato *PRESENCE_DETECTED* viene impostata a *true* la variabile condivisa *presenceDetected* per notificare al *MessageSenderTask* di procedere alla notifica di Raspberry Pi e Arduino 1.

Inoltre viene emanato un segnalazione acustica continua tramite il *buzzer*.

Successivamente, il sistema transita automaticamente allo stato *WAITING_FOR_PRESENCE* nel quale il buzzer terminerà la segnalazione solo

se il sistema di allarme verrà disattivato. Inoltre, finché il sistema di allarme rimarrà attivo, *DetectPresenceTask* rimarrà in ascolto di eventuali altri rilevamenti all'interno dell'abitazione.

Accensione e spegnimento di luci

Come abbiamo visto, i led sono collegati tramite shift register ad Arduino 2. Questo permette di controllare un gran numero di led, utilizzando un numero ridotto di pin del microcontrollore.

La classe che ci permette di gestire la scrittura su shift register è *ShiftOutManager*.

```
void ShiftOutManager::writeShiftOut(int data)
{
  digitalWrite(latchPin, LOW);

  shiftOut(dataPin, clockPin, MSBFIRST, (data >> 8));

  shiftOut(dataPin, clockPin, MSBFIRST, data);

  digitalWrite(latchPin, HIGH);
}
```

Listing 6.10: Estratto di codice che mostra la scrittura di un dato su shift register

Il metodo *writeShiftOut(int data)* di questa classe permette la scrittura sui due shift register installati. Per procedere alla scrittura occorre impostare il pin di Latch a LOW, mentre per chiudere la scrittura occorre impostare tale pin a HIGH.

Il dato che viene scritto è un intero a 16 bit, che rappresenta le luci che devono essere illuminate. Ci sono due luci per ogni stanza, più tre luci che

indicano lo stato dell'allarme. Il dato che viene inviato per essere scritto sugli shift register può essere il risultato di operazioni logiche tra i seguenti valori:

```
enum LedCode {
  ALL_OFF = 0x0000,
  FRONT_ON = 0x8000,
  FRONT_ALARM_ON = 0xC000,
  FRONT_DETECTED = 0xE000,
  ROOM0_LED = 0x0003,
  ROOM1_LED = 0x000C,
  ROOM2_LED = 0x0030,
  ROOM3_LED = 0x00C0,
  ROOM4_LED = 0x0300
};
```

Listing 6.11: Estratto di codice che mostra i valori associati ai led o gruppi di led

La classe *TurnLighOnOffTask* si occupa dell'accensione delle luci in caso di richiesta da remoto o in caso di rilevamento di una presenza.

Nel caso in cui giunga una richiesta da remoto e l'allarme è disattivato, il task legge la variabile condivisa *ligh ToSetOn* nel caso di accensione di una luce o *lightToSetOff* nel caso di spegnimento luce e richiama il metodo *turnLightOn(int roomId)* nel primo caso, *turnLightOff(int roomId)* nel secondo.

Quando invece l'allarme è acceso, non vengono accettate richieste da remoto in quanto *TurnLighOnOffTask* accenderà automaticamente le luci della stanza che sono oggetto di un rilevamento di intruso.

Il metodo *turnLightOn(int roomId)* prende in ingresso l'id della stanza, lo confronta con le stanze disponibili e, se trova una corrispondenza, effettua un OR bit a bit con i led attualmente accesi. Richiama in seguito il metodo *writeShiftOut(int data)* della classe *ShiftOutManager* per procedere alla

scrittura su shift register e, quindi, all'accensione dei led.

```
void TurnLightOnOffTask::turnLightOn(int roomId)
{
    switch(roomId)
    {
      case ROOM0_ANGLE:
        activeLeds = activeLeds | ROOM0_LED;
        ledManager->writeShiftOut(activeLeds);
        break;
      case ROOM1_ANGLE:
        activeLeds = activeLeds | ROOM1_LED;
        ledManager->writeShiftOut(activeLeds);
        break;
      case ROOM2_ANGLE:
        activeLeds = activeLeds | ROOM2_LED;
        ledManager->writeShiftOut(activeLeds);
        break;
      case ROOM3_ANGLE:
        activeLeds = activeLeds | ROOM3_LED;
        ledManager->writeShiftOut(activeLeds);
        break;
      case ROOM4_ANGLE:
        activeLeds = activeLeds | ROOM4_LED;
        ledManager->writeShiftOut(activeLeds);
        break;
    }
}
```

Listing 6.12: Estratto di codice che mostra il metodo per l'accensione dei led di una stanza

Lo spegnimento delle luci di una stanza è molto simile all'accensione: l'unica differenza è che viene effettuato uno XOR bit a bit (anziché OR), in modo da azzerare i bit corrispondenti alle luci che si intende spegnere.

```
void TurnLightOnOffTask::turnLightOff(int roomId)
{
  switch(roomId)
    {
      case ROOM0_ANGLE:
        activeLeds = activeLeds ^ ROOM0_LED;
        ledManager->writeShiftOut(activeLeds);
        break;
      case ROOM1_ANGLE:
        activeLeds = activeLeds ^ ROOM1_LED;
        ledManager->writeShiftOut(activeLeds);
        break;
      case ROOM2_ANGLE:
        activeLeds = activeLeds ^ ROOM2_LED;
        ledManager->writeShiftOut(activeLeds);
        break;
      case ROOM3_ANGLE:
        activeLeds = activeLeds ^ ROOM3_LED;
        ledManager->writeShiftOut(activeLeds);
        break;
      case ROOM4_ANGLE:
        activeLeds = activeLeds ^ ROOM4_LED;
        ledManager->writeShiftOut(activeLeds);
        break;
    }
}
```

Listing 6.13: Estratto di codice che mostra il metodo per lo spegnimento dei led di una stanza

Calcolo della temperatura

```
void CalculateTemperatureTask::tick()
{
  if (this->numOfSurvey == MAX_NUM_SURVEY)
  {
    context->setTemperature(sumTemperatures /
        numOfSurvey);
    this->sumTemperatures = 0;
    this->numOfSurvey = 0;
  }
  else
  {
    this->sumTemperatures += tempSensor->getTemperature
        ();
    this->numOfSurvey++;
  }
}
```

Listing 6.14: Estratto di codice che mostra il calcolo della temperatura in Arduino 2

CalculateTemperatureTask si occupa del calcolo della temperatura dell casa. Per effettuare il calcolo, il task preleva dal sensore il valore un numero di volte pari a *MAX_NUM_SURVEY*.

Terminata la raccolta dei dati, effettua una media matematica e imposta con tale valore la variabile condivisa permettendo a *MessageSenderTask* di inviare il valore aggiornato a Raspberry Pi.

Infine azzera i valori precedentemente calcolati e comincia una nuova raccolta di dati dal sensore.

La classe MessageService e il parsing dei messaggi

La classe MessageService ha le stesse caratteristiche e funzionamento spiegati nell'omonima sottosezione del paragrafo 6.5.1, così come il parsing dei messaggi.

Inizializzazione ed esecuzione dei task

Così come in Arduino 1, anche in Arduino 2 abbiamo un metodo *setup()* dove avviene l'inizializzazione dei vari task e un metodo *loop()* dove i task vengono mandati in esecuzione da una classe che ha il ruolo di *Scheduler*.

```
void setup()
{
  Serial.begin(BAUD_RATE);
  analogReference(INTERNAL);

  int pirPins[PIR_NUMBER] = {PIR0_PIN, PIR1_PIN,
    PIR2_PIN, PIR3_PIN, PIR4_PIN};

  sched.init(SCHED_PERIOD);
  Context* context = new Context();

  MessageSenderTask* msgSender = new MessageSenderTask(
    context);
  msgSender->init();
  sched.addTask(msgSender);

  MessageReceiverTask* msgReceiver = new
    MessageReceiverTask(context);
  msgReceiver->init();
  sched.addTask(msgReceiver);
```

```
  DetectPresenceTask* detectPresence = new
      DetectPresenceTask(context, pirPins, BUZZER_PIN);
  detectPresence->init();
  sched.addTask(detectPresence);

  TurnLightOnOffTask* turnLightOnOff = new
      TurnLightOnOffTask(context);
  turnLightOnOff->init();
  sched.addTask(turnLightOnOff);

  CalculateTemperatureTask* tempTask = new
      CalculateTemperatureTask(context);
  tempTask->init();
  sched.addTask(tempTask);
}

void loop()
{
  sched.schedule();
}
```

Listing 6.15: "Estratto di codice che mostra i metodi *setup()* e *loop()* di Arduino 2

6.5.3 Raspberry Pi

Procediamo alla fase di implementazione in Raspberry Pi, mostrando le principali scelte implementative ed estratti di codice del sistema.

Ricezione e invio di messaggi

A differenza dei due Arduino, in Raspberry Pi la ricezione e l'invio di messaggi è delegato a due thread differenti: *Sender* e *Receiver*.

```
public void run() {
  try {
    while (true) {
      String rcvString = channel.receiveMsg();
      final Msg rcvMessage = Parser.getInstance().
          parseStringToMsg(rcvString);
      if (rcvMessage != null && !rcvMessage.isEmpty()) {
          SwingUtilities.invokeLater(new Runnable() {
              @Override
              public void run() {
                  mainFrame.getLogText().append(``Messaggio
                      ricevuto da: '' + rcvMessage.getSender()
                       + '\n');
                  mainFrame.getLogText().append(``Contenuto
                      messaggio: '' + rcvMessage.getContent()
                      + '\n')
                  }
          });
          MsgService.getMsgService().doActionFromMsg(
              rcvMessage, channel);
      }
    }
  } catch (InterruptedException e) {
     e.printStackTrace();
```

```
    }
}
```

Listing 6.16: Estratto di codice del thread *Receiver*

In questo thread il metodo *receiveMsg()* è bloccante e si mette in attesa di messaggi. Dopo aver fatto un parsing della stringa ricevuta, il *Receiver* richiama il *MsgService* affinché analizzi il contenuto di tale messaggio.

Per ogni messaggio che Raspberry Pi intende inviare, invece, crea un nuovo thread *Sender*. Questo thread, ricevuto in input il messaggio da inviare, lo converte, in un primo momento, in una stringa e successivamente procede all'invio nella seriale.

```
public void run() {
    String msgToSend = Parser.getInstance().
       parseMsgToString(message);
    channel.sendMsg(msgToSend);
    SwingUtilities.invokeLater(new Runnable() {
        @Override
        public void run() {
            mainFrame.getLogText().append(``Inviato
                messaggio a: '' + message.getReceiver() + '\n
                '');
            mainFrame.getLogText().append(``Contenuto
                messaggio: '' + message.getContent() + '\n');
        }
    });
}
```

Listing 6.17: Estratto di codice del thread *Sender*

Salvataggio dei dati nella piattaforma Carriots

Una funzione caratteristica del sistema in esecuzione su Raspberry Pi è il salvataggio dello stream dei dati ricevuti da Arduino e Apple Watch nel Cloud.

In particolare, a livello implementativo è stata scelta la piattaforma Carriots in quanto, oltre ad offrire uno spazio per la memorizzazione dei dati, offre la possibilità di creare dei *listener*, ovvero porzioni di codice che vengono eseguiti alla ricezione di certi dati nella piattaforma.

```
public void run() {
  String stream = ''{\''protocol\'':\''v2\'',\''checksum\''
     :\''\'',\'' + ''\''device\'':\'''' + DEVICE_ID + ''\''
     ,\''at\'':\''now\'',\''data\'''' + dataToStream;

  try {
      URL url = new URL(MAIN_URL);
      HttpsURLConnection connection = (HttpsURLConnection)
          url.openConnection();
      connection.setRequestMethod(''POST'');
      connection.setDoOutput(true);
      connection.setRequestProperty(''Content-type'', ''
          application/json'');
      connection.setRequestProperty(''carriots.apikey'',
          API_KEY);

      OutputStreamWriter outputStreamWriter = new
          OutputStreamWriter(connection.getOutputStream());
      outputStreamWriter.write(stream);
      outputStreamWriter.close();

     [...]
     } catch (MalformedURLException e) {
         e.printStackTrace();
```

```
        } catch (IOException e) {
            e.printStackTrace();
        }
}
```

Listing 6.18: Estratto di codice che mostra il funzionamento dell'upload di un dato nella piattaforma Carriots

Il thread inizialmente crea una stringa in formato JSON contenente i dati che Raspberry intende salvare nella piattaforma. Successivamente viene effettuata una richiesta HTTP di tipo POST a Carriots, che contiene l'api key, necessaria per i privilegi di scrittura. Terminata la richiesta, vengono chiusi gli stream.

Salvataggio delle foto nella piattaforma Cloudinary

```
public void run() {
    Cloudinary cloudinary = new Cloudinary(ObjectUtils.
        asMap(``cloud_name'', ``matty***'', ``api_key'',
        ``***'', ``api_secret'', ``***''));
    File fileToUpload = new File(path);
    try {
        cloudinary.uploader().upload(fileToUpload,
            ObjectUtils.asMap(``public_id'', roomId));
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Listing 6.19: Estratto di codice che mostra l'upload di un file nella piattaforma Cloudinary

Il salvataggio delle foto viene effettuato su Cloudinary. È stata scelta questa piattaforma in quanto fornisce un SDK disponibile in diversi linguaggi,

fra cui Java, permettendo di creare dei progetti in cui è possibile l'upload e l'editing delle foto tramite codice.

Nell'esecuzione del sistema, l'upload di una foto avviene quando viene rilevata una presenza non autorizzata. Si crea, quindi, un'istanza di Cloudinary a cui si passano i dati relativi al proprio spazio cloud e, infine, si richiama il metodo *upload* a cui si passa il file corrispondente alla foto da salvare e informazioni relative al file, quale il nome.

Il server

```
<?php
    $host = ``tcp://localhost'';
    $port = 6789;

    $text = urldecode($_GET['text']) . PHP_EOL;

    $errstr = '';
    $errno = '';

    if ( ($fp = fsockopen($host, $port, $errno, $errstr, 3)
        ) === FALSE)
        echo ``$errstr ($errno)'';
    else {
        fwrite($fp, $text);
            while (! feof($fp)) {
              echo fgets($fp, 4096);
            }
        fclose($fp);
    }
?>
```

Listing 6.20: Codice del server PHP in esecuzione su Raspberry Pi

Il server in esecuzione su Raspberry Pi consente a quest'ultimo di ricevere da remoto i comandi di Apple Watch.

Si tratta di richieste di tipo *GET*, in cui il server preleva la stringa corrispondente all'argomento *text*.

Successivamente apre una socket in localhost con il programma Java – anch'esso in esecuzione su Raspberry Pi – e scrive la stringa ricevuta.

Sarà poi compito di Raspberry Pi interpretare il messaggio ricevuto ed eseguire le opportune azioni.

6.5.4 Apple Watch

Procediamo alla fase di implementazione in Apple Watch, mostrando le principali scelte implementative ed estratti di codice del sistema.

Richiesta di un JSON a Carriots

```
func makeHttpGetRequestToCarriots(url: String) -> JSON? {
    var url = NSURL(string: url)
    var request = NSMutableURLRequest(URL: url!)
    request.HTTPMethod = ''GET''
    request.addValue(apikey, forHTTPHeaderField: ''carriots
        .apikey'')
    request.addValue(''application/json'',
        forHTTPHeaderField: ''Content-Type'')

    var json: JSON?
    var response: AutoreleasingUnsafeMutablePointer<
        NSURLResponse?> = nil

    var dataFromCarriots: NSData = NSURLConnection.
        sendSynchronousRequest(request, returningResponse:
        response, error:nil)!
```

```
    json = JSON(data: dataFromCarriots)

    return json
}
```

Listing 6.21: Estratto di codice che mostra il metodo per la richiesta di un JSON alla piattaforma Carriots in Swift

La piattaforma Carriots restituisce su richiesta le informazioni salvate tramite JSON.

Per riceverle, occorre effettuare una richiesta Http *GET* all'url di interesse, impostando l'api key per l'autorizzazione. Il risultato sarò la ricezione di un JSON con i valori richiesti.

Per ottenere l'url a cui fare la richiesta, è stato implementato un metodo che, dato l'id del JSON memorizzato su Carriots, crea l'url ed effettua la richiesta, restituendo il JSON.

```
func getJSONFromId(id: Int) -> JSON? {
    let url = mainUrl + ``?sort=at&order=-1&max=1&data[id
        ]=\(id)&device='' + deviceName
    return makeHttpGetRequestToCarriots(url)
}
```

Listing 6.22: Estratto di codice che mostra il metodo per la generazione dell'Url a cui fare la richiesta Http

Richiesta di una foto a Cloudinary

Come visto in precedenza, Cloudinary offre un SDK per numerosi linguaggi, tra cui Objective-C.

È quindi possibile integrarlo in un progetto Swift, sfruttando tutte le funzionalità di upload, editing e richiesta delle foto tramite codice.

Inizialmente si crea un'istanza di Cloudinary, specificando il link del proprio storage sulla piattaforma (api key e api secret vanno incluse nel link).

Successivamente si crea un'istanza *CLUploader* di Cloudinary e si effettua una richiesta esplicita, specificando l'id della foto.

```
override func awakeWithContext(context: AnyObject?) {
    cloudaryCompletion = false;
    if let id = context as? Int {
        var cloudinary = CLCloudinary(url: cloudinaryLink)
        var uploader = CLUploader(cloudinary, delegate:
            self)
        uploader.explicit(''\(id)'', options: [''type'':''
            upload'']), withCompletion:
            onCloudinaryCompletion, andProgress:
            onCloudinaryProgress)
    }
}
```

Listing 6.23: Estratto di codice che mostra l'invio di una richiesta di foto a Cloudinary in Swift

Infine i metodi *onCloudinaryProgress* e *onCloudinaryCompletion* ci permettono di gestire rispettivamente l'avanzamento della richiesta e quando la richiesta è completata.

Invio di un comando a Raspberry Pi

```
@IBAction func turnLightOnOff() {
    var url: NSURL!

    if lightOn! {
        url = NSURL(string: ''RASP_IP:8080/server.php?text=
            WATCH;ARDUINO2;LIGHT_OFF:\(roomId)'')
    } else {
```

```
        url = NSURL(string: ''RASP_IP:8080/server.php?text=
            WATCH;ARDUINO2;LIGHT_ON:\(roomId)'')
    }

    var request = NSMutableURLRequest(URL: url!)
    request.HTTPMethod = ''GET''

    var response: AutoreleasingUnsafeMutablePointer<
        NSURLResponse? >= nil
    NSURLConnection.sendSynchronousRequest(request,
        returningResponse: response, error:nil)!

    [...]
}
```

Listing 6.24: Estratto di codice che mostra l'invio di una richiesta di accensione o spegnimento luce a Raspberry Pi

L'invio di un comando a Raspberry Pi procede dapprima con la generazione dell'url a cui fare la richiesta, formato da l'ip del Raspberry, la porta, e il comando inviato tramite *GET* alla pagina php.

In seguito si procede con una richiesta Http all'url generato e si attende l'esecuzione da parte di Raspberry Pi.

6.6 Testing

Essendo il sistema composto da diversi dispositivi, sono stati necessari diversi test per verificarne il corretto funzionamento.

I principali test sono stati eseguiti riguardo:

- Lettura corretta di messaggi sulla seriale
- Lettura corretta dei comandi dal server
- Upload corretto dei dati nel Cloud

- Funzionamento corretto dei sensori e attuatori
- Reattività del sistema

6.6.1 Lettura corretta di messaggi sulla seriale

Poiché i vari dispositivi cooperano tramite scambio di messaggi, è necessario che questi vengano letti e interpretati correttamente.

Per quanto riguarda i due Arduino, è stato utilizzato dapprima il monitor seriale di *Arduino IDE*, inviando messaggi di esempio attui a verificare la corretta ricezione e a visualizzare i cambiamenti di stato dei sottosistemi.

Successivamente sono stati collegati a Raspberry Pi, per verificare il corretto svolgimento della comunicazione tra i tre dispositivi. In particolare, nella GUI del software in esecuzione su Raspberry Pi è stata sviluppata un'area di testo di log in cui vengono segnalati i principali eventi, tra cui la ricezione e l'invio di un messaggio.

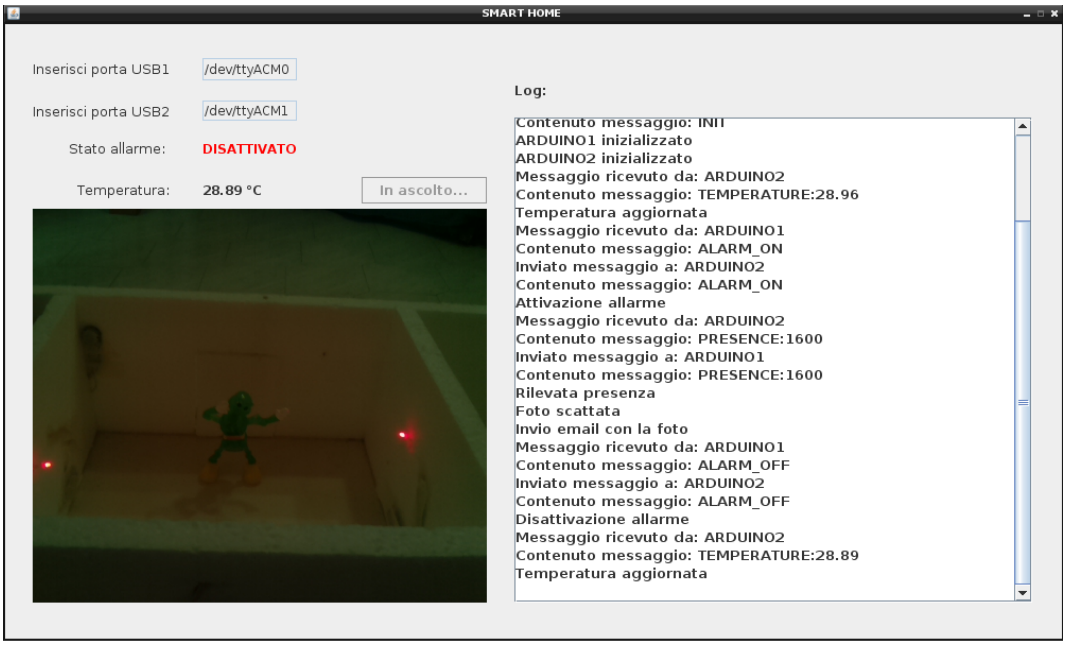

Figura 6.31: Software in esecuzione su Raspberry Pi

È quindi, possibile, verificare se il contenuto del messaggio è giunto correttamente a Raspberry Pi e, in caso di inoltro, se viene inviato correttamente al dispositivo specificato.

6.6.2 Lettura corretta dei comandi dal server

Come già visto nei paragrafi precedenti, in Raspberry Pi è in esecuzione un server al quale è possibile mandare richieste con specifici comandi da eseguire.

Oltre a verificare che le richieste effettuate da Apple Watch venissero ricevute correttamente, sono state effettuate richieste multiple al server, il quale ha risposto a tutte correttamente e senza rallentamenti.

6.6.3 Upload corretto dei dati nel Cloud

Raspberry, a fronte di specifici messaggi ricevuti, deve procedere ad aggiornare le informazioni nel Cloud. Alcuni test sono stati mirati a verificare che questi dati fossero correttamente memorizzati nei server delle due piattaforme scelte: Carriots e Cloudinary.

In entrambe le piattaforme, è possibile accedere ad un portale nel quale visualizzare tutti i dati memorizzati. Il test, quindi, ha sottoposto il sistema a diversi eventi, sottoponendo i sottosistemi a cambiamenti di stato e invio e ricezione di numerosi messaggi.

Raspberry Pi risponde correttamente a tutti i messaggi ricevuti, pubblicando i dati sul Cloud senza errori (tali dati sono perfettamente visibili nel portale delle piattaforme).

6.6.4 Funzionamento corretto dei sensori e attuatori

Ogni sensore e attuatore, prima di essere parte integrante del sistema, è stato oggetto di test indipendenti per conoscerne il corretto funzionamento.

Per esempio, i sensori a infrarossi sono stati inizialmente gestiti singolarmente, notando che hanno una sensibilità molto elevata che ha comportato controlli aggiuntivi a livello di codice.

Terminata la fase di testing dei singoli sensori e attuatori, essi sono stati integrati nel sistema complessivo, dove sono stati oggetto di ulteriori test, in particolare riguardo l'integrazione e l'interazione con i task.

6.6.5 Reattività del sistema

Una caratteristica fondamentale che il sistema deve avere è la reattività, in particolare a fronte di eventi come il rilevamento di una intruso all'interno dell'abitazione.

Sono stati eseguiti, quindi, diversi test attui a impegnare il sistema come carico di lavoro (come ad esempio rilevamenti multipli di intrusi, invio e ricezione multipla di messaggi) ed esso ha mantenuto la fluidità e reattività che si intendeva avere in tale progetto.

6.7 Limiti del sistema

Il progetto realizzato si propone come un prototipo di un sistema di domotica. Benché funzionante, presenta delle limitazioni che è opportuno elencare se, in futuro, il progetto volesse essere utilizzato in una situazione reale.

Una prima considerazione da effettuare è la presenza di una sola fotocamera all'interno del prototipo sviluppato. Volendo implementare il sistema di allarme in una casa reale, occorrerebbe effettuare alcune modifiche in modo tale da poter gestire una fotocamera per ogni stanza.

Un secondo punto degno di nota è la mancanza di criptaggio delle informazioni inviate. Come abbiamo visto, infatti, in Raspberry Pi gira un server web che permette a dispositivi esterni di inviare richieste. L'assenza di cifratura dei comandi inviati dai dispositivi esterni rende il sistema vulnerabile ad attacchi come *man in the middle*, dove le informazioni vengono intercettate o modificate prima che queste giungano a destinazione. È, quindi, opportu-

no integrare dei protocolli crittografici come SSL o TSL, offrendo cifratura, integrità dei dati e autenticazione end-to-end.

Inoltre, affinché il sistema funzioni correttamente è necessario che il server web su Raspberry Pi e le piattaforme cloud siano costantemente online. In un'implementazione reale è, quindi, opportuno considerare l'eventualità che questi possano non essere raggiungibili a fronte di inconvenienti. Nel caso di Raspberry Pi, una causa potrebbe essere la mancanza temporanea di corrente elettrica che renderebbe il sistema offline. Una possibile soluzione è un'alimentazione a batterie del dispositivo, che viene attivata quando non è disponibile la corrente elettrica e permette, quindi, di mantenere online il sistema. A livello software, l'utente potrebbe essere avvisato del passaggio ad un'alimentazione a batterie, così da poter intervenire successivamente nella risoluzione del problema. Per quanto riguarda le piattaforme cloud, queste potrebbero non essere raggiungibili per diversi motivi, quali problemi ai server. In questa situazione, i dispositivi esterni non avrebbero più accesso alle informazioni sulla casa, creando disagi all'utente finale. Una soluzione possibile è un salvataggio temporaneo delle informazioni sul sistema in locale su Raspberry Pi, fino a quando la connettività con le piattaforme non viene ripristinata. I dispositivi, allo stesso tempo, non potendo ricevere da queste le informazioni sulla casa, inoltrano le loro richieste a Raspberry Pi, il quale procede a inviare la risposta. In questo modo l'utente finale è in grado di utilizzare comunque il software a disposizione, anche a fronte di errori non legati alla programmazione.

Infine, a differenza della costruzione di nuova casa, l'implementazione del sistema su una casa preesistente può risultare difficoltosa con una connessione seriale fra i dispositivi Arduino e Raspberry Pi. Una soluzione più efficace sarebbe creare un'interconnessione di dispositivi tramite Wi-Fi, in modo da permettere un'installazione semplificata dei dispositivi nelle stanze.

Conclusioni

Lo sviluppo di questa tesi mi ha permesso di approfondire diversi argomenti, terminando con lo sviluppo di un progetto che li comprendesse tutti: dalla programmazione su dispositivi wearable, come Apple Watch, alla programmazione su dispositivi Arduino e Raspberry Pi fino all'integrazione con il Cloud.

Un concetto chiave e ricorrente di questo percorso è sicuramente *Internet of Things*, di cui la domotica è uno dei principali campi al quale questo nuovo modo di vedere Internet può applicarsi. Nel futuro, sempre più accessori e dispositivi saranno progettati per essere connessi alla rete e l'utente potrà godere dei benefici di questa rivoluzione. Infatti, questa connessione (e interconnessione) dei dispositivi consente non solo di aggiungere funzionalità alla nostra abitazione ma anche di evitare sprechi di cibo e di elettricità. L'utente potrà essere informato dal frigorifero della presenza di alimenti in scadenza e, allo stesso tempo, potrà gestire da remoto i propri elettrodomestici, per esempio azionando la lavatrice nelle ore in cui costa meno. Queste sono solo due delle potenzialità che una casa domotica sarà in grado di offrire a chi vi abita. Non solo: il passo successivo sarà di interconnettere i vari oggetti delle nostre case anche con il mondo esterno, entrando in un'ottica di smart city e smart grid, in cui è possibile leggere e inviare dati utili a una gestione globale.

Il progetto sviluppato si propone di essere un prototipo di un sistema di domotica, sviluppando temi quali il controllo e la sicurezza dell'abitazione. Con aggiornamenti mirati a superare i limiti descritti nel paragrafo 6.7, il

progetto può essere integrato in una vera abitazione, emulando quelli che ad oggi sono i sistemi più diffusi, come ad esempio SmartThings di Samsung.

Come detto in precedenza, il sistema può essere integrato con dei moduli Wi-Fi, per evitare connessioni seriali tra i dispositivi Arduino e Raspberry Pi. Una soluzione possibile è la creazione di una WPAN (Wireless Personal Area Network), utilizzando standard di comunicazione come ZigBee e permettendo di avere una comunicazione tra device a basso consumo e basso costo.

Il sistema, inoltre, ha una struttura modulare e le funzionalità sono facilmente estensibili. Ad esempio, è possibile inserire ulteriori dispositivi Arduino che, tramite opportuni sensori, gestiscano situazioni critiche come incendi, allagamenti e fughe di gas all'interno dell'abitazione, aumentandone la sicurezza.

Infine, poiché è stato introdotto un template comune per lo scambio di messaggi all'interno del sistema, viene semplificato lo sviluppo di un possibile porting dell'applicazione per Apple Watch su altri dispositivi, come per esempio smartphone o smartwatch montanti Android Wear. In questo modo viene aumentata la portabilità del software e si permette all'utente di scegliere il dispositivo che meglio risponda alle sue esigenze.

Bibliografia

[1] Trisciuoglio D., *Introduzione alla domotica*, Milano, Tecniche Nuove, 2009

[2] Mell P., Grance T., *The NIST Definition of Cloud Computing*, in "NIST Special Publication 800-145", 2011

[3] Evans D., *Internet of Things. Tutto cambierà con la prossima era di Internet*, in "White Paper Cisco System", 2011

[4] Leslie Pack Kaelbling, *Learning in Embedded Systems*, MIT Press, 1993

[5] E. A. Lee, S. A. Seshia, *Introduction to Embedded Systems. A Cyber-Physical Systems Approach*, LeeSeshia.org, 2011

[6] Dr. Y. Narasimha Murthy, *UNIT-I. Introduction to Embedded systems*, in "http://www.slideshare.net/yayavaram/unit-1-embedded-systems-and-applications"

[7] Spela Kosir, *Wearables in Healthcare*, in "https://www.wearable-technologies.com/2015/04/wearables-in-healthcare/"

[8] Ashton K., *That 'Internet of Things' Thing*, in "RFID Journal", 2009

[9] Bonato P., *Wearable Sensors/Systems and Their Impact on Biomedical Engineering*, 2003

[10] Atzori L., Iera A., Morabito G., *The Internet Of Things: A Survey*, in "Computer Networks", 2010

[11] Collins T., *Tech Talk: HealthPatch MD Latest in Wearable Health Devices*, in "http://www.the-rheumatologist.org/article/tech-talk-healthpatch-md-latest-in-wearable-health-devices/", 2015

[12] Maxi Sport, *Cos'è e come funziona Nike+?*, in "http://www.maxisport.com/ap/cos-e-come-funziona-nike-2-a.htm"

[13] Microsoft, *Microsoft HoloLens*, in "https://www.microsoft.com/microsoft-hololens/en-us"

[14] WebNews, *Google Glass, tutto sugli occhiali Android*, in "http://www.webnews.it/speciale/google-glass/"

[15] Xiaomi, *Mi Band*, in "http://www.mi.com/in/miband/"

[16] Ari Brockman, *Compare Smart Glasses*, in "http://viewer.tips/smart-glasses/", 2015

[17] Apple, *Apple Watch Programming Guide*, in "https://developer.apple.com/library/ios/documentation/General/Conceptual/WatchKitProgrammingGuide/", 2015

[18] Apple, *HomeKit Developer Guide*, in "https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/HomeKitDeveloperGuide/Introduction/Introduction.html", 2015

[19] Google, *Building Apps for Wearables*, in "https://developer.android.com/training/building-wearables.html"

[20] Samsung, *SmartThings Documentation*, in "http://docs.smartthings.com/en/latest/", 2015

[21] Cloudinary, *Cloudinary Documentation*, in "http://cloudinary.com/documentation/", 2015

[22] Carriots, *Carriots Documentation*, in "https://www.carriots.com/documentation/api", 2011

[23] TempoIQ, *TempoIQ Documentation*, in "https://app.tempoiq.com/docs/html/index.html", 2014

[24] CyberVision, *Kaa Documentation*, in "http://docs.kaaproject.org/display/KAA/Kaa+IoT+Platform+Home", 2014

[25] SimpleThings, *Canopy REST API*, in "http://canopy.link/devzone/restapi/", 2014

[26] Nest Labs, *Architecture Overview*, in "https://developer.nest.com/documentation/cloud/architecture-overview"

[27] GSMA, *GSMA: The impact of the Internet of Things. The Connected Home*, in "http://www.gsma.com/newsroom/wp-content/uploads/15625-Connected-Living-Report.pdf"

[28] Robert E. Hall, *The Vision of A Smart City*, 2nd International Life Extension Technology Workshop, 2000