POLITECNICO DI MILANO

Scuola di Ingegneria Industriale e dell'Informazione

Corso di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

“ Realizzazione del caso di studio *SafeHome* in openHAB e in ambiente virtuale ”

Relatore: Prof. Raffaela MIRANDOLA

Tesi di Laurea di:

Alessandro Verga

matricola 880235

Anno Accademico 2018 / 2019

Abstract (english)

Uncertainty situations in smart homes are often the cause of the alarm system's malfunctioning. The process of acquiring knowledge on the nature and causes of these non deterministic anomalies is not trivial. However, in a virtual environment, when said uncertain situations are correctly modeled, it is possible to observe the behaviour of a software system in response to those events in isolated and controlled conditions.
In this work, the author proposes the development of a software system possessing said characteristics which simulates a smart home and its functioning: some uncertainty situations typically found in reality, ranging from measurement errors to sensors malfunctioning, are carefully injected into this virtual environment, and the system's behaviour is observed in response to such situations.

The smart home itself is a realization as a software of the case study *SafeHome*, a fictitious project invented by *Dr. Roger Pressman, Ph.D.*, in his textbook *Software Engineering: a Practitioner's Approach* [1], where it is used for teaching purposes.
A remote control panel is built using *openHAB* [5], an automation open-source software developed by openHAB Foundation [6] which lets users construct a remote interface from which they can control their smart homes.

Abstract (italian)

Nell'ambito della domotica, situazioni di incertezza legate ai dispositivi installati in una smart home sono spesso la causa di un malfunzionamento del sistema di allarme. L'analisi delle cause e della natura di queste anomalie non deterministiche non è un processo banale. In un ambiente virtuale, però, se queste situazioni incerte vengono modellate correttamente, si può osservare il comportamento di un sistema software in risposta a tali eventi in condizioni isolate e controllate.
In questo elaborato si propone lo sviluppo di un sistema software con tali caratteristiche che simuli il funzionamento di una smart home: in questo ambiente virtuale vengono accuratamente iniettate delle situazioni di incertezza estremamente comuni in natura, come errori di misurazione o malfunzionamenti di sensori, e viene osservato il comportamento del sistema presentato in risposta ad esse.

Per la costruzione della casa intelligente viene preso in esame il caso di studio *SafeHome*, un progetto accademico per una smart home inventato dal *Dr. Roger Pressman, Ph.D.,* utilizzato per scopi didattici nel suo libro di testo *Software Engineering: a Practitioner's Approach* [1], e viene proposta una sua realizzazione come applicativo.
Per la costruzione del pannello di controllo remoto viene invece utilizzato un applicativo dal reale funzionamento, *openHAB* [5], un software di domotica open source che dà la possibilità ai suoi utenti di realizzare un'interfaccia di controllo remoto per la moderazione della propria abitazione intelligente.

Alla mia famiglia, che mi ha sempre supportato e sostenuto;
ai miei amici e colleghi, senza i quali non avrei mai superato questi anni di università.

1. Introduzione 8
1.1 Background e stato dell'arte 10
SafeHome: descrizione del caso di studio 10
openHAB 11
Concetti di openHAB 12
Interfaccia utente di openHAB 13
Stato dell'arte: approcci per la simulazione di una smart home 14
1.2 Soluzione proposta 16
Valutazione 17

2. Analisi del caso di studio 19
2.1 Funzionalità principali 19
2.2 Analisi e specificazione dei requisiti 20
Requisiti non funzionali 21

3. Descrizione della soluzione: il sistema SafeHome 22
3.1 Sensori 22
I tipi di dato supportati, nel dettaglio: 24
3.2 Ambiente 26
La variazione dei dati, nel dettaglio: 27
3.3 La comunicazione interna: PubNub 29
3.4 L'interfaccia remota: openHAB 30
Il binding di openHAB 32

4. Architettura del sistema 34
4.1 Modelli UML 35
Component diagrams 35
State machine diagrams 38
Activity diagrams 38
Sequence diagrams 40

5. Design del sistema 44
5.1 Analisi delle UI 44
UI Sensori 45
UI Ambiente 46
UI openHAB 48
5.2 API utilizzate 48

6. Come utilizzare SafeHome 50
6.1 Modificare la configurazione della smart home 50
Aggiungere nuovi dispositivi, sensori e stanze 51
Modificare openHAB 53

7. Incertezza del sistema 57
7.1 Contesto standard di esecuzione 58
7.2 Scenari di test 63
Fallimento di un sensore (Scenario S1) 64
Errore nella misurazione (Scenario 2) 68

Ritardo di openHAB (Scenario 3) 70

8. Conclusioni 733

9. Lavoro futuro 744

Appendice A: Ambiente hardware e software utilizzati 75

10. Bibliografia e sitografia 76

1. Introduzione

Un *ambiente virtuale*, per definizione, è una riproduzione di un luogo o situazione della realtà nella quale sono stati eliminati alcuni dettagli considerati trascurabili: è stata cioè compiuta una *approssimazione*, o per usare un termine molto più caro all'informatica, una *astrazione*.
Il modello risultante da questo processo di ricostruzione omette dei particolari della realtà, ma non sempre questa mancanza è per forza di cose un difetto: in tutti i campi ingegneristici si produce una approssimazione di un fenomeno reale tralasciando delle inezie considerate più o meno rilevanti in nome della semplificazione teorica del modello stesso; l'*ingegneria del software* - la disciplina che si occupa dello studio dei processi che caratterizzano lo sviluppo di un applicativo - non è da meno.
Esiste infatti un elevato numero di situazioni definite **incerte** nelle quali il sistema informatico che l'ingegnere del software produce o analizza assume un comportamento **non deterministico**, ovvero che non si conosce a priori del verificarsi della situazione d'incertezza stessa.
Le indagini necessarie per la comprensione di queste anomalie, spesso focalizzate sulla natura e sulle cause del fenomeno, non sono affatto banali, perché i motivi potrebbero essere molteplici e tra loro correlati e quindi complicare ulteriormente il processo di analisi.

Un ambiente virtuale, però, proprio a causa dell'elevato grado di astrazione introdotto nella modellazione può permettersi di fornire un contesto *isolato* e *controllato* nel quale osservare il comportamento della simulazione al verificarsi di tali eventi incerti - i quali, visto che di solito non sono presenti naturalmente nella simulazione stessa, devono prima essere ricostruiti ed accuratamente inseriti al suo interno.

Nell'ambito della **domotica** le situazioni di incertezza sono spesso la diretta causa di un malfunzionamento del sistema di allarme installato nella casa intelligente, e che il proprietario nonchè utente finale del software di controllo ha tutto l'interesse di mantenere perfettamente deterministico. Occasionalmente accade infatti che un rilevatore di attività misuri erroneamente un movimento in una stanza e faccia scattare l'antifurto installato nell'abitazione, o che il sensore di fumo a causa di un qualche malfunzionamento si accorga di un inesistente incendio - o, decisamente peggio, che essi non attivino il sistema di allarme quando realmente ce n'è bisogno.

In questo elaborato si propone lo sviluppo di un sistema software intrinsecamente isolato e controllato che permetta una simulazione di una smart home e del suo funzionamento: in questo ambiente virtuale vengono attentamente iniettate delle situazioni di incertezza estremamente comuni in natura, come errori di misurazione o malfunzionamenti dei sensori installati, e si osserva il comportamento del sistema presentato in risposta ad esse. La proprietà che viene osservata in tutti questi casi è sempre *il numero di attivazioni del sistema di allarme*.

Per la costruzione della casa intelligente viene preso in esame il caso di studio ***SafeHome***, un progetto accademico per una smart home proposto dal *Dr. Roger Pressman, Ph.D.,* utilizzato per scopi didattici nel suo libro di testo *Software Engineering: a Practitioner's Approach* [1], e viene proposta una sua realizzazione come applicativo.
Per la costruzione del pannello di controllo remoto viene invece utilizzato un applicativo dal reale funzionamento, ***openHAB*** [5], un software di domotica *open source* sviluppato dalla *openHAB Foundation* [6] per permettere la comunicazione tra un ampio numero di dispositivi intelligenti e dare la possibilità ai suoi utenti di realizzare un'interfaccia di controllo remoto per la moderazione della propria abitazione smart.

La soluzione proposta è di interesse sia a livello didattico, perché esattamente come il caso di studio preso in esame, segue a livello pratico i processi dell'ingegneria del software caratteristici dello sviluppo di un applicativo, sia come ambiente virtuale isolato e controllato all'interno del quale sono stati iniettati modelli di fonti di incertezza tipiche della realtà e si è osservato il comportamento in loro risposta.
Il pannello utilizzato per il test di tali situazioni non deterministiche è inoltre utilizzabile da chiunque sia interessato ad una simulazione di smart home, per effettuare i propri esperimenti riguardo gli eventi incerti proposti con questo elaborato o per la futura costruzione di altri.

Il software prodotto è sviluppato come *open source*, e pertanto è reso accessibile nella sua interezza a chiunque sia interessato alla sua esecuzione, al suo funzionamento interno o alla sua futura modifica.

L'applicativo è reperibile su *GitHub* [60] al seguente link: *https://github.com/aleverga10/SafeHome* .

1.1 Background e stato dell'arte

SafeHome: descrizione del caso di studio

SafeHome è un progetto accademico per una smart home inventato dal *Dr. Roger Pressman, Ph.D.,* nel suo libro *Software Engineering: a Practitioner's Approach* [1].
L'autore, noto esperto di *software engineering* a livello internazionale, utilizza per tutta la durata del libro il progetto *SafeHome* a scopi didattici per illustrare i principi e i processi caratteristici della materia, e il caso di studio infatti è ben noto in letteratura nel campo dell'ingegneria del software: Pressman passa dalla spiegazione teorica di processi quali analisi dei requisiti, design dell'architettura, modellazione e strutturazione delle interfacce utente, ad un approccio più pratico utilizzando proprio a titolo esemplificativo il progetto di una smart home.
L'intero ciclo di vita di un software viene così illustrato grazie al caso di studio tramite scenari a cui l'autore immagina di assistere in terza persona, quasi sempre comprendenti conversazioni di stampo più o meno tecnico tra vari membri della fittizia azienda, tra i quali sviluppatori, designer, manager e collaboratori terzi intervistati per questo o per quel motivo.

Il libro di testo, che da oltre quarant'anni è considerato tra i maggiori e più importanti scritti nel campo dell'ingegneria del software [2] nonché la più grande e comprensiva guida pratica alla materia [3], è utilizzato come tomo consigliato nei corsi di studio di varie università in tutto il mondo [7] [8] [9][10] e *SafeHome* è dettagliatamente analizzato in ciascun programma didattico.
In nessuno di questi programmi didattici, però, sono presentate modifiche al design originale di SafeHome, mantenendo un punto di vista più astratto e meno dipendente da una realizzazione fisica della smart home - che avrebbe invece bisogno di alcuni chiarimenti su punti lasciati volutamente vaghi dall'autore.

L'idea del progetto, nell'esempio del Dr. Pressman, parte infatti da alcuni dipendenti dell'azienda di home security CPI Corporation che inventano un dispositivo wireless, il quale installato su un qualsiasi sensore ne permette l'accesso all'output senza l'utilizzo di cavi o elettronica esterna di alcun tipo; grazie a questo dispositivo, i dipendenti, in una conversazione col proprio business manager, propongono lo sviluppo di un software di home security controllato in remoto da un PC. Tale software, SafeHome appunto, includerebbe un'interfaccia per visualizzare i dati dei sensori, un pannello di controllo dei regolatori, nonché altre funzioni di gestione di un ambiente smart come ad esempio accensione e spegnimento di luci ed altre apparecchiature elettroniche.
La tecnica di sviluppo software utilizzata da Pressman è la cosiddetta *scenario-based design*, nella quale egli delinea - o in questo caso, fa delineare ai personaggi del suo caso di studio - degli scenari di utilizzo dell'applicazione, come ad esempio quando "l'utente desidera controllare lo stato di sicurezza della propria smart home ed accede all'applicazione", che possono essere facilmente analizzati per estrarre le funzionalità desiderate dai fittizi stakeholders.
Da questi scenari basilari, tre in totale, l'autore costruisce tramite i formalismi tecnici del *Linguaggio UML*, dei diagrammi di Caso d'uso (*Use Case diagrams*), dai quali ricava diagrammi di Attività (*Activity diagrams*), che successivamente raffina in tutti gli altri modelli UML usati per descrivere design e architettura del software, sino a spiegare nella sua totalità il ciclo di vita di un software.

Apparentemente in letteratura non vi sono però realizzazioni del caso, al di fuori di quella, comunque teorizzata, proposta da Pressman stesso. Esiste anche un prototipo *web based* dell'applicazione [11], che egli utilizza come ausilio nella stesura del suo altro libro di testo *Web Engineering: a Practitioner's Approach*

[12], dedicato questa volta all'analisi di quest'altra branca dell'ingegneria del software: tuttavia, questa web app è implementata solo parzialmente e la sua analisi e confronto con la soluzione proposta in questo elaborato non è significativa.

openHAB

OpenHAB è un software open source di domotica sviluppato in Java dalla *openHAB Foundation*, una organizzazione no-profit registrata in Germania che si propone l'obiettivo di "*educare il pubblico alle possibilità che il software libero offre nel campo delle smart home*" [4].
Il software di per sé è offerto come servizio web allo scopo di connettere e permettere la comunicazione tra dispositivi IoT diversi, venduti da un'ampia gamma di aziende.
Dal punto di vista dell'utente finale, la piattaforma dà la possibilità di costruire un'interfaccia per la visualizzazione e il controllo dei dati che sensori, attuatori, regolatori e apparecchiature varie installate nel proprio ambiente smart, scambiano con e attraverso openHAB.

OpenHAB, il cui acronimo sta per open Home Automation Bus, è basato sul progetto *Eclipse SmartHome*, della *Eclipse Foundation*, una piattaforma middleware che connette tra loro dispositivi smart e offre funzionalità software tramite API (*Application Programming Interface*).
Dal punto di vista del produttore di dispositivi smart, il progetto offre la possibilità di sviluppare un connettore, detto *binding*, per accedere ai dati in output senza dover in alcun modo adattare l'hardware; questa astrazione permette di utilizzare il proprio device in un ampio numero di soluzioni nel campo della domotica.

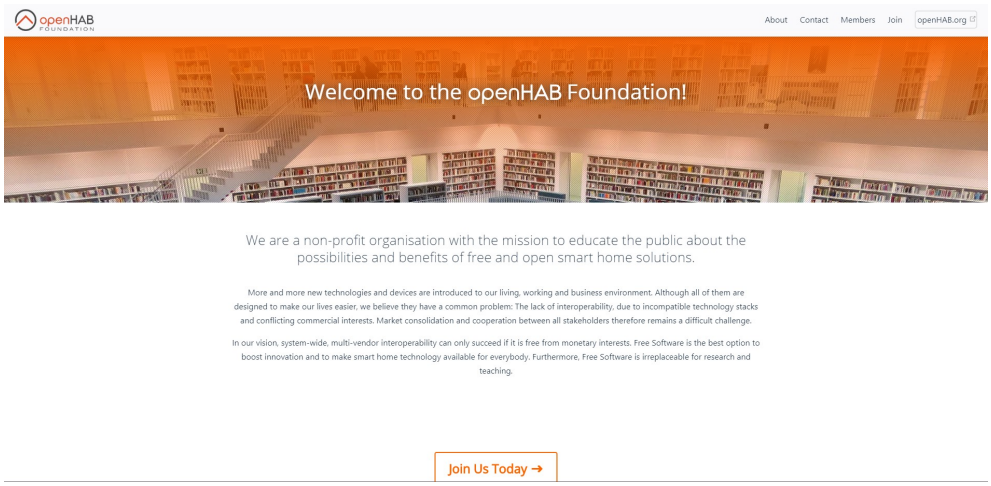

Figura 1: la pagina principale del sito web della *openHAB Foundation* [4].

La piattaforma di openHAB tramite il suo framework permette anche la specificazione di precise regole per l'automazione di processi, le quali una volta "scattate" avvieranno script con funzionalità di controllo e manipolazione dei dispositivi installati nella propria smart home: ad esempio, è possibile pianificare una regola che accenda le luci del soggiorno al tramonto, o che avvii il termoregolatore quando la temperatura

in bagno scende sotto una precisa soglia. Le potenzialità offerte dalla web app sono quindi molto numerose, e ad oggi sono supportati circa 200 device di produttori più che noti nell'ambito degli elettrodomestici (nomi tra i quali figurano Amazon, Apple, Sony, LG, e molti altri).

Il framework per poter essere utilizzato correttamente richiede determinate capacità di sviluppo, definite "hacking skills" dagli autori stessi [6], perché per la sua configurazione, costruzione ed utilizzo richiede installazioni di particolari software, scrittura di file testuali, analisi di log ed una certa competenza tecnica generale.

Concetti di openHAB

L'applicativo utilizza un insieme di concetti che l'utente deve comprendere prima di poter realizzare la propria interfaccia per la sua smart home, e per poterla quindi utilizzare per il controllo dell'ambiente domestico.
Il principale è il concetto di *sitemap*, termine usato per indicare la UI stessa che lui o lei dovrà creare per poter visualizzare o controllare i dati in output dai dispositivi che deciderà di installarvi. OpenHAB infatti come detto offre sì interfacce utente, ma queste vanno configurate per adattarle alle proprie necessità e soprattutto ai propri device: questa può avvenire tramite modifica di file testuali o anche tramite l'utilizzo dell'app stessa.
Ogni elettrodomestico smart o servizio web di cui l'utente desidera il controllo è modellato dal concetto di *Thing*, che vanno installati e configurati uno ad uno; ogni thing possiede vari *Channel,* proprietà che rappresentano le varie funzionalità offerte dal dispositivo - così, ad esempio, un sensore che misura la qualità dell'aria avrà un canale per i valori di anidride carbonica ma anche uno per il monossido di carbonio, nonché un terzo per una possibile descrizione qualitativa ("buona", "ottima" etc.) e via dicendo.

I produttori di dispositivi smart, come detto, scrivono dei connettori, che come nel progetto sorgente Eclipse SmartHome vengono chiamati *binding*; ogni binding può offrire una o più thing, ciascuna delle quali avrà uno o più channel. L'utente finale, dopo aver installato uno di questi binding, può inserire nella propria sitemap le thing che offre, e collegare i canali facendo uso del concetto di *Items*, ovvero rappresentazioni dei suddetti channels. L'algoritmo scritto nel binding poi si occuperà di aggiornare i valori di questi items con l'output dei dispositivi che governa, e in questo modo l'utente vedrà i dati che vengono offerti dai device installati nella propria smart home.

```
// ***** Bathroom items start here *****

Number Bathroom_TEMP04 "Temperature" < temperature > (Bathroom, Temperature)
{channel = "safehome_se:model_thf:THF01:temperature_sensor#curr_temperature"}

Number Bathroom_HUMD05 "Humidity" < humidity > (Bathroom, Humidity)
{channel = "safehome_se:model_thf:THF01:humidity_sensor#curr_humidity"}

Contact Bathroom_FLOD08 "Water leakage [MAP(safehome.map):%s]" < water > (Bathroom, Flood)
{channel = "safehome_se:model_thf:THF01:flood_sensor#curr_flood"}

Switch Bathroom_ArmDisarm "Arm sensor [MAP(safehome.map):%s]" <switch> (Bathroom, ArmDisarm)
```

Figura 2: alcuni *items* della soluzione proposta, che si collegano alle *things* tramite la configurazione del *channel* corretto specificato nel *binding* che è stato sviluppato con questo elaborato.

Esistono anche altri concetti, ma non sono fondamentali per l'utilizzo base dell'applicazione, né per la comprensione del lavoro di tesi qui presentato, per i quali si rimanda alla documentazione ufficiale di openHAB [6].

Interfaccia utente di openHAB

OpenHAB, attualmente nella sua versione 2, offre all'utente finale tre diverse interfacce con finalità differenti: *Basic UI*, *Paper UI* e *Classic UI*; queste tre sono presenti nella configurazione del pacchetto Standard, ma è possibile installarne altre tramite specifici add-on prodotti da terze parti. In *Figura 3* è possibile vedere una tipica schermata iniziale dell'esecuzione del server, nella quale si notano tra gli altri componenti, proprio le tre interfacce, con anche un piccolo scorcio della loro grafica.

L'interfaccia Basic è quella che l'utente si troverà più spesso ad utilizzare perché offre funzionalità di visualizzazione delle sitemap presenti nei file di configurazione. Se ce ne fosse più di una, all'utente verrebbe dapprima presentata una schermata di selezione. Da questa interfaccia, il cui stile è ispirato al material design di Google, è possibile interagire con i dispositivi tramite i componenti web offerti da openHAB, come ad esempio switch per rappresentare interruttori di luci o campi testuali per visualizzare i l'output dei device installati.

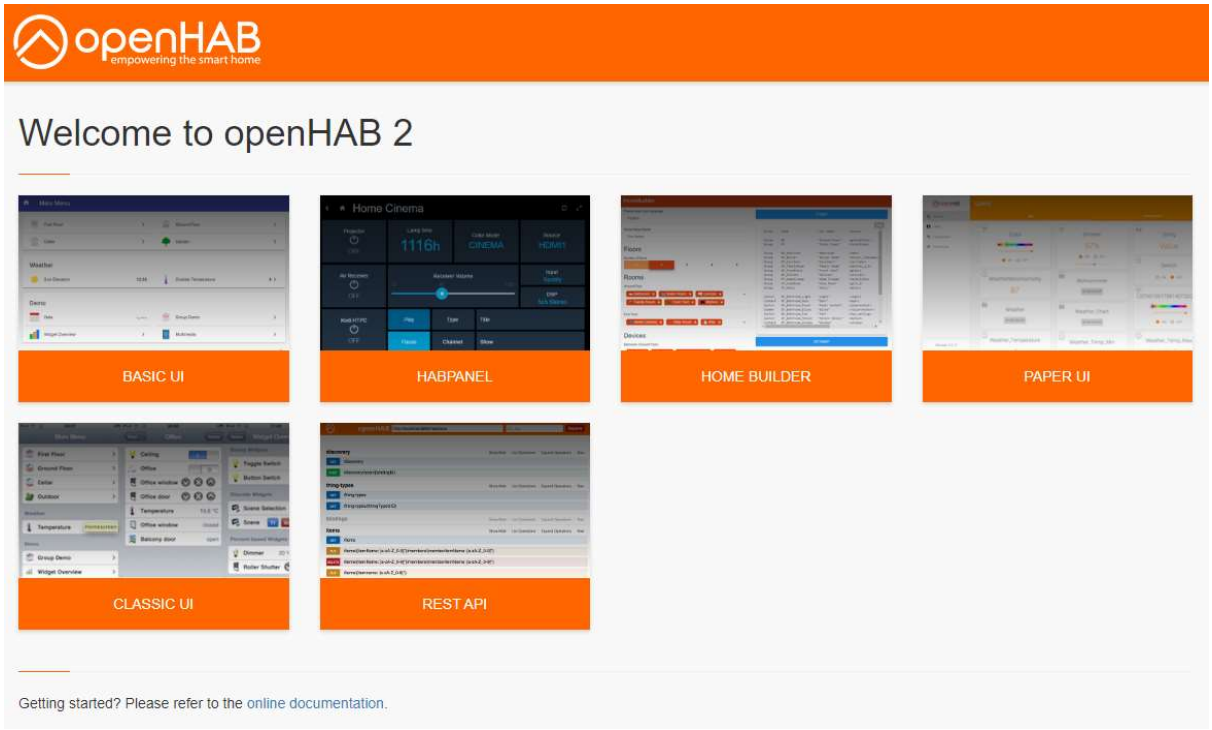

Figura 3: schermata principale di un'esecuzione del server di openHAB. Sono visibili le tre UI di default; inoltre è stato installato il componente REST API per l'esecuzione delle chiamate alle omonime interfacce.

L'interfaccia Paper offre funzionalità di installazione e configurazione, anche se non è del completamente sostituibile all'editing dei file di testo: alcune specifiche richieste di setup, infatti, necessitano che l'utente

vada a modificare comunque determinati file. Inoltre offre la possibilità di visualizzare i dati senza dover necessariamente creare una sitemap, ma la sua UI è parecchio limitata in questo senso ed appare evidente che non è pensata per quello scopo.

La Classic UI invece è semplicemente l'originale interfaccia di openHAB, ha un look simile ad un'applicazione iOS ed ha le stesse funzionalità che ha la Basic UI.

Oltre a queste tre, un avvio dell'applicativo ci presenta, nella sua pagina principale, altre due scelte, *Home Builder* e *HABPanel*: la più interessante è certamente la prima, perché offre un metodo automatico di generazione dei file di configurazione - gli stessi che altrimenti l'utente si ritroverebbe a dover scrivere da solo. Qui è possibile creare la propria sitemap rappresentante l'ambiente smart che si vuole controllare, ed addirittura specificare e configurare i componenti web che modellano gli Items; non è potente quanto una configurazione testuale, comunque, perché ad esempio non offre la possibilità di inserirvi Things, e limita la scelta dei suddetti Items a quelli offerti dal sistema base - niente dispositivi reali, dunque, ma solo campi astratti come Number o String.
HABPanel invece è l'ennesima interfaccia di visualizzazione, questa volta più simile ad un pannello di controllo, ma con le stesse funzionalità delle altre.

Stato dell'arte: approcci per la simulazione di una smart home

In letteratura si possono trovare molti riferimenti alla simulazione di un ambiente smart, perché le case intelligenti sono ormai di grande interesse sia commerciale che scientifico nel campo dell'automazione e dell'*Internet of Things*.
Le pubblicazioni analizzate possono essere sommariamente classificate in due tipi: **simulazioni a scopo di test per la valutazione di modelli IoT** e proposte di **sistemi che dimostrano o aiutano il design** di una casa intelligente.
Nel primo caso rientrano ad esempio *OpenSHS* [13] e *SIMACT* [14], mentre tra i secondi è possibile trovare soluzioni come *Home I/O* [15], o ad esempio pubblicazioni quali '*Smart home simulation system*' [16] o '*Simulation modeling of "Smart Home" operations using data on a man's position*' [17].

- *OpenSHS* [13] è un simulatore 3D di una smart home orientato a ricercatori nel campo dell'IoT che desiderano un ambiente virtuale in cui creare e valutare un modello e generare un insieme di dati di attività di sensori. L'applicazione utilizza un approccio misto tra simulazione interattiva e modello statistico basato su precise equazioni. Gli autori hanno anche sviluppato una serie di librerie che implementano dispositivi smart pronti per essere installati nell'ambiente virtuale.
  Il tool copre l'intera fase di design di una smart home nel contesto desiderato dal ricercatore e ne simula il funzionamento generando specifici eventi; da questi ne ricava automaticamente il dataset aggregato. È presente anche una interessante funzionalità di *fast forwarding* nella parte di raccolta dati basata sull'interazione.

- *SIMACT* [14] condivide alcuni dei suoi obiettivi con il precedente, perché anch'esso è un simulatore 3D indirizzato ai ricercatori, questa volta nel sotto campo della IoT detto *activity recognition*, allo scopo di condurre esperimenti con dati realistici. Il tool è sviluppato in Java e permette ad un'applicazione esterna (come un rilevatore smart di presenza e di movimento) di connettersi con

il database offerto e interagire con gli scenari e i dati sviluppati assieme al software. Interessante la funzionalità di scripting per la creazione di nuovi scenari di dati.

- *Home I/O* [15] invece è un'applicazione che simula una casa intelligente a scopo educativo indirizzato ad un pubblico giovanile: è stata creata infatti a somiglianza di un videogame in prima persona in cui l'utente esplora l'ambiente domestico e interagisce con i moltissimi (174 per la precisione) dispositivi simulati utilizzando un pannello di controllo virtualizzato all'interno del mondo di gioco. Lo scopo è quello di introdurre ragazzi preadolescenti ai concetti della domotica con particolare enfasi sui consumi sostenibili. L'intero sistema è inoltre personalizzabile tramite scripting con *Scratch [18]*, un progetto educativo dell'università *MIT* di Boston, azione che può essere svolta anche dagli utenti stessi con supervisione.

- In *Smart home simulation system* [16] viene invece proposta una soluzione nella quale oltre alla simulazione di un dispositivo viene virtualizzato anche il comportamento di una persona, tramite un algoritmo di Intelligenza Artificiale basato su precondizioni ed obiettivi. Nell'ambiente simulato sono però virtualizzati soltanto sensori di movimento, che rilevano l'attività dell'ipotetico utente. Non vengono forniti scenari di test.

- Nella pubblicazione [17] infine viene presentata (ma non realizzata) una soluzione per controllare automaticamente alcuni dispositivi di una smart home in base alla posizione di una persona: essa verrebbe rilevata da un device in grado di distinguere i tre stati *seduto*, *sdraiato* e *in piedi* e in base a fattori come l'ora del giorno o la stanza in cui viene rilevata la presenza verrebbero attivate funzionalità come accensione della luce, della TV o controllo delle tapparelle.

La simulazione di un ambiente smart è inoltre di interesse scientifico anche nel settore dell'ingegneria dei software *self-adaptive* [19], ovvero sistemi molto avanzati capaci di adattarsi automaticamente a cambiamenti nel suo contesto di operazione, riconfigurandosi e cambiando proprietà e funzionalità.

Per quanto riguarda infine la virtualizzazione del funzionamento di sensori, anche qui si trovano vari articoli in letteratura, come [20] [21] in cui rispettivamente si modellano un igrometro e un termostato. È inoltre possibile utilizzare *Simulink* [22], tool basato su MATLAB, per la modellazione di sensori di ogni genere. Tali soluzioni comunque analizzano il funzionamento dei dispositivi nel dettaglio, tenendo conto di ogni legge fisica correlata, cosa che in SafeHome si è optato per omettere in nome della semplificazione e soprattutto per focalizzare l'attenzione sul processo di design e sviluppo dell'applicativo attenendosi agli standard imposti dall'ingegneria del software.

1.2 Soluzione proposta

La soluzione che viene proposta in questo lavoro di tesi - da ora in poi chiamata *sistema SafeHome* - è appunto una realizzazione dell'ipotetico software inventato per il caso di studio omonimo avente le funzionalità di base elencate nella *sezione 2.1* e soddisfacente i requisiti definiti nella *sezione 2.2*.
È stata data particolare attenzione al design e all'architettura della soluzione in accordo con i principi definiti dall'ingegneria del software.

Il sistema SafeHome è stato implementato mediante un **ambiente virtuale** composto da due applicazioni, entrambe scritti in Java, che possono essere considerate sistemi a sé, in quanto in loro funzionamento è totalmente indipendente: *Sensori* ed *Ambiente*.

Il primo ha una funzione di modellazione e virtualizzazione della misurazione di un determinato insieme di dispositivi, tra i quali troviamo, oltre a quelli elencati da Pressman [23], rilevatori comunemente presenti in smart home quali termometri ambientali, igrometri (*m. di umidità*) e indicatori di qualità dell'aria (che effettuano rilevazione dei livelli di anidride carbonica, monossido di carbonio, polveri sottili del tipo PM 2.5 e un insieme di altri gas dannosi detti *Composti Volatili Organici Totali*).

La scelta di inserire proprio questi sensori è stata fatta conducendo una ricerca dei modelli di sensori più venduti e più frequentemente ritrovabili all'interno delle più moderne smart home [24].
Il modello del loro funzionamento effettivo, come si vedrà nelle sezioni successive, è in realtà di natura generale, volutamente astratto per poter permettere una futura sostituzione di una o più virtualizzazioni con una controparte reale senza compromettere irreversibilmente l'intera stabilità del sistema, e contemporaneamente per focalizzare lo sforzo produttivo sulla fase di creazione e design.

Il sistema *Ambiente*, invece, è una rappresentazione del luogo che l'utente desidera monitorare, modellato attraverso la presenza di varie stanze, aventi in ogni momento tutti i dati di interesse ai sensori del sistema SafeHome. Tali dati possono essere variati a piacimento dall'utente, che potrà poi osservarne la raccolta tramite l'interfaccia principale o tramite le interfacce a mo' di display di ciascun misuratore.

Viene utilizzato il framework di openHAB come interfaccia principale per la visualizzazione dei dati, nonché come pannello di controllo, idealmente remoto, dell'intera smart home. Da lì è possibile, in particolar modo, inserire un allarme di sicurezza che avviserà l'utente in caso si verificassero situazioni indesiderate.
L'aggiornamento degli items nella sitemap di openHAB è stata realizzata tramite l'utilizzo delle API (*Application Programming Interfaces*) offerte dalla piattaforma stessa.
I due sistemi inoltre sono in costante comunicazione al fine di virtualizzare le misurazioni tramite il servizio di messaggistica real-time offerto da *PubNub* [45], un noto webservice di comunicazione online, che funge da intermediario.

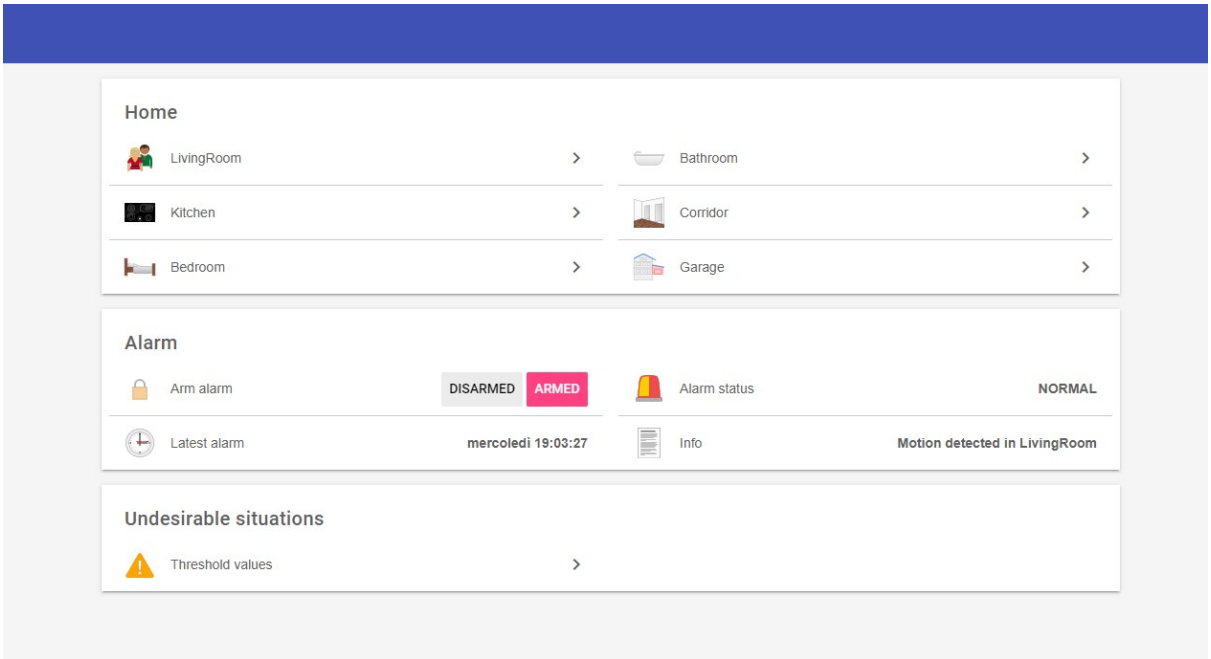

Figura 4: la *sitemap* del sistema SafeHome, interfaccia remota della soluzione proposta.

Valutazione

Nel ciclo di vita di un software una delle sfide più impegnative che il designer del sistema deve affrontare è soddisfare i cosiddetti *requisiti non funzionali* dell'applicazione, ovvero quelle proprietà qualitative desiderate dagli stakeholders che sono quasi sempre difficili o totalmente impossibili da quantificare. Esempi tipici di requisiti non funzionali che si possono incontrare in pressoché qualunque applicativo sono il *tempo di risposta* del sistema, la *semplicità d'utilizzo* dell'interfaccia o ancora la garanzia che il prodotto *funzioni sempre* - ma questi sono solo alcuni.

In letteratura nel campo dell'ingegneria del software infatti si possono trovare diverse classificazioni [25] [26] [27] e tassonomie per i requisiti non funzionali. In [25] ad esempio i suddetti vengono raggruppati nelle categorie di *disponibilità*, *performance*, *affidabilità*, *sicurezza*, *manutenzione*, *look and feel*, *usabilità* e *aspetti legali*.

Tra tutte queste 'non funzionalità', in questo elaborato vengono selezionate in particolar modo ***affidabilità*** e ***performance***, per le quali il comportamento del sistema SafeHome viene analizzato tramite **tre scenari di test**, al fine di valutare la risposta del sistema a **situazioni di incertezza** tipiche della realtà. Ognuno di questi casi di test viene modellato e iniettato accuratamente in SafeHome e analizzato all'interno di un contesto isolato, controllato e inoltre standardizzato: diversi parametri rilevabili dai dispositivi vengono infatti variati seguendo un andamento ben definito in precisi istanti di tempo. In tutti e tre gli scenari è stato osservata la proprietà *Numero di raggiungimenti dello stato InAllarme da parte del sistema*, ovvero il numero di volte in cui l'allarme della smart home è scattato a seguito del rilevamento di una situazione considerata potenzialmente pericolosa. Questi dati sono stati raccolti al variare di precisi parametri, ad esempio probabilistici, immessi in input, ed è stato effettuato un numero di ripetizioni sufficientemente elevato da garantire la confidenza dei risultati ottenuti, che sono stati dunque messi a confronto con quelli ottenuti nel solo contesto standard.

Il pannello di testing che è stato realizzato per l'implementazione dei suddetti scenari è comunque utilizzabile dall'utente anche nella versione dimostrativa della soluzione, che ad esempio può farne uso per effettuare i propri test nell'ambiente controllato e quasi del tutto esente da errori esterni che è SafeHome. Un utente interessato, infatti, potrebbe selezionare uno dei tre scenari proposti e inserire i valori di probabilità d'errore desiderati ed eseguirli nel contesto standard oppure addirittura in uno scenario realizzato ad hoc tramite i processi di variazione dei dati offerti da *Ambiente*.

2. Analisi del caso di studio

Il caso di studio SafeHome viene utilizzato dal Dr. Pressman durante tutta la stesura del proprio libro di testo come esempio ricorrente al fine di spiegare i processi ingegneristici interni che caratterizzano la creazione di un prodotto software.
Questi esempi, come detto, sono descritti attraverso fittizie conversazioni tra sviluppatori e stakeholders di SafeHome, e nel libro addirittura è possibile trovare il processo iniziale di esposizione dell'idea di SafeHome ancora quando essa è in fase concettuale. È stato possibile per l'autore di questo lavoro di tesi ricavare da ciò le funzionalità desiderate ed elicitare i requisiti esattamente come se avesse partecipato a, o svolto in prima persona, tali processi.
I personaggi fittizi inventati dal Dr. Pressman sono stati, così, trattati come clienti, collaboratori e stakeholders reali durante tutto questo lavoro.

2.1 Funzionalità principali

Dagli esempi [28] [29] che descrivono un ipotetico meeting tra stakeholders e sviluppatori è stato possibile ricavare le seguenti proprietà e funzionalità che sono di particolare interesse alle persone considerate come clienti:

G1. **riconoscere** la presenza di **situazioni indesiderate**, come incendi, violazioni di domicilio, livelli elevati di monossido di carbonio, allagamenti, ed altri [30]

G2. essere programmabile e **configurabile dall'utente finale** [30]

G3. **effettuare** automaticamente **una telefonata all'utente** e ad una agenzia di monitoraggio quando viene rilevata una situazione indesiderata [30]

G4. essere accessibile da remoto allo scopo di **monitorare i dati di output** dei sensori [28]

Questi servizi, che normalmente vengono definiti *goals del sistema*, sono stati utilizzati nella stesura ed elicitazione dei requisiti funzionali del sistema SafeHome nella sezione successiva.
Come si può notare, queste funzionalità sono di carattere estremamente vago e generale, cosa assolutamente intenzionale da parte di Pressman: la mancanza di requisiti più specifici, come ad esempio un'indicazione del tempo di risposta del sistema o una completa specificazione delle situazioni indesiderate, ha lasciato notevole **libertà di implementazione** durante il lavoro di tesi; tale libertà è stata sfruttata quanto più possibile, e nelle sezioni successive verranno descritte anche funzionalità che non vengono menzionati nel caso SafeHome, dipendenti appunto dall'implementazione del sistema in un ambiente completamente virtuale.

2.2 Analisi e specificazione dei requisiti

In questa sezione vengono analizzati i requisiti, sia funzionali che non funzionali, del sistema SafeHome derivati dall'analisi delle funzionalità appena elencate.
I requisiti sono stati definiti utilizzando la tecnica di **elicitazione mediante interviste**, appunto ricavate da una ricerca approfondita sul libro di testo di Pressman.

La **specificazione**, invece, è **formale**, ed è stata svolta attenendosi agli standard specificati in [31]. Tale documento specifica che l'elenco dei requisiti debba ricondursi ad una di queste tre forme sintattiche:

- [Condition] [Subject] [Action] [Object] [Constraint]
- [Condition] [Action or Constraint] [Value]
- [Subject] [Action] [Value]

I requisiti del sistema SafeHome sono scritti seguendo questo standard sintattico, ma in italiano. Inoltre viene usata la parola *deve* per esprimere il concetto di *shall* - che è una parola chiave definita nel documento.

Nonostante sia formale, la specificazione dei requisiti non presenta un linguaggio tecnico, e la lettura è indirizzata a tutti i possibili stakeholders, anche a chi non possiede competenze nel campo dell'ingegneria del software.

Viene fatto di seguito un raggruppamento logico dei requisiti in base alle parole chiave definite nell'elenco delle funzionalità nella *sezione 2.1*. Successivamente al primo gruppo di requisiti (detti *funzionali* in quanto derivati dalle funzionalità) viene elencato un secondo gruppo di requisiti (detti *non funzionali*), che specificano ad esempio caratteristiche non sempre quantificabili come il tempo di risposta o la facilità d'uso dell'interfaccia utente. Tali caratteristiche vengono elencate da Pressman come vincoli (*constraints*) [23].

Riconoscere situazioni indesiderate

1. il sistema deve permettere il riconoscimento di almeno le seguenti situazioni indesiderate qui elencate: violazione di domicilio; incendio; allagamento; livelli malsani di monossido di carbonio
2. il sistema deve automaticamente far scattare un allarme in caso di situazioni indesiderate

Configurazione

3. il sistema deve essere configurabile dall'utente
4. il sistema deve poter permettere aggiunta, rimozione e modifica di dispositivi di misurazione
5. il sistema deve permettere tramite un pannello di controllo attivazione e disattivazione remote dell'allarme dei sensori installati
6. il sistema deve permettere l'attivazione e la disattivazione dell'allarme

Avvertimento dell'utente in caso di situazioni indesiderate

7. il sistema deve avvertire l'utente * mediante una telefonata in caso di situazioni indesiderate

* il caso di studio non fa esplicito riferimento all'avvertimento dell'utente, ma piuttosto ad una non meglio specificata "*monitoring agency*" [30].

Monitoraggio dei dati in output

8. il sistema deve permettere la visualizzazione dei dati in output di ciascun sensore
9. il sistema deve essere accessibile da remoto
10. il sistema deve tenere traccia degli eventi che accadono durante il suo funzionamento **

** viene fatto esplicito riferimento ad un sistema di log degli eventi, fornendo l'esempio pratico di evento come "*a sensor has been activated*"

Requisiti non funzionali

11. il sistema deve riconoscere lo stato di non operazione di un sensore [23]
12. il sistema deve essere facile da usare per l'utente [23]
13. il sistema deve riconoscere un evento in un sensore sempre entro pochi secondi [23]

I requisiti non funzionali verificati negli scenari di test nella *sezione 7.2* non vengono qui riportati perché non sono mai esplicitamente nominati nel caso di studio, nonostante rappresentino comunque tre requisiti molto frequenti nell'ambito della progettazione software, e la loro comparsa in questa lista non sarebbe significativa per il confronto con SafeHome; l'analisi degli stessi viene pertanto rimandata alla suddetta sezione, in cui si riporta anche una possibile formulazione realistica della caratteristica qualitativa desiderata.

3. Descrizione della soluzione: il sistema SafeHome

In questa sezione viene descritto il funzionamento del sistema SafeHome, analizzando tutti i suoi sottosistemi: oltre ai già citati Sensori e Ambiente, di cui viene spiegato il comportamento a livello generale e non tecnico ma comunque completo, vengono descritte le interfacce costruite per openHAB e il funzionamento del servizio di comunicazione *PubNub*. Inoltre viene descritta la struttura del binding che è stato scritto per openHAB.
Una guida tecnica per la creazione di una sitemap di openHAB diversa da quella proposta è presente nella *sezione 6.1*.
È stata data particolare enfasi alla *requirements traceability*, cioè quando una funzionalità di uno di questi sottosistemi soddisfa uno o più tra i requisiti appena elencati.

3.1 Sensori

Il sistema Sensori (nel codice, *Sensors* o *Simulated Sensors*) ha la funzione di **modellare e virtualizzare il funzionamento** di un determinato numero di sensori. Possiede inoltre una user interface in locale atta a visualizzare i dati in output e che, in pratica, rappresenta la controparte virtuale del comune display che ogni misuratore reale possiede.
La scelta di inserire una GUI (*Graphical User interface*) permette di soddisfare il requisito 8 anche senza dover necessariamente fare uso di openHAB; l'utente del sistema SafeHome, infatti, può scegliere di osservare l'andamento dei dati utilizzando questa UI, invece che il runtime di openHAB che è inteso come principale, e questa è una libertà che viene assolutamente lasciata all'utilizzatore; anche nella realtà, infatti, il possessore di una smart home può sempre consultare i display dei sensori invece che il pannello elettronico della piattaforma, e questa è una caratteristica delle smart home che si è voluto riportare anche nel design del progetto. Per dovere di cronaca, comunque, qualora la piattaforma di openHAB dovesse essere installata su un server accessibile online (come comunque richiede la funzionalità G2 del caso di studio) l'utente non potrebbe accedere da remoto al 'display', ma dovrebbe obbligatoriamente utilizzare il runtime della web app.

Tornando al sistema in analisi, nel suo algoritmo viene fatta una distinzione logica tra *Sensori e Dispositivi*: i primi rappresentano entità capaci di effettuare rilevazioni o misurazioni; gli ultimi, invece, modellano oggetti fisici reali, e contengono uno o più misuratori differenti a seconda del proprio modello.
Ogni dispositivo può essere scelto da un elenco di modelli supportati, ognuno dei quali si distingue per la varietà di sensori contenuti. Ad esempio, un dispositivo *modello TH* potrà effettuare misurazioni di temperatura ed umidità proprio perché tale modello supporta quei due tipi di sensori. Una lista completa dei modelli supportati è disponibile nella *sezione 3.4* in *Figura 14*.
Un sensore, al contrario, non viene distinto da un modello ma dal tipo di dato che è in grado di rilevare. Abbiamo così rilevatori di fumo, di movimento e di perdita d'acqua - che corrispondono a tre delle quattro situazioni indesiderate specificate nel requisito 1 - e misuratori di temperatura, di umidità e di qualità

dell'aria. Questi ultimi, oltre alla quantità di monossido di carbonio voluta da Pressman, sono in grado di avere in output un valore di anidride carbonica (*CO2*), di polveri sottili PM 2.5 e di un insieme di altri gas comunemente raggruppati sotto il nome di Composti volatili organici totali (*TVOC*).

Il funzionamento di ciascun sensore come detto è stato reso volutamente astratto, e sono stati ridotti ad entità che, ricevuto un dato, vi applicano un arrotondamento in base alla propria sensibilità, e - nel caso in cui questa opzione venga selezionata - vi sommano una piccola incertezza di misura.
Entrambi questi parametri, per tutti i sensori in analisi, sono stati presi da datasheet di sensori reali: ad esempio, il modello di igrometro [32] sulla propria misura di umidità relativa applica un errore di ± 4%, e possiedono una sensibilità di 1 cifra decimale.
I rilevatori inoltre possiedono una soglia interna, superata la quale 'scattano' (tecnicamente, modificando il proprio stato): nel caso in cui dal pannello di controllo si sia scelto di inserire l'allarme, questo cambiamento lo attiverà.
Come si vedrà più in dettaglio nelle prossime sezioni, inoltre, il funzionamento di un sensore è indipendente dal dispositivo in cui è virtualmente installato.

Ogni rilevatore effettua periodicamente una richiesta di poter misurare un dato del proprio tipo, ad esempio di temperatura, ma senza che l'oggetto sia conscio di dove esso stesso sia collocato: in altre parole, un rilevatore di fumo non sa di essere installato in un dispositivo piazzato in cucina, conosce soltanto il modo per effettuare una rilevazione e il modo per emettere in output il dato misurato. Questa semplificazione, oltre ad aver permesso di rendere il modello più realistico e sufficientemente vicino al reale funzionamento di un vero misuratore, permette in futuro di poter sostituire una o più parti del sistema con una controparte reale con un adattamento del codice quasi nullo, e senza compromettere irrimediabilmente la stabilità dell'intero sistema.
Una volta ottenuto il dato dall'esterno, come detto un sensore vi applica la propria misurazione, e una volta terminato questo processo, informa del nuovo valore sia la propria UI, sia il runtime di openHAB.
Unica eccezione è il rilevatore di movimento, che è l'unico sensore che non fa *polling* periodico della richiesta di misurazione, bensì gli viene 'consegnato' un valore - al quale poi applicherà internamente i propri calcoli - al verificarsi del cambiamento del dato *Movimento* nell'area in cui è virtualmente installato.

I dispositivi possono essere accesi o spenti durante l'esecuzione dell'applicazione, tramite l'utilizzo dell'interfaccia locale: questo provocherà lo spegnimento, o l'accensione, di ogni sensore virtualmente installato all'interno del device. In caso di spegnimento, proprio come specificato dal requisito 11, il sistema sarà in grado di riconoscere lo stato di non funzionamento del sensore ed ovviamente non verrà prodotto alcun output.

È possibile agire su determinati file testuali, contenuti nelle cartelle di installazione del progetto, per cambiare la configurazione iniziale di sensori e dispositivi, modificandone uno od aggiungendo altri misuratori all'interno di uno stesso dispositivo, seguendo un processo che verrà spiegato in dettaglio più avanti. Queste modifiche devono essere effettuate prima di avviare l'applicazione, in quanto cambiamenti ai file in corso d'opera verranno ignorati.

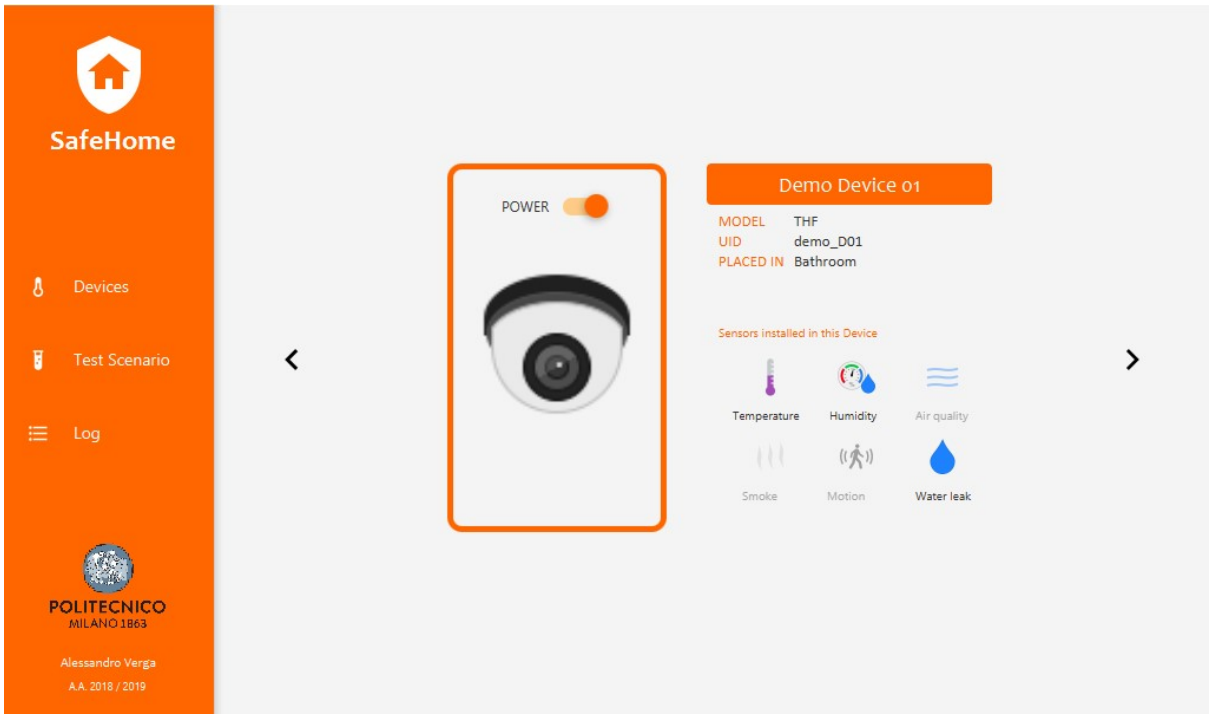

Figura 5: la home del sistema *Sensori*, che mostra la pagina di riepilogo dei dispositivi.

L'applicazione infine possiede un'interfaccia con la quale può comunicare con openHAB, che viene meglio analizzata nella *sezione 5.1*.

Il **Log** invece ha funzionalità di riepilogo di tutti gli avvenimenti del sottosistema: vengono registrate ogni per ogni misura il momento e il valore 'reale' ricevuto da Ambiente e il valore una volta effettuata la misurazione, messaggi relativi all'attivazione di scenari di test, cambiamenti di stato in sensori e dispositivi (come ad esempio fallimento o spegnimento) ed eventuali messaggi e cause d'errore.

I tipi di dato supportati, nel dettaglio:

In *Figura 6* vengono presentati per ogni tipo di dato tutti i suoi parametri utilizzati in SafeHome.
Oltre a unità di misura, sensibilità del sensore (ovvero, il minimo valore in grado di produrre una variazione visibile nell'output) e accuratezza (utilizzata nello scenario di test *Errore nella misurazione*), si indica la soglia per la situazione indesiderata associata.

Tipo di dato	Unità di misura	Sensibilità di misura	Accuratezza di misura	Soglia situazione indesiderata
Temperatura	°C	0.1	± 0.4 °C	12 (MIN) - 35 (MAX)
Umidità relativa	%	0.1	± 4 %	30 (MIN) - 70 (MAX)
Qualità dell'aria CO2	ppm	1	± 30 ppm	7000
Qualità dell'aria CO	ppm	0.1	± 15 ppm	35
Qualità dell'aria PM 2.5	ppm	0.1	± 6.8 ppm	55
Qualità dell'aria TVOC	ppm	0.1	± 200 ppm	3000
Presenza di fumo	obs / m	0.1	± 1.2 obs / m	2.5 *
Presenza di movimento	-	0.01	± 0.15	0.15 *
Presenza di allagamento	V	0.1	± 0.3 V	2.4 *

Figura 6: ad ogni sensore, capace di misurare un solo tipo di dato, è associato un insieme di parametri: unità, sensibilità e accuratezza di misura. Si indicano con * i valori di soglia di situazione indesiderata che sono intrinsecamente legati al sistema, che non sono cioè lasciati alla configurazione dell'utente.

Ogni valore di accuratezza e sensibilità è stato registrato da almeno un dispositivo reale, nell'ordine [32] [32] [33] [34] [35] [33] [36] [37] [38] . Il valore di accuratezza per la rilevazione del movimento è stato impostato a 0.15 (su una scala da 0 a 1) perché in [37] si specifica che la configurazione consigliata è tra 0.2 e 1 di giorno, e 0.01 e 0.2 di notte (i valori specificati nella fonte citata, in realtà, sono differenti a causa del cambio del fondo scala, che nella fonte citata varia da 0 a 50 mentre in SafeHome da 0 a 1; i valori sono stati quindi rapportati). Inoltre, impostare questo valore esattamente uguale al limite indesiderato ha permesso di simulare situazioni reali relativamente infrequenti, ma possibili, in cui il rilevamento avviene a causa di un errore.

I valori di soglia, invece, sono stati ricavati consultando nell'ordine [39] [40] [41] [42] [43] [44] [36] [37] [38], spesso autorevoli organizzazioni sanitarie o di nuovo gli stessi produttori dei dispositivi.
Una scelta di design è stata resa obbligatoria dai limiti di PM 2.5, CO e TVOC, parametri frequentemente misurati in lunghi periodi, spesso nell'ordine delle ore: in SafeHome vengono considerati **parametri istantanei** e come loro valore soglia è stato preso il numero più basso possibile tra le esposizioni a breve termine considerate dannose dalle fonti citate.

Il dato *presenza di allagamento* è misurato in volt seguendo il funzionamento comune dei misuratori [38], i quali non appena rilevano dei liquidi permettono il passaggio di una certa corrente in base alla quantità di liquido, la quale esercita una tensione ai capi del sensore; essa viene quindi prontamente confrontata con un valore di soglia interno. Non vi è alcuna unità di misura standard per la grandezza '*quantità di allagamento*', e nella realizzazione del software si è dovuto optare per una simile scelta di design.

Per la rappresentazione testuale dei dati visualizzabile sia nelle interfacce locali di Sensori e Ambiente, sia in quella remota di openHAB, si rimanda alla sezione successiva, *Figura 9*.

3.2 Ambiente

Il sistema Ambiente (nel codice, *Environment* o *Simulated Environment*), invece, ha la funzione di **modellare e virtualizzare la casa** in cui Pressman colloca SafeHome. Nel libro, però, non viene fatta mai menzione della struttura fisica dell'abitazione, perché l'autore utilizza una semplice astrazione per estendere il proprio caso di studio ad un esempio, appunto, astratto, selezionando come possibile target una qualsiasi casa, ufficio o luogo che l'ipotetico utente desidera rendere sicuro.
Nel sistema Ambiente c'è invece bisogno di questa concretizzazione e viene così determinata una pianta della casa che si vuole interfacciare con openHAB. La scelta delle stanze può essere fatta indipendentemente agendo sui file di configurazione del sistema Ambiente stesso, tramite un processo che verrà descritto più avanti, od agendo sulla piattaforma di openHAB.
È di fondamentale importanza per il corretto funzionamento del sistema la corrispondenza 1 : 1 tra la configurazione del sistema Ambiente e la configurazione di openHAB, in termini di stanze, con i relativi nomi, e in termini di dispositivi collocati al loro interno.
Nessun requisito del caso di studio viene in ogni caso soddisfatto dal design di questo sistema.

Ogni locale dell'abitazione modellata possiede un valore per ciascuno dei tipi di dato rilevabili dal sistema Sensori. La precisione di questi dati però è di molto superiore a quella visualizzabile nell'altro sistema, e infatti il dato posseduto dal sistema Ambiente viene considerato, ai fini della simulazione, un dato *reale*. Per questi motivi, ad esempio, una stanza potrà avere un valore di temperatura pari a 20.256180 °C, ma un qualunque sensore non potrà misurare il dato con tale precisione e dovrà effettuare un'approssimazione prima di emettere il proprio output di temperatura.
Ambiente è **in costante ascolto di richieste** provenienti dall'esterno tramite un'interfaccia: quando si accorgerà della presenza di una richiesta **di misurazione**, il sistema invierà tramite la stessa il dato corretto. Nella richiesta viene specificato il codice identificativo del sensore interessato al dato, e viene controllata la 'pianta della casa' specificata nei file testuali; all'interno di questi, tra le varie informazioni, è contenuta per ogni locale una lista degli identificativi dei sensori che vi sono stati installati. Verrà quindi controllata tale lista allo scopo di determinare la corretta stanza alla quale si sta facendo richiesta. A titolo di esempio, l'elenco dei sensori installati nella stanza *Bathroom* presente nella versione dimostrativa del sistema SafeHome è mostrato nella *sezione 6.1* in *Figura 31*.

All'utente viene fornita la possibilità di **controllare l'andamento di un dato**: tramite l'uso dell'interfaccia infatti lui o lei potrà decidere se, quando e di quanto variare un qualsiasi tipo di dato, specificando oltre al valore desiderato, anche **la velocità di variazione** fra tre quantità.
Il sistema gestirà automaticamente la modifica dei valori, ma non ci sono leggi fisiche coinvolte: la **variazione è percentuale** ed avviene **periodicamente** fino a che non si sarà raggiunto il valore target inizialmente inserito in input.
Di nuovo, l'unica eccezione è il rilevatore di movimento, che invece avvertirà direttamente il sensore presente in quell'area del cambiamento avvenuto quando si avrà una variazione. Durante il processo di alterazione del dato è necessario specificare, oltre all'intensità del movimento (in una scala da 0 ad 1), la durata dello stesso, oltre la quale ne verrà automaticamente comunicata l'assenza al sensore interessato.
Come già detto è importante che la configurazione locale del sistema coincida con quella remota di openHAB, altrimenti non è garantito l'aggiornamento degli items contenuti nella sitemap dell'app.
L'utente che non ha correttamente svolto questo lavoro potrà avviare soltanto il funzionamento locale del sistema SafeHome.

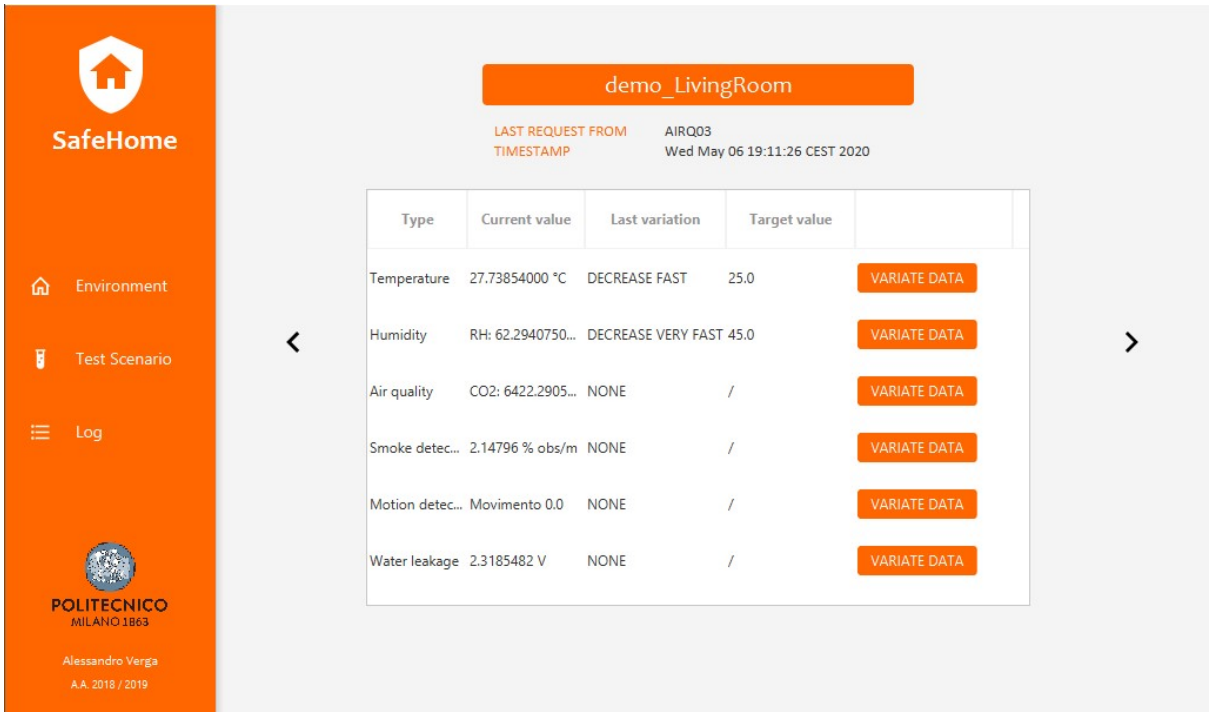

Figura 7: la home del sistema *Ambiente*, che mostra il riepilogo dei dati presenti nelle stanze.

Ambiente ha inoltre limitate funzionalità di visualizzazione dei dati, visibile in *Figura 7*: questa scelta di design è stata fatta per poter permettere un doppio controllo sul funzionamento di un sensore - cioè, che stia per esempio fornendo ad openHAB i dati corretti e non abbia errori interni - analogamente a come si potrebbe fare nella realtà misurando qualitativamente con il proprio corpo i dati di interesse.

Anche Ambiente presenta funzionalità di Log, in cui viene specificato momento e richiedente di ogni dato, richiesta di variazione di un parametro e successivi andamenti intermedi, nonché eventuali messaggi d'errore.

La variazione dei dati, nel dettaglio:

Senza scendere troppo nell'analisi dell'interfaccia grafica, meglio descritta in *5.1*, l'utente può manipolare l'andamento di ciascuno dei dati in una stanza tramite un pannello di controllo.
Il procedimento funziona nel modo seguente:

- l'utente **immette i parametri** *tipo di variazione, velocità di variazione* e *valore obiettivo*, che devono essere verificati sensati dal sistema (per esempio, non si può incrementare un dato fornendo un valore obiettivo minore del valore corrente)
- il sistema **ottiene**, per il tipo di dato da variare, **il *periodo di variazione***, cioè 'ogni quanto avviene una variazione intermedia', misurato in secondi, i cui valori sono mostrati in *Figura 9*.
- il sistema automaticamente dopo il suddetto periodo **effettua una variazione intermedia** aumentando o diminuendo il valore del dato di una quantità **percentuale** standard e definita dalla *velocità di variazione* immessa in input
- viene effettuato un controllo sul nuovo valore raggiunto dal dato contro il parametro *valore obiettivo*: se esso dà esito positivo, la variazione viene terminata, altrimenti il procedimento ritorna

al punto precedente nel quale dopo il periodo di variazione verrà effettuata una successiva modifica e un successivo **controllo di terminazione**.

Nel caso l'utente desideri virtualizzare un movimento, il procedimento salterà il punto 3, effettuando immediatamente la variazione desiderata (il movimento è considerato infatti istantaneo); deve però immettere un parametro extra, la *durata del movimento*, dopo la quale il sistema rimetterà automaticamente il dato a zero.

Per quanto riguarda i parametri, *tipo di variazione* si distingue in *incremento*, *decremento* e *trigger* (prerogativa del dato Movimento). Nella GUI e nel codice, in realtà, incremento e decremento vengono fusi alle tre *velocità di variazione* (*normale, rapida, molto rapida*) per distinguere univocamente ognuna delle variazioni, creando così sei possibili valori di *velocità di variazione*, mostrate in *Figura 8*, con i valori percentuali utilizzati per il cambiamento del valore, che utilizza la seguente formula:

$$V_1 = V_0 + (\frac{v}{100} * V_0 * F)$$

dove V0 è il valore al tempo di inizio variazione e V1 il nuovo valore, v la *velocità di variazione* e F il *Fattore Correttivo*, parametro che serve a modellare il fatto che variazioni di presenza di fumo ed allagamento sono solitamente molto rapide. Per tutti gli altri tipi di dato, F = 1.

Variazione	Valore dell'enum associato nel codice	Velocità di variazione [%]
decremento molto rapido	DECREASE_FASTEST	- 2.25 %
decremento rapido	DECREASE_FASTER	- 1.50 %
decremento normale	DECREASE	- 1.25 %
incremento normale	INCREASE	+ 1.25 %
incremento rapido	INCREASE_FASTER	+ 1.50 %
incremento molto rapido	INCREASE_FASTEST	+ 2.25 %
trigger	TRIGGER	istantanea

Figura 8: le velocità di variazione percentuali associate ad ogni valore.

I valori in *Figura 8* sono stati assunti relativamente alti allo scopo di permettere una simulazione rapida della variazione di un dato: questo ha comportato però una perdita di realismo nel modello.

Ad esempio, per una variazione *incremento rapido* di temperatura da 20 °C a 25 °C, il primo valore intermedio, dopo soli 2.5 secondi (il periodo di variazione preso dalla *Figura 9*), sarà uguale a

$$V_1 = 20 + (\frac{1.5}{100} * 20 * 1) = 20.3°C$$

La variazione come detto si ferma non appena viene rilevato il primo valore superiore (o inferiore, decrementando) al *valore obiettivo* ma, data la natura percentuale di ogni modifica intermedia, il valore finale sarà **all'interno di un intorno positivo dell'obiettivo** (o negativo, decrementando); ad esempio, per la stessa variazione indicata sopra, è molto probabile che il valore finale non sarà *esattamente* 25°C, bensì qualcosa come 25.1278 °C o 25.0457 °C.

Tipo di dato	Periodo di variazione [s]	Fattore correttivo F	Rappresentazione testuale (esempio)
Temperatura	2.5 s	1	20.2 °C
Umidità	2.5 s	1	RH: 43.1 %
Qualità dell'aria	2.5 s	1	CO2: 3280 ppm - CO: 15 ppm - PM 2.5: 31 ppm - TVOC: 250 ppm
Presenza di fumo	1.0 s	3	1.45 % obs/m
Presenza di movimento	/	/	Movimento 0.65
Presenza di allagamento	1.5 s	1.2	2.24 V

Figura 9: periodi di variazione e fattore correttivo associati ai tipi di dato, nonché la loro rappresentazione testuale visibile nell'interfaccia grafica del sistema nelle due app locali.

In *Figura 9* inoltre ad ogni tipo di dato viene associata la rappresentazione testuale che è possibile visualizzare in tutte e due le interfacce grafiche locali del sistema SafeHome - ad esempio in Sensori, come si mostrerà anche in *Figura 27* nell'ambito della descrizione della UI, o in Ambiente, come parzialmente visibile in *Figura 7*. L'interfaccia remota di openHAB, invece, come indicato a breve nella sua descrizione e rappresentata in *Figura 13*, fa uso di testi più semplici, in cui viene estratto soltanto il dato di interesse e collocato all'interno dei vari campi.

3.3 La comunicazione interna: PubNub

I due sistemi sopra descritti sono in costante comunicazione tramite il servizio di messaggistica offerto da PubNub [45], una piattaforma di comunicazione realtime online la cui SDK (Software Development Kit) è disponibile per Java, che funge da intermediario.
Questa comunicazione, che è indipendente dall'attività dell'interfaccia remota di openHAB, è finalizzata allo scambio di informazioni riguardanti le misurazioni. Periodicamente, ogni sensore richiede una misura attraverso la propria interfaccia a questo servizio e l'ambiente risponderà con il dato richiesto sempre utilizzando PubNub. Nessuno dei due sistemi effettua un controllo sull'identità del mittente del messaggio, perché entrambi sono semplicemente in continuo ascolto su uno dei due canali di comunicazione; l'altro viene utilizzato per le risposte.

Tra i vari servizi che la piattaforma offre ce n'è appunto uno che permette questo tipo di comunicazione in tempo reale, dettol *Pub/Sub* (che sta per *Publisher / Subscriber*). Tale servizio implementa un pattern architetturale noto in cui un'entità, detta *Publisher*, pubblica un certo tipo di contenuti ai quali altre entità, dette *Subscriber*, sono per così dire "abbonate". Per la precisione, il tipo di contenuti che SafeHome utilizza è rappresentato da oggetti JSON che includono tutte le informazioni necessarie per il funzionamento delle richieste e delle risposte di misurazione.

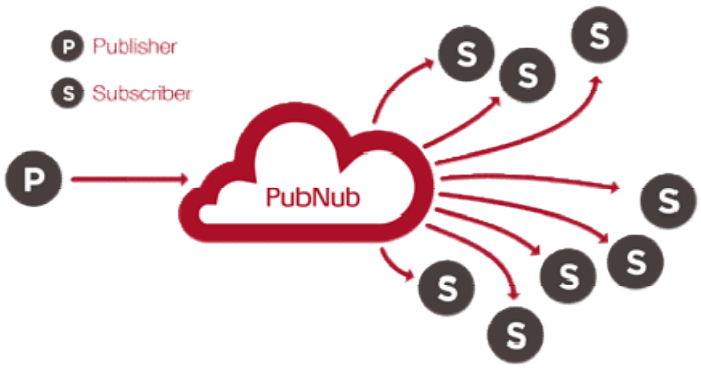

Figura 10: il funzionamento del servizio Pub/Sub offerto da PubNub [45].

Il servizio di PubNub è offerto tramite una serie di API, delle quali SafeHome possiede una coppia di chiavi senza le quali non potrebbe pubblicare né ascoltare su un determinato canale. Il canale su cui vengono pubblicate le richieste di misurazione è internamente nominato come "*SH_SE_MREQ*", mentre quello dove transitano le risposte provenienti dall'Ambiente è detto "*SH_SE_MRESP*"; è sufficiente per una delle due applicazioni 'abbonarsi' al canale d'interesse e pubblicare sull'altro.

3.4 L'interfaccia remota: openHAB

L'interfaccia remota implementata grazie alla piattaforma di openHAB è fondamentale per soddisfare tutte e quattro le funzionalità principali desiderate da Pressman nel caso di studio in esame, realizzando così i restanti requisiti funzionali (1, 2 3, 5, 6, 7 e 9).
Il sottosistema serve infatti sia allo scopo di visualizzare da remoto tutti i dati in output ai sensori, sia di allarmare l'utente in caso di situazioni indesiderate, sia di controllo dell'intera smart home tramite un apposito pannello (implementato tramite una sitemap).
Nello specifico, le situazioni indesiderate sono in totale undici, e sono implementate come valori di soglia massimi o minimi. Di queste, otto sono completamente configurabili dall'utente, a patto che lui o lei modifichi un apposito file (il file con estensione *.rules* chiamato come la propria sitemap) contenuto in una precisa cartella di installazione dell'applicazione server offerta dalla piattaforma. Più dettagli su come modificare tali file nella *sezione 6.1*.
Uniche eccezioni non configurabili sono quelle dipendenti dai rilevatori di movimento, fumo ed allagamento, poiché i loro valori di soglia sono intrinsecamente legati ai sensori stessi, come nella realtà: due diversi dispositivi reperibili sul mercato avranno quasi certamente valori di rilevazione differenti; i valori che ne determinano l'indesiderabilità sono quindi legati al codice del sistema Sensori.

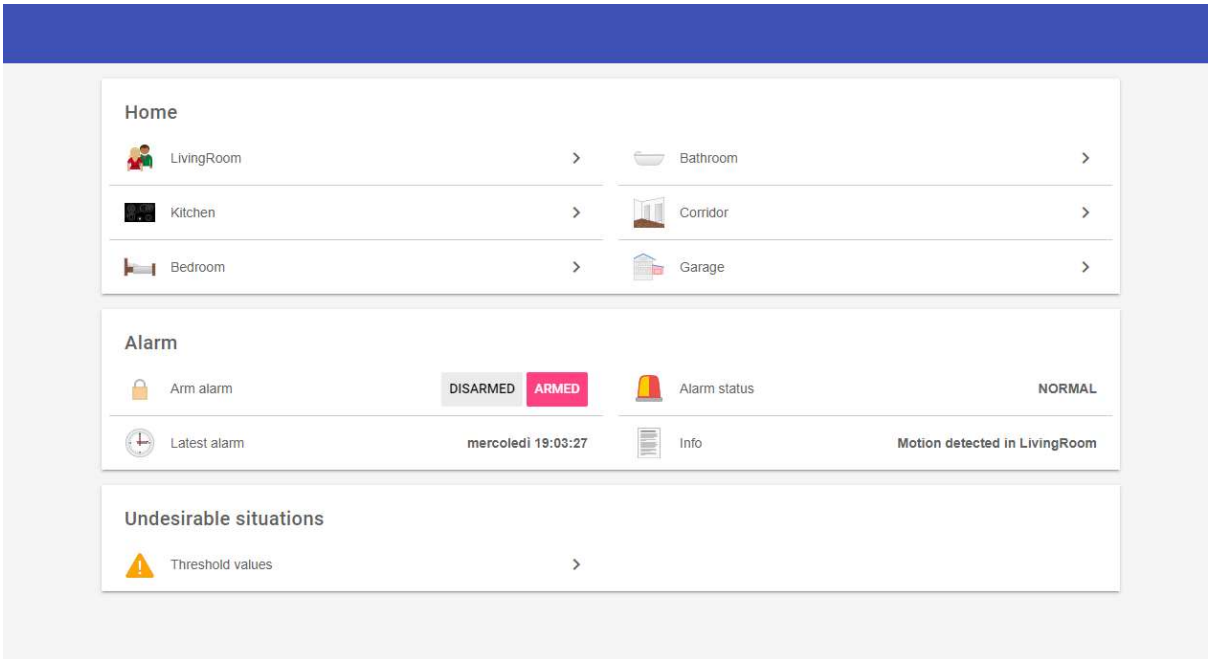

Figura 11: schermata principale dell'interfaccia remota del sistema SafeHome.

Nel caso un dato superi tali valori limite l'allarme scatterà, visivamente modificando il proprio stato nella sitemap, ed effettuando una telefonata all'utente. Automaticamente avviata dal sistema, l'utente riceverà **la chiamata tramite Telegram**, il noto servizio di messaggistica istantanea. Alla risposta, **una voce registrata** comunicherà brevemente un riepilogo dell'anomalia, in cui ne verranno specificati causa, stanza e se possibile valore di errore. La telefonata è implementata utilizzando il servizio web *CallMeBot* [46], offerto come API. Il numero del destinatario non è visibile, nel codice del file *.rules*, ma lo è il nome dell'account Telegram, che andrà modificato qualora si desiderasse avvertire un'altra persona.

Figura 12: logo di CallMeBot [46], il servizio utilizzato per effettuare chiamate.

Dall'interfaccia remota è possibile per l'utente inserire e disinserire l'allarme con un solo click e visualizzare un riepilogo informativo riguardo il più recente avvenimento indesiderato. Ciascun dispositivo, inoltre, può essere allarmato o meno da remoto, permettendo la creazione di vere e proprie zone di allarme distinte: ad esempio, solo i sensori posti nel locale Garage potrebbero essere desiderati attivi.

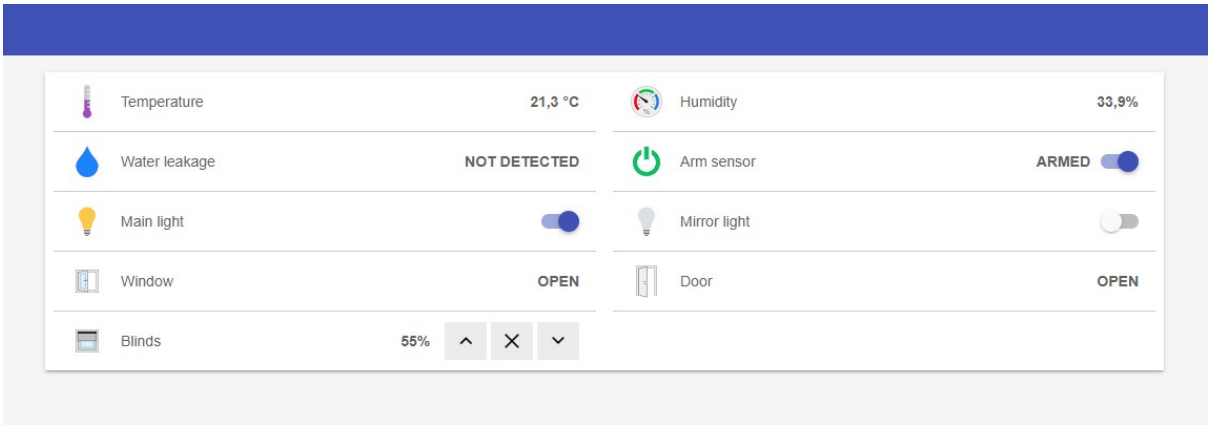

Figura 13: schermata della stanza Bathroom nell'interfaccia remota.
Oltre a visualizzare i dati in output dei sensori, è possibile attivarlo e disattivarlo con il pulsante Arm, nonché controllare lo stato di luci, finestre, porte e tapparelle.

Per lo scopo di questo lavoro di tesi, il sistema SafeHome non è stato installato su alcun server online, e pertanto non è accessibile da remoto, come invece richiederebbe la funzionalità G4: una release della piattaforma è stata infatti **avviata in locale**, per motivi di indisponibilità di un reale server.
OpenHAB può essere installato infatti su vari server dedicati, e sono supportati vari sistemi operativi come Windows, Linux, macOS, ma anche ad esempio dispositivi quali Raspberry Pi.
Le funzionalità offerte dall'interfaccia, che sia locale o remota, **non cambiano in alcun modo**, a patto che l'installazione sia eseguita correttamente [47] .
Si continuerà ad usare la terminologia 'interfaccia remota' come si è fatto finora per indicare una qualunque release di openHAB, che sia effettivamente online oppure no.

Ultima funzionalità dell'interfaccia degna di nota è il modello del **pannello di controllo** per la smart home stessa. Il suo funzionamento è completamente distaccato dai sistemi Ambiente e Sensori, e serve unicamente **per dimostrare un possibile caso d'uso** dell'applicazione. Tramite l'uso della sitemap, nella quale è incluso il pannello stesso, l'utente potrà ad esempio accendere e spegnere luci o elettrodomestici collegati, oppure aprire e chiudere finestre e porte. È possibile visualizzare questo caso d'uso all'opera solo se si sceglie di aprire la sitemap *SafeHome Use Case* dalla schermata iniziale.

L'aggiornamento dei valori di temperatura, umidità, eccetera è gestito automaticamente dal sistema SafeHome tramite una comunicazione diretta e a senso unico tra Sensori e runtime di openHAB. Tale comunicazione è implementata attraverso l'uso delle API che la web app espone, e viene ripetuta ad ogni nuova misurazione. Si rimanda alla *sezione* 5.2 per una completa analisi di questo processo.

Il binding di openHAB

Ogni interfaccia remota di openHAB per funzionare correttamente necessita dell'inserimento di alcune specifiche *thing* legate ai dispositivi installati nella propria smart home, e SafeHome non è da meno: ha bisogno infatti che nella configurazione remota si dichiari l'utilizzo di una tra le *thing* che sono specificate

dal **binding *safehome_se***. Come già accennato, i *binding* sono l'analogia di openHAB dei driver dei dispositivi, cioè dei software che permettono la comunicazione device / runtime.
La soluzione proposta fa uso di tale binding, anche se le sue funzioni si limitano ad una **specificazione** formale **dei modelli di dispositivi** (e per estensione di sensori) supportati dal sistema SafeHome, e ad una descrizione dei concetti di thing e channel richiesti dal server di openHAB.

Modello del dispositivo	Temperatura	Umidità	Qualità dell'aria	Rilevatore di fumo	Rilevatore di movimento	Rilevatore di allagamento
MODEL T	✓					
MODEL H		✓				
MODEL A			✓			
MODEL S				✓		
MODEL M					✓	
MODEL F						✓
MODEL TH	✓	✓				
MODEL TF	✓					✓
MODEL HA		✓	✓			
MODEL SM				✓	✓	
MODEL THF	✓	✓				✓
MODEL THA	✓	✓	✓			
MODEL HAS		✓	✓	✓		
MODEL THASM	✓	✓	✓	✓	✓	
MODEL THASMF	✓	✓	✓	✓	✓	✓

Figura 14: lista dei modelli di *thing* supportati dal *binding* del sistema SafeHome.
Per ognuno si indica quali sensori sono già installati.

L'analisi del processo di installazione e modifica dei dispositivi, tramite l'editing di determinati file di configurazione, è rimandata alla *sezione 6.1*; di seguito invece è presentata una tabella comprensiva dei modelli supportati e dei loro parametri di configurazione necessari.

La tabella seguente, invece, è di interesse solo se si ispezionano i file di configurazione del runtime, nello specifico il file *.items*: si associa infatti per ogni tipo di dato supportato il nome del channel e il tipo di canale che sono dichiarati nel suddetto file; se un ipotetico utente volesse aggiungere altri dispositivi dovrebbe specificare i parametri *channel group* e *channel* nella configurazione del nuovo item (rif. *sez. 6.1* per l'analisi del procedimento di aggiunta di un dispositivo).

Tipo di dato	openHAB channel group	openHAB channel	Tipo di canale
Timestamp ultima misurazione	Tutti *	last_measurement_time	DateTime
Temperatura	temperature_sensor	curr_temperature	Number
Umidità	humidity_sensor	curr_humidity	Number
Qualità dell'aria CO2	airquality_sensor	curr_airq_co2	Number
Qualità dell'aria CO	airquality_sensor	curr_airq_co	Number
Qualità dell'aria PM 2.5	airquality_sensor	curr_airq_pm25	Number
Qualità dell'aria TVOC	airquality_sensor	curr_airq_tvoc	Number
Presenza di fumo	smoke_sensor	curr_smoke	Trigger
Presenza di movimento	motion_sensor	curr_motion	Trigger
Presenza di allagamento	flood_sensor	curr_flood	Trigger

Figura 15: i *channel* associati nel *binding* ad ogni tipo di dato supportato.

4. Architettura del sistema

L'architettura del sistema SafeHome, nei suoi sottosistemi Sensori ed Ambiente ampiamente già descritti, segue un paradigma noto come **Event-driven architecture** (in italiano, *architettura guidata dagli eventi*), nel quale i vari componenti software producono, rilevano, consumano e reagiscono a specifici cambiamenti nello stato di alcune entità, che appunto vengono definite *eventi*. Tali eventi hanno l'effetto di generare specifici *messaggi* contenenti una qualche rappresentazione di ciò che è accaduto.

In letteratura, in verità, il termine event-driven viene spesso usato erroneamente per indicare un altro pattern architetturale, più specifico, ovvero quello *Message-driven* (*guidato dai messaggi*), nel quale le comunicazioni tra i suddetti componenti avvengono usando unicamente stringhe testuali, mentre nel caso generale un messaggio può consistere in un qualsiasi tipo di codifica dell'informazione.

Il sistema SafeHome fa uso di veri e propri eventi realizzati ad-hoc, e quindi rientra tra le implementazioni del primo pattern.
Vi si distinguono alcuni componenti chiave, definiti dal pattern stesso, come i *publisher*, cioè coloro che generano eventi (a volte chiamati anche *agenti*), i *canali,* attraverso i quali i messaggi generati dagli eventi vengono scambiati e trasferiti, e i *subscriber*, ovvero le entità diciamo "abbonate" ai canali, che li osservano alla ricerca di messaggi. Tale operazione viene più tecnicamente definita *ascolto*.
Quando un subscriber si accorge della presenza di un messaggio su uno dei canali sul quale è in ascolto, egli reagisce, eseguendo un certo insieme di azioni in risposta all'evento.

Frequentemente una simile architettura viene implementata attraverso componenti che sono ignari l'uno della presenza dell'altro (o per meglio dire *loosely coupled*), ovvero un publisher non ha idea di chi sia in ascolto su un certo canale, né se ci sia effettivamente qualcuno o meno, né i subscriber si conoscono tra di loro.

In SafeHome queste astrazioni sono tutte presenti e in fase di design il sistema è stato pensato avendo in mente un'architettura event-driven, ma i ruoli di publisher e subscriber non sono così distinti come nella teoria. Così, i sensori periodicamente richiedono di effettuare una misurazione generando un evento appropriato (che informalmente per ora chiamiamo *Misurazione*) e trasmettono il messaggio generato da esso sul proprio canale, svolgendo il ruolo di publisher.
Misurazione viene ascoltato dal componente incaricato di comunicare attraverso PubNub con il sistema Ambiente, e dopo che la comunicazione sarà avvenuta e lo stesso componente avrà ricevuto la risposta, genererà un evento (per ora *Valore della Misura*) sul canale del sensore richiedente.
Come si può notare, i ruoli di publisher e subscriber sono invertiti in fase di risposta.

Vari altri componenti sono interessati all'ascolto di tutti questi messaggi, come ad esempio l'interfaccia grafica che dovrà visualizzarli all'utente, o il log che dovrà tenere traccia di ogni evento.

Allo stesso modo, anche il sistema Ambiente è gestito dagli eventi, e il modulo comunicante attraverso PubNub ricoprirà prima il ruolo di publisher, per richiedere il dato reale posseduto nelle varie stanze, e poi di subscriber, per riceverlo e inoltrarlo attraverso Internet all'altro sistema.

I moduli che formano i due sottosistemi sono tra loro *loosely coupled* (lett. non strettamente accoppiati) proprio per la natura del pattern architetturale stesso, ma l'astrazione è presente anche tra i sottosistemi stessi: né Ambiente né Sensori sono consci della presenza dell'altro; conoscono soltanto PubNub, l'intermediario.
Questo dà al sistema SafeHome un vantaggio non indifferente, cioè quello di poter funzionare a prescindere da *chi comunica con l'intermediario.*
In futuro, uno dei sottosistemi di SafeHome potrà essere riutilizzato (con un riadattamento minimo, o nullo) per un caso di studio reale, ad esempio sostituendo Ambiente con un'applicazione in grado di fornire *dati reali*, provenienti ad esempio da vere stanze di una casa - più dettagli nella *sezione 9, Lavoro Futuro*.

4.1 Modelli UML

Il linguaggio UML (*Unified Modeling Language*) è uno standard per la visualizzazione del design di un sistema attraverso specifici modelli, ciascuno con le proprie regole ed obiettivi, che vengono di seguito presentati allo scopo di chiarificare alcuni aspetti salienti dell'architettura del sistema realizzato.
La lettura di questa sezione, nonostante i modelli siano quasi universalmente comprensibili, è consigliata a chi possiede conoscenze tecniche di modellazione UML.

Per la precisione, si riportano tre *Component diagram*, uno più generale che descrive i sottosistemi citati, ed uno per ciascun sottosistema - Sensori ed Ambiente. Per questi due, viene presentato un *Activity diagram*, che meglio esplicita il funzionamento di base, due *Sequence diagram* a testa, per descrivere nel dettaglio il passaggio di messaggi in determinati casi di utilizzo.
Inoltre, uno *State machine diagram* descrive lo stato di allarme che può assumere SafeHome e un altro descrive l'insieme di stati raggiungibili da ogni sensore.

I diagrammi contengono i vocaboli formali in lingua inglese (ad esempio *requires* o *component*). I nomi informali di componenti, stati è invece in italiano per coerenza descrittiva. Quando invece si fa esplicito riferimento ad algoritmi, funzioni o nomi di variabili è stato mantenuto l'inglese, come nel codice.

Component diagrams

La *Figura 16* rappresenta un *diagramma dei componenti* di Safehome, che come già ampiamente descritto, si compone di due applicazioni indipendenti, che possono essere considerate a tutti gli effetti dei sottosistemi. Sensori ed Ambiente, questi due, comunicano direttamente con il servizio web offerto da PubNub, il quale agevola e permette indirettamente lo scambio di dati tra di essi.
Solo l'applicazione Sensori comunica con openHAB e la sua UI, la quale per ricevere correttamente aggiornamenti ai propri dati ha bisogno dell'installazione del binding safehome_se.

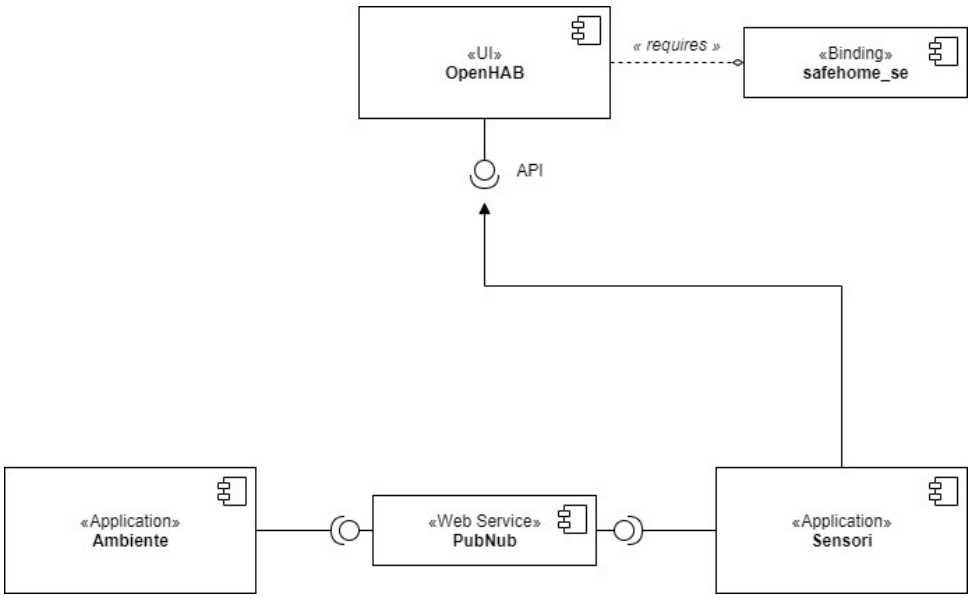

Figura 16: componenti del sistema SafeHome.

Le due figure seguenti, invece, descrivono i componenti delle due applicazioni separatamente.
Entrambe derivano da un'azione iniziale dell'utente, unico attore attivo, che avvia il sistema attraverso la GUI (Interfaccia Grafica Utente). PubNub e openHAB sono attori passivi con cui i componenti interfaccia comunicano periodicamente.

Si può notare che in entrambe è presente il sistema di Publish / Subscribe interno che implementa il pattern architetturale guidato dagli eventi.
Le frecce indicano il verso della comunicazione tramite tali eventi, che il più delle volte è bidirezionale.
Viene esplicitata anche la dipendenza della GUI di entrambi i software dai propri file di configurazione, che vengono letti all'avvio. Le frecce tratteggiate che partono dal componente *File di configurazione* indicano che i dati lì contenuti riguardano i componenti destinazione della relazione (*Dispositivi* e *Sensori*; *Stanze* per il sistema Ambiente).
Inoltre nel primo diagramma è visibile la relazione di composizione 1 : N tra Dispositivi e Sensori.

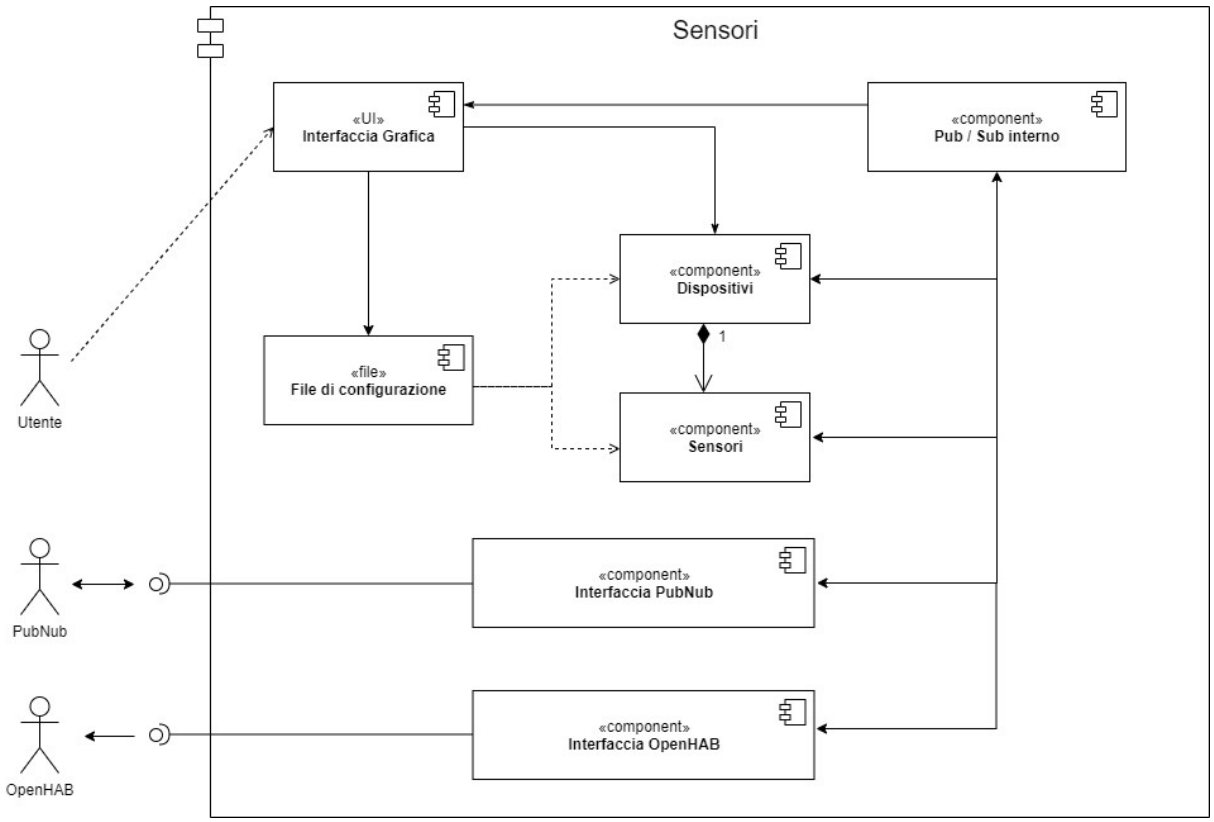

Figura 17: componenti del (sotto) sistema Sensori.

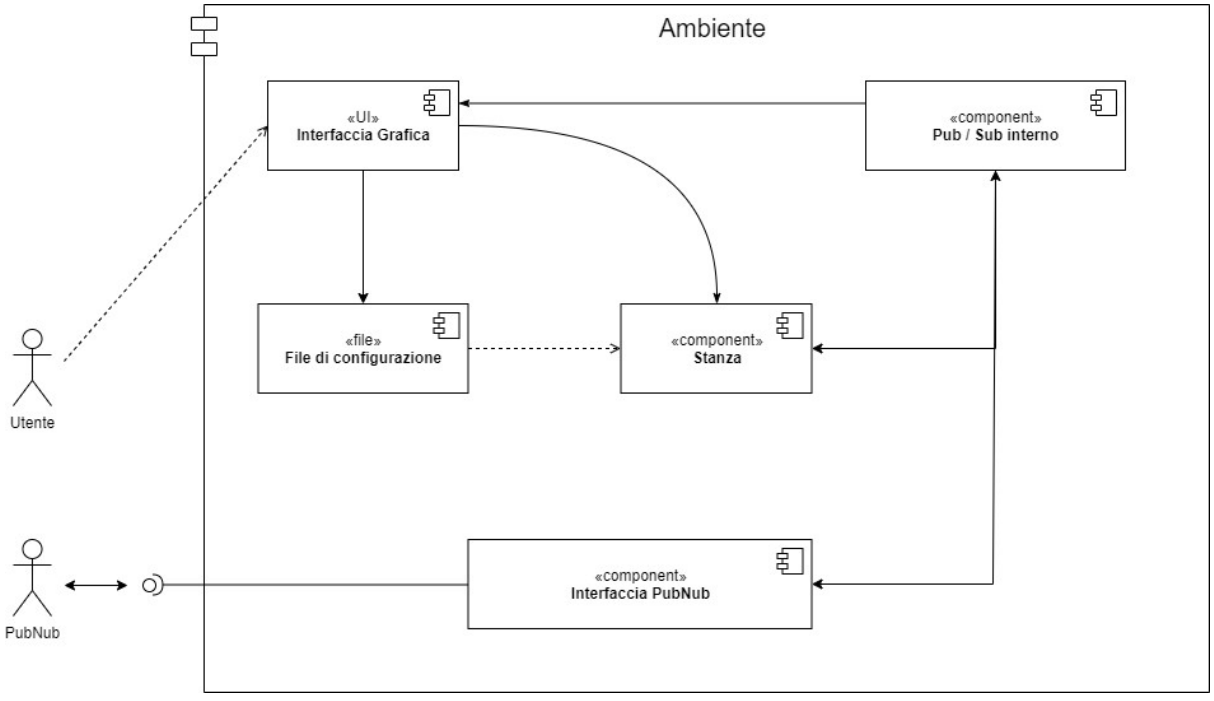

Figura 18: componenti del (sotto) sistema Ambiente.

State machine diagrams

I diagrammi in *Figura 19 e 20*, noti come *macchine a stati finiti*, esplicitano appunto rispettivamente stati e transizioni del sistema SafeHome e del componente che implementa i sensori.
Reset time è un parametro configurabile dall'utente che modella appunto il periodo di tempo durante il quale il sistema permane nello stato *InAllarme*.

Figura 19: macchina a stati del sistema di allarme implementato con SafeHome.

Nel secondo diagramma è visibile lo stato *Failure*, ma esso è raggiungibile soltanto nello scenario di test '*Fallimento di un sensore*' alla quale transizione viene associata una probabilità di fallimento che l'utente che effettua il test immette in input. In una esecuzione normale del sistema, tale stato è raggiungibile soltanto modificando il parametro apposito nei file di configurazione dell'applicazione.

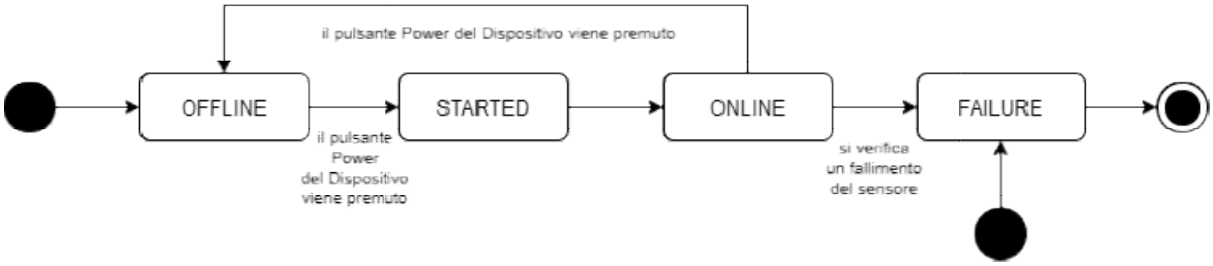

Figura 20: macchina a stati del funzionamento di un sensore.
Lo stato FAILURE è raggiungibile soltanto nello scenario di test *Fallimento di un sensore*.

Activity diagrams

I successivi diagrammi vengono utilizzati per esplicitare il loop di funzionamento normale delle due applicazioni, nel quale si analizza a sinistra il ciclo periodico di misurazione automatica di un sensore, e a destra gli avvenimenti in risposta a tali richieste di misura all'interno del sistema Ambiente.

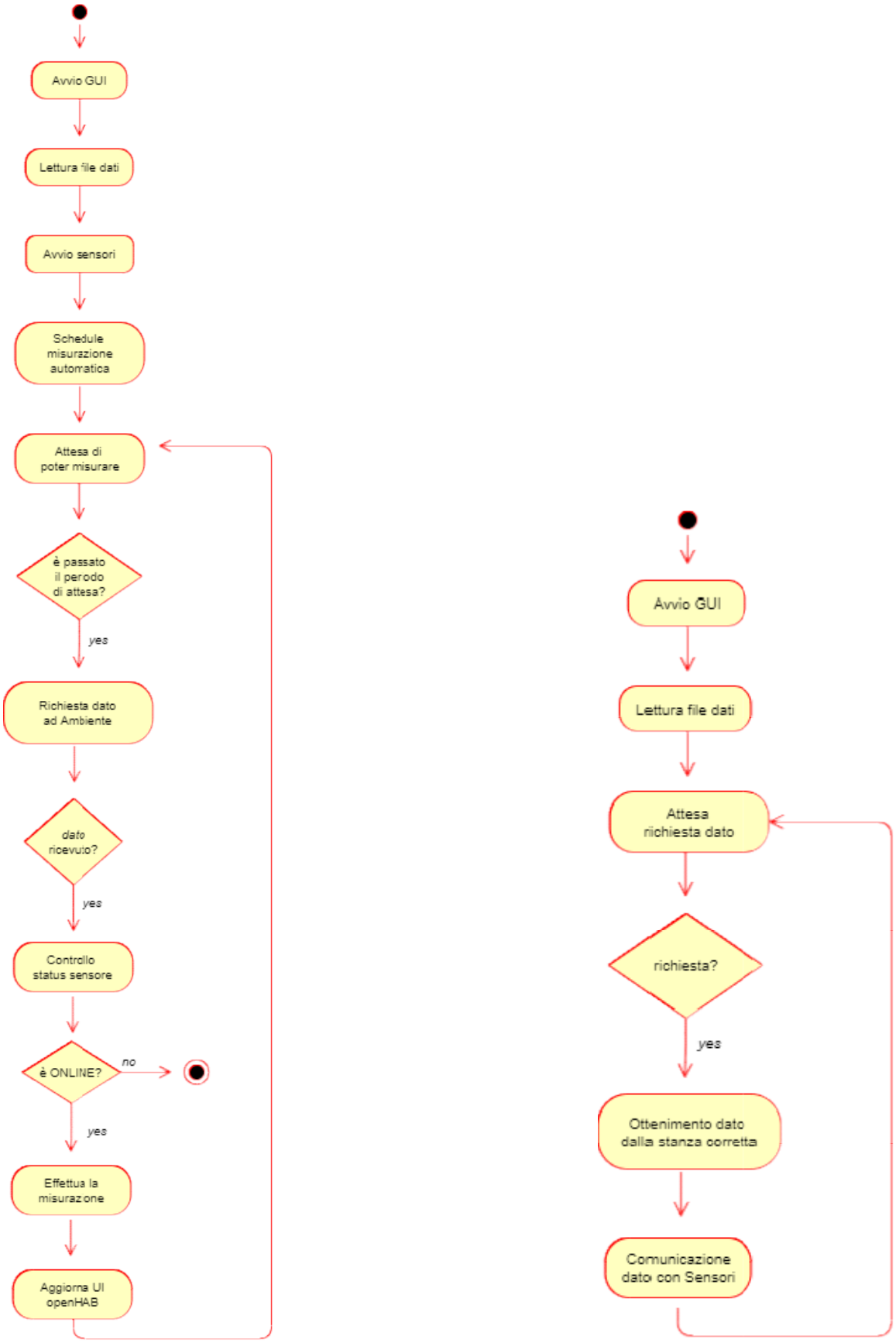

Figura 21: il funzionamento di ogni sensore (a sinistra) e del loop del sistema Ambiente (a destra).

Sequence diagrams

I sequence diagrams seguenti vengono utilizzati allo scopo di chiarificare l'architettura guidata dagli eventi di SafeHome e di esplicitare momento di creazione, mittente e destinatario dei più importanti eventi e messaggi utilizzati dal sistema di Publish / Subscribe interno.
Vengono inoltre mostrate nelle figure le chiamate ai metodi principali che causano la comunicazione tra i vari componenti.

Il primo modello riguarda **l'avvio di Sensori**.
Dopo che l'utente ha avviato l'interfaccia grafica, è possibile notare che le interfacce di openHAB e PubNub vengono inizializzate, dopodiché vengono letti i file dati dei dispositivi (il cui contenuto è analizzato nella sezione *6.1*). Ogni dispositivo viene quindi avviato dopo che la fase di lettura è terminata, e ognuno di questi si occuperà automaticamente di avviare i propri sensori, e di effettuare lo *scheduling* dei *thread* di misurazione automatica, che sono in pratica dei comandi periodici con periodo uguale al valore di *refresh time* di ogni sensore (anch'esso contenuto nei file dati).

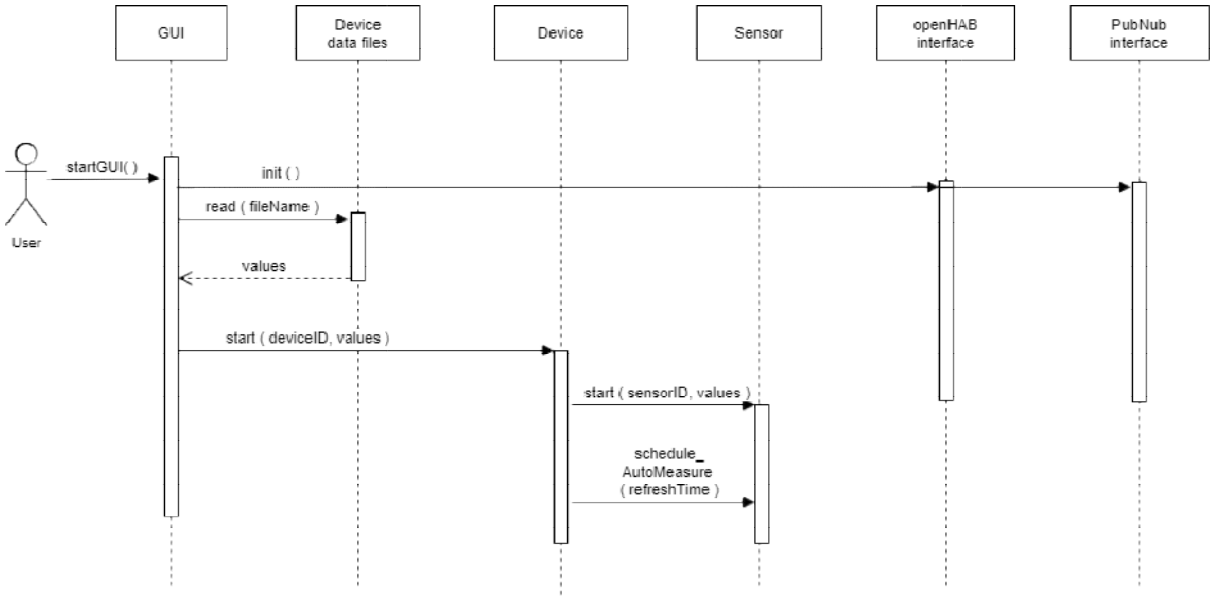

Figura 22: l'avvio del sistema Sensori da parte dell'utente.

Il secondo diagramma mostra nel dettaglio la sequenza di messaggi scambiati tra UI, sensori e interfacce durante il **processo periodico di misurazione automatica**.
È possibile notare che ad ogni *refresh time* ogni sensore solleva un evento mandando un messaggio *MEASURE_AUTO*, specificando il proprio ID univoco, all'interfaccia di PubNub. L'intermediario comunica tramite il proprio servizio *Pub / Sub* con il database sul canale *REQUEST*, e poco dopo riceve un messaggio sul canale *RESPONSE*. A questo punto l'interfaccia solleva un evento di tipo MEASURE_RESPONSE nel cui messaggio sono contenuti i dati *ID sensore* e *valore del dato richiesto*. Il messaggio viene ricevuto dal sensore identificato da quell'ID sul proprio canale: dopo aver effettuato la misurazione, solleva l'evento *MEASURE_DONE* con il valore del *dato dopo la misura*. L'evento viene ricevuto indipendentemente dalla GUI e dall'interfaccia di comunicazione con openHAB, le quali effettueranno l'aggiornamento del valore mostrato rispettivamente nell'interfaccia locale e remota (questa tramite la chiamata alle API esposte).

Non è stato mostrato per semplicità **il componente Log** che di default riceve tutti i messaggi su tutti i canali e si occupa separatamente di comunicare con la GUI per aggiornare la propria pagina dedicata.

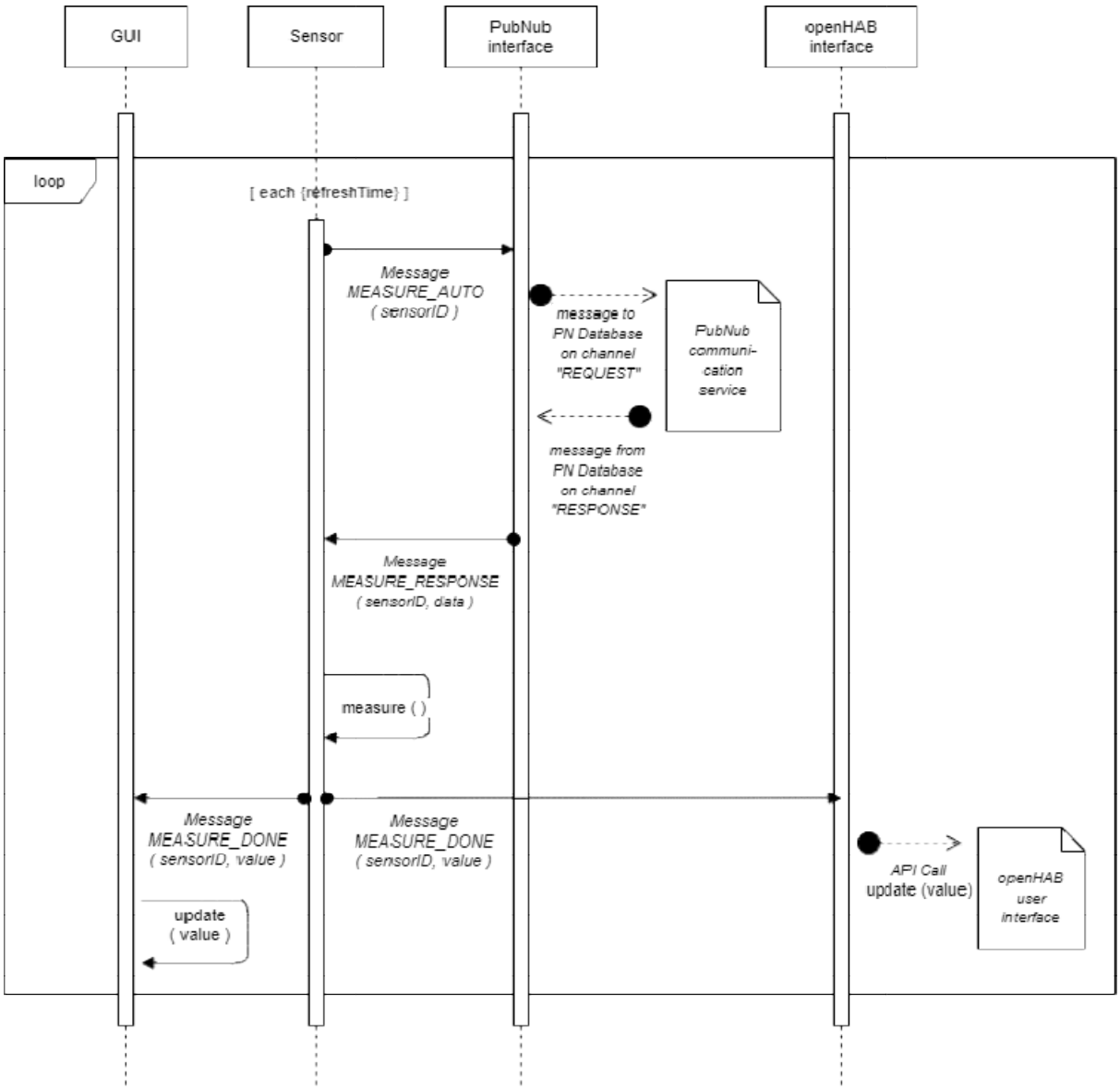

Figura 23: il processo di misurazione automatica compiuto da ogni sensore.

I sequence diagram in *Figura 24 e 25* invece si riferiscono al funzionamento di Ambiente, e aiutano a spiegare il **processo di richiesta di una misurazione** e il processo di **variazione di un dato**.

Nel primo si nota che il messaggio *MEASURE_REQUEST* viene ricevuto dalla stanza che ha al suo interno installato il sensore *sensorID* e dall'interfaccia grafica dell'app. Quest'ultima aggiorna alcuni campi informativi sull'ultima richiesta effettuata, mentre una stanza poco dopo solleva l'evento *MEASURE_RESPONSE* contenente il dato corretto, il quale viene rimandato indietro a Sensori tramite il canale *RESPONSE* di PubNub. Questo processo di verifica ogni volta che un l'interfaccia di PubNub identifica la presenza di un messaggio sul canale *REQUEST*.

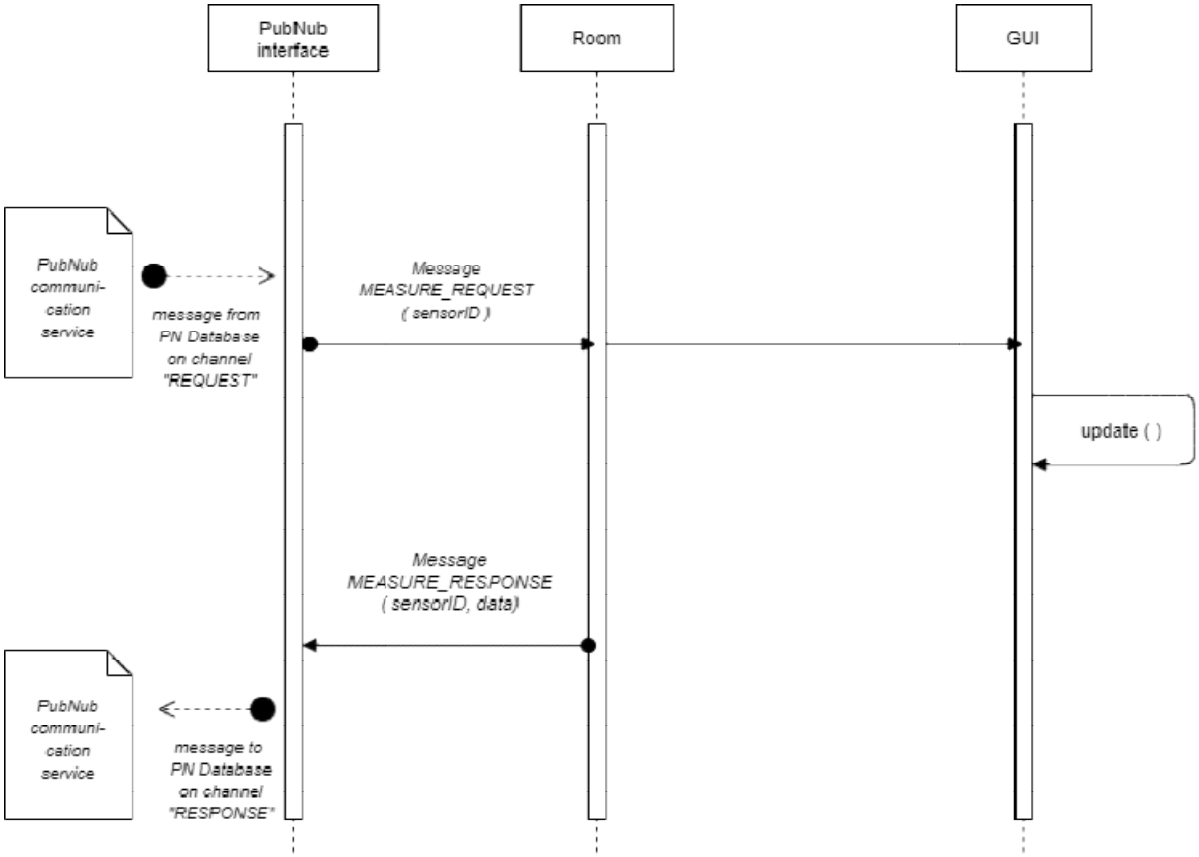

Figura 24: il processo richiesta di una misurazione da un punto di vista interno al sistema Ambiente.

Infine, l'ultimo modello riassume il processo di variazione di un dato, che viene iniziato dall'utente tramite la GUI, dove specifica i tre parametri fondamentali. Essa crea un evento di tipo *VARIATE_DATA_REQUEST* che viene ricevuto dalla stanza adatta sul proprio canale di comunicazione interno. A quel punto, l'oggetto stanza crea un *thread* periodico di periodo *jobDelay* (parametro diverso per ogni tipo di dato, rif. *sez. 3.2*, *Fig. 9* dove è chiamato *periodo di variazione*). Il thread appunto ogni periodo effettua una variazione intermedia a seconda della rapidità desiderata e a modifica avvenuta spedisce un messaggio di tipo VARIATE_DATA alla stanza corretta, che si occupa di salvare il nuovo valore nella sua lista di dati.
La GUI riceve lo stesso messaggio ed effettua la stessa modifica al fine di mostrarne visivamente il valore.

Prima di ricominciare il ciclo viene controllato che il nuovo valore sia ancora inferiore all'obiettivo voluto dall'utente, e se non è così, il thread viene terminato insieme al ciclo stesso.

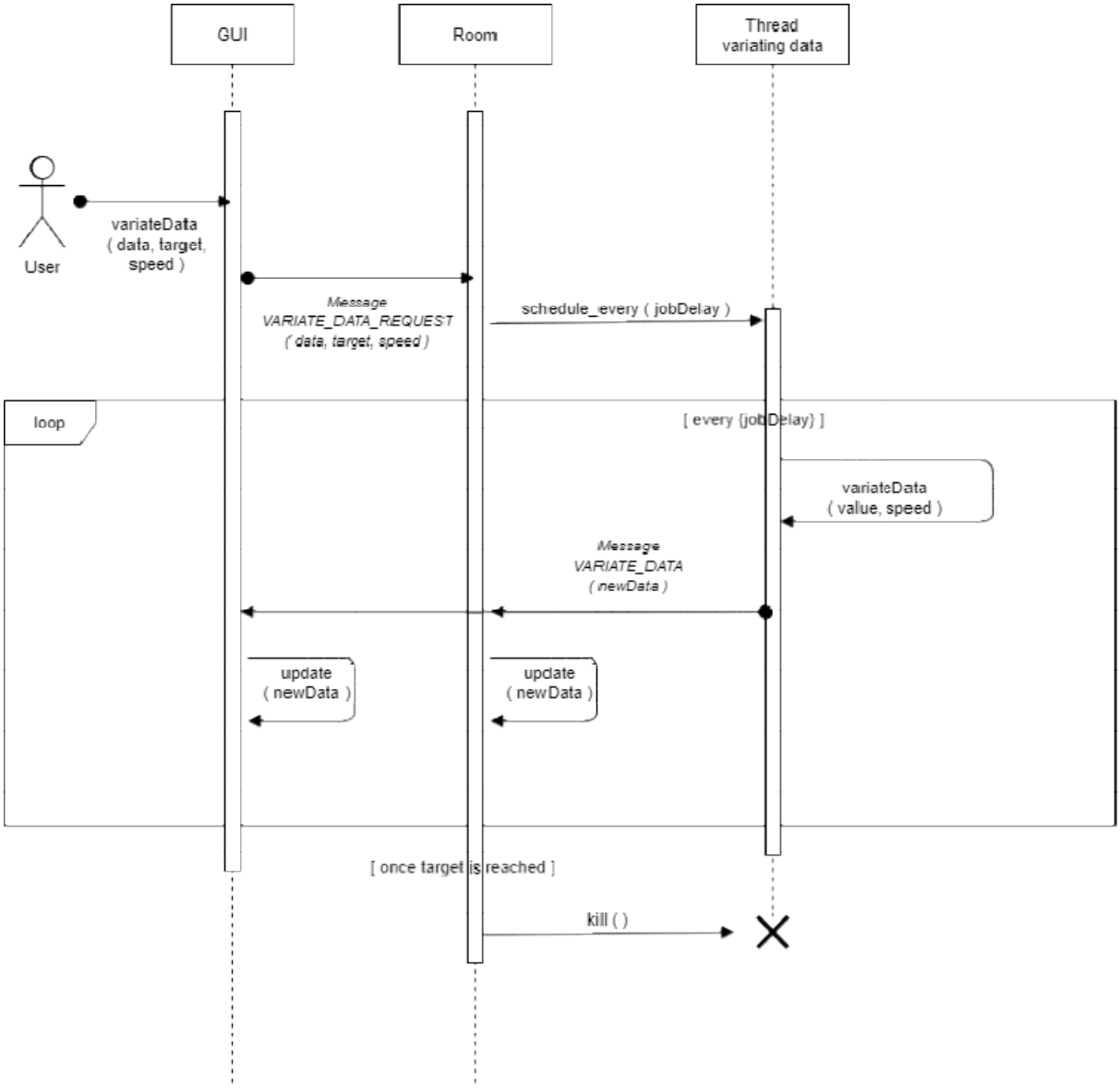

Figura 25: il processo di variazione dei dati compiuto dall'utente tramite il sistema Ambiente.

5. Design del sistema

Questa sezione viene utilizzata per descrivere il design delle interfacce utente proposte per il sistema e il loro funzionamento. Vengono spiegati nel dettaglio le pagine di visualizzazione dei dati di dispositivi e sensori, e il tool di variazione offerto dall'applicazione Ambiente; inoltre viene descritta l'interfaccia remota che l'utente troverà avviando un'istanza del server di openHAB, che comunque è già stata presentata in *Figura 11* e in *Figura 13*.
Infine, si dedica una piccola sezione all'analisi dell'unica API di openHAB che è stata utilizzata, quella che permette l'aggiornamento dello *stato* di un *item* remoto.

5.1 Analisi delle UI

Entrambe le interfacce utente locali del sistema SafeHome sono state pensate avendo come obiettivo la semplicità di utilizzo e l'immediatezza di comprensione.
All'avvio di entrambe le applicazioni, all'utente viene presentata una schermata in cui nel sistema Sensori è possibile visualizzare informazioni rilevanti i Dispositivi presenti, e in Ambiente invece presenta un riepilogo dei dati presenti nelle varie stanze della smart home.
È presente un menù di navigazione laterale, sulla sinistra, la cui prima opzione se cliccata riconduce a questa schermata principale da qualsiasi altro punto dell'applicazione.
Procedendo verso il basso si trova il tasto *Test Scenario*, che conduce ad un pannello di controllo utile per effettuare una serie di test sull'applicazione, i quali verranno discussi ampiamente nella *sezione 7*; da un punto di vista grafico, comunque, l'utente può selezionare dall'app Sensori lo scenario che vuole osservare ed immettere i parametri chiave legati allo stesso. Nella pagina è presente anche una breve guida all'uso dello strumento.
Infine, il menù presenta un tasto che se cliccato mostra il log di tutti gli eventi accaduti sin dall'avvio dell'applicativo stesso.

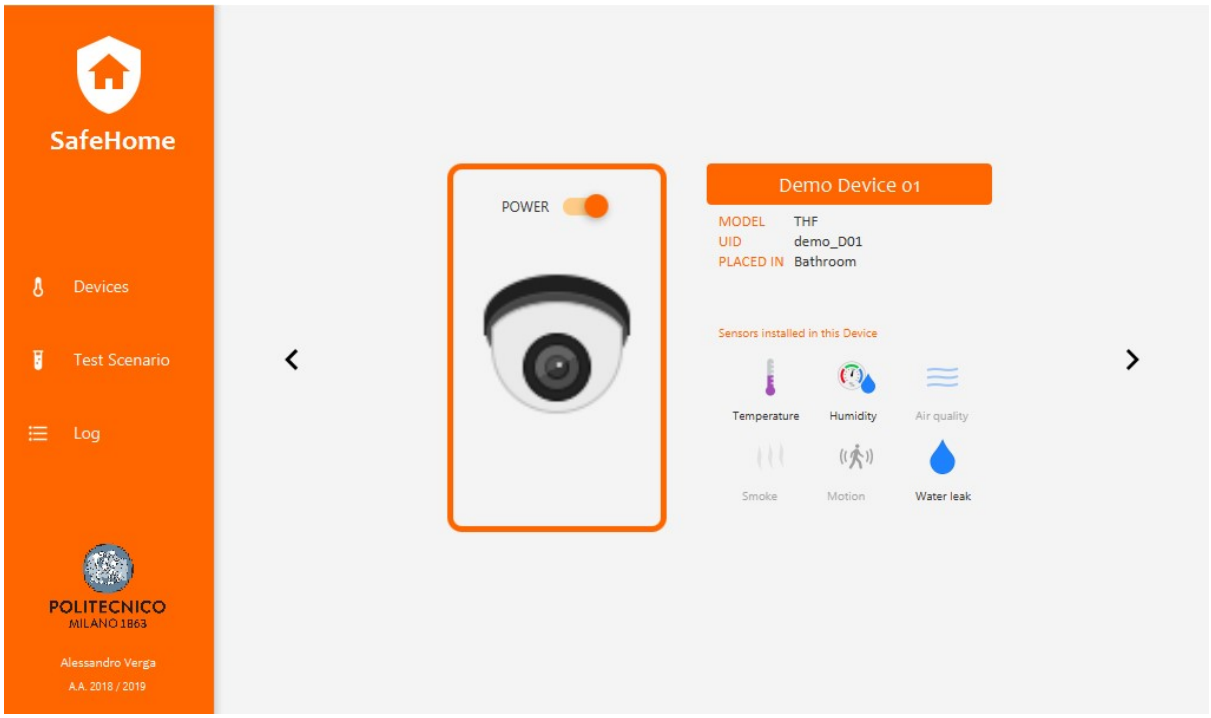

Figura 26: la pagina dell'interfaccia grafica di Sensori che mostra info sul dispositivo *Demo Device 01*.

il colore predominante, l'arancione, è stato scelto in concordanza con quello dominante presente in openHAB, visibile nella sua homepage sia in quella della Foundation creatrice del servizio. Non sembra che il colore sia comunque ufficiale, perché ad esempio pagine come le singole UI presentano pattern differenti: ad esempio, tutte le sitemap creabili e visualizzabili attraverso la Basic UI avranno una barra blu in testa alla pagina, mentre la Classic UI sarà principalmente grigia, e di nuovo Home Builder presenta un arancione più tendente verso il rosso.
Tutte le icone utilizzate invece per rappresentare i tipi di dato di SafeHome sono state prese dalla webapp stessa [48].

UI Sensori

La pagina *Devices* presenta come detto un riepilogo delle caratteristiche di un dispositivo: è possibile trovarci, oltre a nome, sigla del modello, UID (IDentificativo Univoco) e nome della stanza di installazione, un elenco dei sensori installati. Questi saranno perfettamente visibili, mentre i sensori non supportati saranno opachi.
Un click su uno dei sensori abilitati indirizzerà l'utente alla pagina contenente i dati del singolo misuratore. In alto si trova un tasto che gestisce accensione e spegnimento del dispositivo. Se premuto, comporterà lo stesso cambio di stato anche per ogni sensore contenuto nel device, che sarà possibile visualizzare tramite le suddette UI. Inoltre, tramite le frecce di navigazione, l'utente potrà scorrere i vari dispositivi installati.

Il design delle interfacce per i sensori non si discosta molto, intenzionalmente, dalla schermata principale: in alto è possibile visualizzare in grande il nome del dispositivo che contiene il rilevatore visualizzato, e subito sotto nome, tipo e stato di funzionamento. Al centro della schermata è presente un pannello

informativo simile ad un display, che conterrà i dati ricevuti da Ambiente dopo che la misurazione interna - con i possibili errori - sarà stata effettuata.
La navigazione tra sensori in uno stesso dispositivo è possibile di nuovo tramite le frecce ai lati, oppure utilizzando la dashboard sottostante. Ovviamente, dispositivi non installati non potranno essere acceduti.

Figura 27: la pagina riepilogativa dei dati del sensore di temperatura all'interno di *Demo Device 01*.

UI Ambiente

L'analoga pagina nel sistema Ambiente (mostrata in *Figura 28*) contiene le informazioni relative a stanze, invece che a dispositivi, e ai dati in esse presenti: l'utente troverà una tabella riepilogativa, in cui per ogni dato verrà riportato il valore attuale, e se è avvenuta o è in corso una variazione, obiettivo e rapidità della stessa. Nell'ultima colonna si troverà un bottone che condurrà alla pagina dalla quale è possibile alterare i dati.

Anche la schermata di variazione dei dati (*Figura 29*) è pensata per essere intuitiva e rapida da utilizzare: uno slider permette di scegliere la velocità di modifica dei dati tra le sei opzioni standard, e il valore obiettivo può essere inserito nell'apposito campo sottostante. Se il parametro da variare è la Qualità dell'aria, si potrà scegliere quale dei quattro valori alterare; se invece si vuole simulare un Movimento, ne andrà specificata la durata, oltre al valore di intensità.

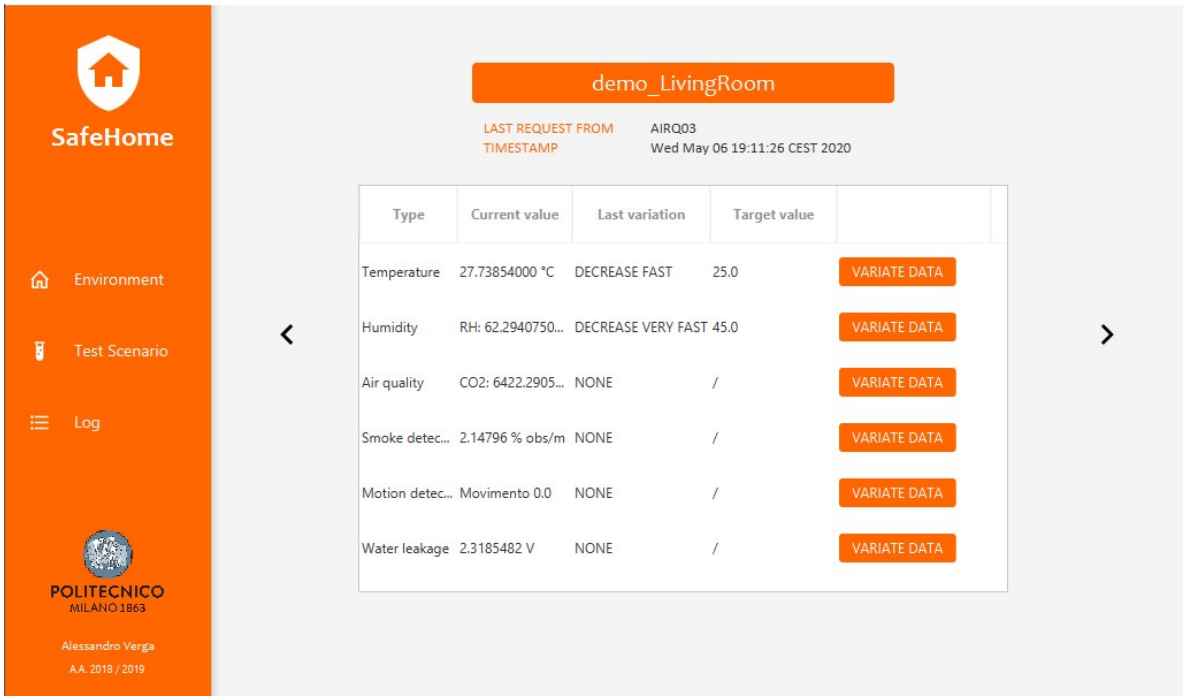

Figura 28: la pagina del sistema Ambiente in cui si visualizzano i dati della stanza *demo_LivingRoom*.

Per tutti i dati, il valore corrente (al momento di apertura della pagina) è riportato in alto.
Una volta premuto il tasto '*Start variating data*', il sistema analizzerà la correttezza dei valori inseriti (ad esempio, che non si sia inserito un valore obiettivo minore del valore corrente tentando di aumentare il dato) e comunicherà l'avvenuta variazione, o la presenza di un errore, tramite le due icone a lato.

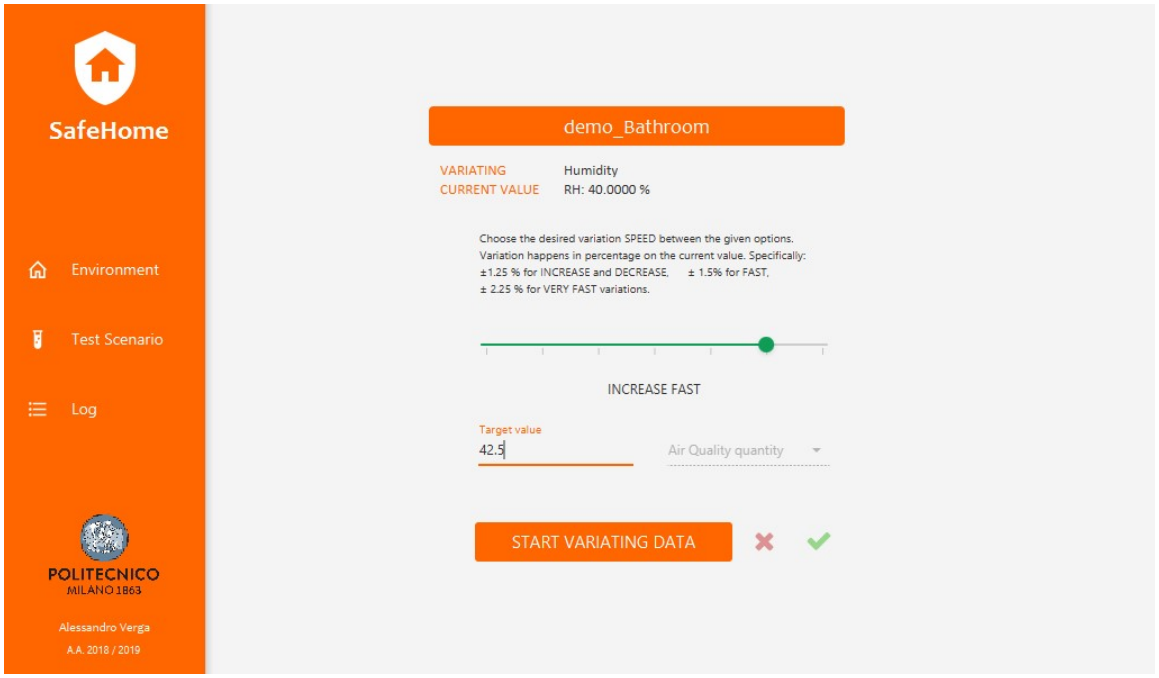

Figura 29: la UI di variazione dei dati. Qui l'ipotetico utente sta effettuando un *aumento rapido* del dato umidità nella stanza *demo_Bathroom*, dal valore corrente di 40% a quello desiderato di 42.5%.

Nel caso tutti i parametri siano corretti, l'utente potrà visualizzare l'andamento del dato nella pagina Environment.

UI openHAB

La sitemap presentata con questa demo di SafeHome (già mostrata nelle *Figure 11 e 13*) include **due stanze e due dispositivi**, che insieme possono rilevare tutti i tipi di dato supportati dal sistema. Oltre a poterne visualizzare l'output da remoto, è possibile inserire o disinserire l'allarme della casa, sia generale, sia personalizzabile specificando quali dispositivi allarmare e quali no, creando delle vere e proprie zone sicure (ad esempio, l'utente potrebbe desiderare che solo l'area Garage sia allarmata).
È anche possibile tenere traccia dei valori di soglia delle situazioni indesiderate, i quali possono essere modificati andando ad effettuare l'editing dell'apposito file *.rules*.
La sitemap dimostrativa inoltre serve come esempio di un possibile caso d'uso realistico, e perciò include elementi di controllo di luci, tapparelle e visualizzazione dello stato di apertura e chiusura di porte e finestre.

È stata anche inserita una ulteriore **sitemap dedicata agli scenari di test**, chiamata "SafeHome Demo" in cui un utente interessato a questa funzionalità può visualizzare parametri riepilogativi degli scenari supportati, come ad esempio numero di allarmi indotti da errori nelle misurazioni.

5.2 API utilizzate

SafeHome utilizza una sola delle API esposte dal servizio di openHAB allo scopo di aggiornare l'interfaccia remota con i valori output delle nuove misurazioni. La documentazione ufficiale delle API è reperibile accedendo ad una speciale cartella del server [49].
La *Figura 30* descrive l'interfaccia e i suoi codici restituiti come risposta, dei quali solo 202 corrisponde ad un successo. I parametri necessari sono nome dell'item che si vuole aggiornare e nuovo stato.

SafeHome utilizza questo metodo nei seguenti casi:

- aggiornamento del valore in output derivato da una misurazione
- attivazione di uno scenario di test, con aggiornamento dell'item informativo
- visualizzazione dei risultati di uno scenario di test, con aggiornamento di tutti gli item interessati

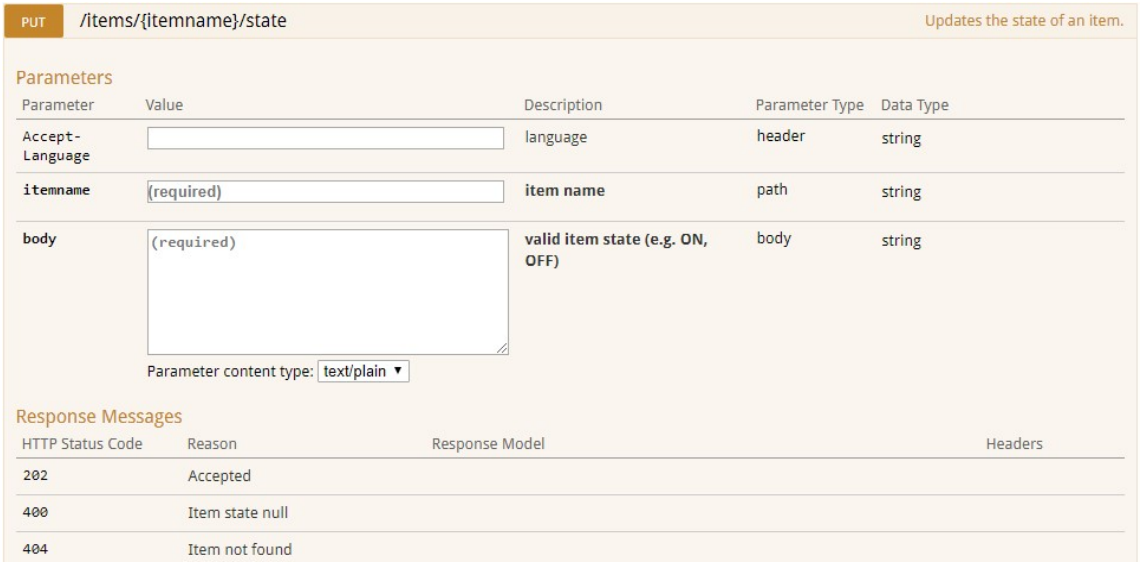

Figura 30: l'interfaccia del metodo PUT /state, esposta dalle API di openHAB.

6. Come utilizzare SafeHome

Dopo aver seguito la guida di installazione di openHAB [47], lanciare il server eseguendo lo script *start.bat* presente nella cartella della release.
Se openHAB non è stato installato su una macchina online, dopo qualche secondo in cui l'applicazione si avvierà, sarà possibile accedere al runtime all'indirizzo *http://localhost:8080*, al quale l'utente visualizzerà la pagina iniziale di openHAB. Potrà quindi scegliere la sua UI preferita, ad esempio Basic UI, nella quale sceglierà di visualizzare la sitemap della propria smart home.
Eseguito questo processo, l'utente può a questo punto lanciare i programmi Ambiente e Sensori nell'ordine che preferisce. I sensori inizieranno a rilevare dati non appena sarà trascorso il loro *refresh time*, modificabile nei file di configurazione nella cartella */devices* come spiegato nella *sezione 6.1*, e il sistema si troverà nello stato di funzionamento continuo appena tutti i sensori (esclusi i rilevatori di movimento) avranno effettuato la prima misurazione.
L'utente potrà accorgersi dei cambiamenti nei dati sulla propria interfaccia remota, e potrebbe anche introdurre nuove alterazioni nei valori tramite il processo di variazione dati.

Se l'ipotetico utente fosse invece interessato unicamente al pannello di testing, dovrebbe a questo punto navigare nell'app locale fino alla pagina *Test Scenario*: attraverso l'interfaccia di Sensori potrà selezionare uno degli scenari predisposti, immettere i parametri desiderati e visualizzare l'andamento sull'interfaccia remota. Nel caso desiderasse utilizzare lo scenario dimostrativo standard utilizzato per la sperimentazione nella *sezione 7.1*, lui o lei potrebbe avviarlo tramite la corrispondente pagina dell'applicazione Ambiente; altrimenti potrebbe immettere personalmente le variazioni desiderate.

6.1 Modificare la configurazione della smart home

Per un ipotetico utente interessato alla virtualizzazione della propria smart home non sono sufficienti le due stanze e i due dispositivi inseriti nella demo presentata con questo lavoro, ma si immagina facilmente che lui o lei vorrà **aggiungere altri locali** in cui **installare altri sensori** o **modificare i dati iniziali** di quelli già presenti; anche un ipotetico utente interessato unicamente al pannello di test inoltre potrebbe voler aumentare il numero di misuratori o di stanze.
È possibile inserire questi elementi alterando la configurazione iniziale del sistema SafeHome, tramite la **creazione di nuovi file dati** o la modifica di quelli già presenti.
Tali file vengono letti dalle applicazioni durante la fase di avvio, nella quale i sistemi virtualizzano degli oggetti aventi le caratteristiche specificate al loro interno, come descritto nella *sezione 4.1*.

È richiesta all'utente una minima competenza tecnica qualora volesse effettuare l'editing di tali file in locale, mentre una decisamente più elevata nel caso in cui volesse alterarne la configurazione remota. La stessa capacità è comunque richiesta da openHAB [6], quindi si suppone che l'utente abbia una sufficiente familiarità con queste materie.

Aggiungere nuovi dispositivi, sensori e stanze

Partendo per semplicità dal sistema Ambiente, **l'aggiunta di nuove stanze** è realizzabile **creando file testuali** (con estensione *.txt*) all'interno della cartella */rooms* (raggiungibile al percorso */src/main/java/safehome_se/local/*). Non c'è un requisito sul nome dei nuovi file, ma si consiglia di nominarli come le stanze che si desidera aggiungere (ad esempio, *Cucina.txt*).
Si consiglia inoltre di prendere come riferimento il file *demo_Bathroom.txt* contenuto nella stessa cartella, il cui contenuto è riportato in figura PIC. Una guida all'uso è stata comunque inserita all'interno di entrambi i file presenti di default, specificata da una riga che inizia con un ***** (carattere che è possibile usare per l'appunto per inserire commenti, che verranno ignorati dalla lettura automatica del file).

```
***** CONFIG FILE FOR ROOM demo_Bathroom *****

***** it contains data such as name, last sensor id who made a measure request to this room, with timestamp.
***** parameters follow the patterns [paramID] "value" *****
***** for each data (e.g. temperature), this file contains info about those data, like their current value, i

[roomName] "demo_Bathroom"
[lastRequest] ""
[lastRequestTime] ""

***** info about sensor ids that can measure this room's data (note: not devices, but sensors! like a Tempera

[sensorUID_PlacedHere_0] "TEMP04"
[sensorUID_PlacedHere_1] "HUMD05"
[sensorUID_PlacedHere_2] "FLOD08"

***** data start here *****

[dataRow_0] "TEMPERATURE"
[dataCell_0] "20.0000 °C"
[dataRow_1] "HUMIDITY"
[dataCell_1] "RH: 40.0000 %"
[dataRow_2] "AIR_QUALITY"
[dataCell_2] "CO2: 3500.0 ppm  - CO: 7.0 ppm - PM2.5: 15.0 ppm - TVOC: 300.0 ppb"
[dataRow_3] "SMOKE"
[dataCell_3] "0.2 % obs/m"
[dataRow_4] "MOTION"
[dataCell_4] "Movimento 0.0"
[dataRow_5] "FLOOD"
[dataCell_5] "2.0 V"
```

Figura 31: il file dati per la stanza demo_Bathroom. Si possono notare le sezioni di riepilogo dei sensori installati e l'elenco dei dati presenti in origine nella stanza.

Il file contiene un elenco coppie **parametro** / **valore** che contengono informazioni quali nome della stanza e relativi dati. La coppia **deve rispettare il formato** [*parametro*] "*valore*", e il nome dei parametri non deve essere modificato.

La prima sezione di informazioni contiene oltre al nome, i valori iniziali dell'ultima richiesta fatta a questa stanza, con id del richiedente e relativo timestamp, ma non è necessario che abbiano valori significativi. Sono perlopiù stati inseriti in un'ottica di espansione del codice in cui sarà implementata la permanenza dei dati ad oltre un esecuzione dell'applicazione (per ora, infatti, tutti i dati sono volatili).

La seconda sezione contiene **la lista dei sensori** in grado di misurare i dati di questa stanza, con il formato [*sensorUID_PlacedHere_x*] dove al posto di x va inserito qualcosa che **renda il parametro univoco** (il file

default utilizza ad esempio dei numeri progressivi, ma nulla vieta di mettere il nome del sensore, o lettere dell'alfabeto, o simili); il campo valore deve contenere lo UID del sensore (non il suo nome, ma l'identificativo, ad esempio *TEMP05*) reperibile nell'altra applicazione.
Questa lista è necessaria per il funzionamento del sistema, in quanto data la scelta di design di rendere i sensori ignari dei dispositivi nel quale sono virtualmente installati, e dato che le richieste sono standardizzate specificando lo UID del misuratore richiedente, c'è bisogno di tenere traccia, da qualche parte nel sistema Ambiente, di queste associazioni stanza -> sensori.

La terza sezione contiene infine **l'intero elenco dei dati posseduti** dalla stanza in analisi, che verranno visualizzati nella apposita tabella nella pagina Environment. L'ordine delle righe della tabella rispecchia l'ordine dei dati in questi file.
Per la creazione, vanno rispettati i formati [*dataRow_x*] e [*dataCell_x*], nei quali stavolta x deve essere sostituito con il numero della riga (partendo da zero, per la prima). I valori di dataRow devono inoltre rispecchiare i sei tipi di dato supportati, in maiuscolo (*TEMPERATURE, HUMIDITY, AIR_QUALITY, SMOKE, MOTION, FLOOD*); i valori di dataCell devono essere significativi per ciascun tipo di dato, seguendo gli standard testuali indicati nella *sezione 3.2*, *Figura 9* (dove per esempio, per un valore di umidità si usa il valore d'esempio "RH: 43.1%").

I file necessari per il sistema Sensori si strutturano in un modo molto simile.
Sono contenuti nella cartella /devices (raggiungibile all'indirizzo *src/main/java/safehome_ss/local/*) e contengono tutte le informazioni necessarie alla configurazione iniziale dei dispositivi e dei sensori contenuti al loro interno. Si prende come riferimento per l'analisi il file *demo_D00.txt*, il cui contenuto è riportato in *Figura 32*.

```
***** CONFIG FILE FOR DEVICE (aka SENSORTHING) demo_D00 *****

***** it contains data such as this SensorThing name, model, unique identifier, status, sensors. *****
***** parameters follow the patterns [paramID] "value" *****
***** for each supported sensor (e.g. temperature), this file contains data about said sensors, like their
***** measured data are currently unused / ignored, but inserted for future work purposes where the data c
***** parameters for each supported sensor follow this pattern:  {paramID} "value" *****

[sensorThingUID] "demo_D00"
[sensorThingName] "Demo Device 00"
[sensorThingModel] "THASM"
[sensorThingPlacedIn] "Living Room"

***** sensor data starts here *****

(sensor) "TEMPERATURE"
{sensorThingName} "Demo Device 00"
{sensorUID} "TEMP01"
{sensorName} "Demo Temperature Sensor 01"
{sensorType} "Temperature"
{sensorStatus} "OFFLINE"
{refreshTime} "10"
{measuredData} "N/A"
{lastMeasurementTime} "N/A"

(sensor) "HUMIDITY"
{sensorThingName} "Demo Device 00"
```

Figura 32: il file dati per il dispositivo *Demo Device 00*. Si possono notare le sezioni di riepilogo delle info sul dispositivo stesso e quelle sul sensore di temperatura in esso installato.

La prima sezione contiene **dati sul dispositivo**, nel solito formato tra parentesi quadre, come UID, nome visualizzato nella UI e modello. Il parametro che definisce la stanza di installazione non è usato internamente dal sistema, ma è inserito solo come valore utile all'interfaccia grafica. Il valore di *model* è usato per definire quali sensori vengono installati e quali dati l'applicazione si aspetta di ricevere più avanti.

La seconda sezione contiene invece **l'elenco dei dati dei sensori**. Essa inizia con una riga del tipo *(sensor) "tipo di dato rilevabile"*, dove le parentesi tonde sono necessarie per la navigazione della lista e il riconoscimento di una zona relativa a questo o quel tipo di dato. Esso inoltre deve essere uno tra i sei valori standard, sempre in maiuscolo (*TEMPERATURE*, ecc.).
Viene ripetuto il nome del dispositivo, ma solo ai fini di visualizzazione nella GUI (ricordiamo che ne è ignaro, ma la UI ne ha bisogno per visualizzarlo, quindi il dato va inserito da qualche parte).
Dopo la specificazione dello UID, fondamentale per il funzionamento interno, si trovano nome e tipo che sono utili per scopi grafici (ad esempio, il tipo può essere tradotto in italiano senza ripercussioni); più sotto si trova lo status, che può valere come già analizzato *OFFLINE, STARTED, ONLINE* e *FAILURE*, e il **refresh time** in secondi che specifica il periodo di richiesta di misurazione.

Un sensore **impostato a *FAILURE*** da qui **mantiene la caratteristica di fallimento delle misurazioni** finché non viene riavviato l'intero dispositivo dall'interfaccia grafica, o modificato questo stesso valore e riavviata l'applicazione.

Gli ultimi due parametri, prima di passare ai dati riguardanti il successivo sensore, contengono due valori (*dato misurato* e *ultima misurazione*) che non sono attualmente utilizzati da SafeHome ma sono inseriti, come per l'analogo campo in Ambiente, nell'ottica di sviluppo futuro in cui i dati vengono resi permanenti: allora, questi campi andrebbero settati con il valore dell'ultima misurazione effettuata.

Modificare openHAB

Modificare l'interfaccia remota è un'operazione molto più complicata perché richiede quelle capacità che la openHAB Foundation definisce *hacking skills* [6]: oltre ad aver assimilato tutti i concetti chiave di openHAB, infatti, all'utente è richiesto di modificare specifici file e se desiderasse implementare nuove regole per automatizzare alcuni processi di possedere anche basilari capacità di programmazione.

È anche possibile **utilizzare le UI fornite da openHAB**, in particolare la PaperUI per la creazione di things e la Home Builder per la creazione di una nuova sitemap, con le stanze ed items desiderati, e nonostante l'editing dei file sia considerato preferibile perché con meno limitazioni, in questa breve sezione si farà riferimento all'uso delle suddette due interfacce grafiche per quanto possibile. Nel caso si desiderasse comunque modificare i file dati, rivolgersi alla documentazione ufficiale [6].

Se l'utente desidera **aggiungere dei dispositivi all'interfaccia remota**, per esempio, deve per prima cosa navigare all'interno della PaperUI fino a *Configuration -> Things* (o equivalentemente *Inbox*) e cliccare sul pulsante + per aggiungere una nuova thing. Quando verrà richiesto di scegliere il binding del nuovo dispositivo, selezionare *SafeHome_SE* (che dovrebbe apparire automaticamente se è presente il file *.jar* nella cartella */addons* della release di openHAB). A questo punto selezionare il modello del nuovo device, ad esempio THASM, e confermare, modificando se desiderato l'id univoco e il parametro unicamente grafico Location.

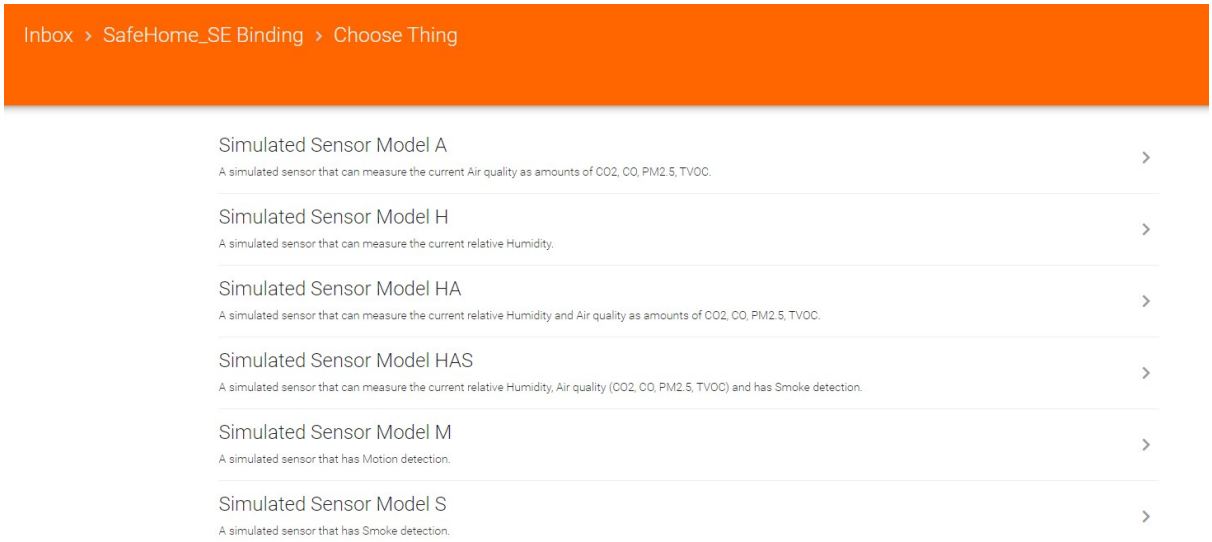

Figura 33: installazione di un dispositivo supportato da SafeHome tramite la PaperUI di openHAB.

A questo punto, è necessario modificare il file *.items* (in */items*) aggiungendo un oggetto e collegandolo alla nuova thing, scrivendo una nuova linea di questo genere (o prendendo l'output dalla UI Home Builder e copia/incollandolo nel file):

itemtype itemname "label" <icon> (group1, ...) {channel= }

sostituendo le appropriate parole alle variabili, come in *Figura 34* per l'item *LivingRoom_TEMP01*.

Per SafeHome è importante che tra l'elenco dei *groups* si includa uno dei valori raffigurati in *Figura 34*, che rappresentano i gruppi dei tipi di dato supportati.

Il processo di aggiunta di un nuovo device a questo punto è terminato, e lo si potrà visualizzare se si è inserito all'interno dei *groups* dell'*item*, una delle stanze già presenti.

```
Group Home "SafeHome Demo" < house > ["Building"]

Group GF "Home" < groundfloor > (Home)["GroundFloor"]

Group FamilyRoom "LivingRoom" < parents_2_4 > (Home, GF)["Room"]
Group Bathroom   "Bathroom" < bath > (Home, GF)["Bathroom"]


// data type groups
Group Temperature
Group Humidity
Group AirQuality
Group CO2 (AirQuality)
Group CO (AirQuality)
Group PM25 (AirQuality)
Group TVOC (AirQuality)
Group Smoke
Group Motion
Group Flood
Group ArmDisarm

// ***** family room items start here *****

Number LivingRoom_TEMP01 "Temperature" < temperature > (FamilyRoom, Temperature)
{channel = "safehome_se:model_thasm:THASM00:temperature_sensor#curr_temperature"}

Number LivingRoom_HUMD02 "Humidity" < humidity > (FamilyRoom, Humidity)
{channel = "safehome_se:model_thasm:THASM00:humidity_sensor#curr_humidity"}
```

Figura 34: estratto del file *.items*, nel quale si notano i Group per ogni tipo di dato e alcuni degli item presenti nella stanza *LivingRoom*, appartenenza definita dalla presenza del gruppo *FamilyRoom* nella lista dei gruppi dell'oggetto.

Se invece l'utente desidera **aggiungere una nuova stanza**, il processo è molto più semplice: basterà infatti creare un nuovo item di tipo Group (ad esempio *Kitchen*) nel file *.items* sul modello in *Figura 34*, poi aggiungere il nuovo gruppo alla lista dei gruppi del dispositivo appena creato al fine di installarlo nella stanza.

```
sitemap safehome label="SafeHome" {
    Frame label="Home"{
        Group item=FamilyRoom
        Group item=Bathroom
    }

    Frame label="Alarm"{
        Switch item=Alarm_Activator mappi
        Default item=Alarm_Triggered
        Default item=Alarm_Triggered_Time
        Text item=Alarm_Triggered_Reason
    }
```

Figura 35: estratto del file *.sitemap*, in cui si nota la struttura della pagina principale dell'interfaccia remota.

Infine, una piccola modifica al file *.sitemap* è necessaria, sempre tramite editing del file o *Home Builder*, aggiungendo all'elemento *Frame* "Home" della sitemap dimostrativa il nuovo elemento Group, subito sotto ai due mostrati in *Figura 35*.

È per ultimo possibile **modificare i valori di soglia delle situazioni indesiderate**, tranne quelle dipendenti da valori interni di un sensore, andando a modificare gli appositi parametri contenuti all'interno del file *.rules*, raggiungibile nell'omonima cartella.

Per una descrizione della modifica o aggiunta di regole per automatizzare processi, o un elenco più dettagliato di tutti i tipi di dato supportati dal framework, o di nuovo un aggiunta di ulteriori things relativi ad altri binding, riferirsi alla documentazione ufficiale [6].

7. Incertezza del sistema

Il sistema SafeHome è stato utilizzato in questo lavoro di tesi allo scopo di **modellare e valutare situazioni di incertezza** tipiche della realtà, le quali sono state inserite nel contesto limitato e controllato offerto dalla soluzione proposta.

In letteratura non c'è una definizione formale univoca di incertezza: per alcuni è *l'assenza di conoscenza*, per altri *l'inadeguatezza delle informazioni possedute* [50] [51], ma la definizione apparentemente più completa è quella proposta da [52] in cui la si classifica su tre dimensioni: *luogo* (*location*), *natura* e *livello*. La prima si può generalmente riassumere in un'indicazione dell'origine dell'incertezza stessa, ad esempio legata agli input di determinati parametri o al funzionamento interno del sistema in analisi; la seconda classificazione ne specifica la causa, in *aleatoria* quando legata al sistema stesso e in *epistemica* quando è la mancanza di informazioni a determinarla; il parametro livello è praticamente un indicatore dell'intensità dell'incertezza, con valori da 0 a 3 che variano tra *conoscenza* ed *ignoranza assoluta* passando per il più interessante livello 1 di *consapevolezza della presenza di incertezza* (*awareness of uncertainty*).

SafeHome possiede intrinsecamente **due fonti di incertezza** che si possono quindi classificare secondo la definizione appena proposta, mostrate e classificate in *Figura 36*.
La prima è legata **al valore finale risultante da una variazione** nel sistema Ambiente: come già descritto, il dato al termine della modifica desiderata dall'utente si troverà in un intorno positivo / negativo del valore obiettivo. A causa di questo, nonché dell'approssimazione fatta dalla sensibilità dei sensori, il sistema SafeHome può occasionalmente transitare nello stato *InAllarme* quando l'utente non lo desidera, nel caso in cui un valore obiettivo sia sufficientemente vicino ad un valore di soglia di una situazione pericolosa.

La seconda fonte, che stavolta è un'assenza di conoscenza, è legata invece alla configurazione lasciata all'utente dei file necessari al funzionamento del sistema in un caso d'uso diverso da quello presentato: se infatti un ipotetico utente volesse aggiungere nuovi dispositivi, sensori o stanze dovrebbe seguire il procedimento piuttosto complesso già descritto, e non è possibile sapere come si comporterebbe il sistema in caso di **errori nella configurazione**.

Fonte di incertezza	Situazione	Luogo	Natura	Livello
Approssimazione	Variazione % dei dati nel sistema Ambiente	Contesto / Struttura del modello	Aleatoria	1
Configurazione erronea	L'utente configura in modo incorretto i file	Input	Epistemica	3
Fallimento di un sensore	Un sensore smette di funzionare	Contesto	Aleatoria	1 / 2
Errore nella misurazione	Un sensore applica una incertezza di misura	Input	Aleatoria / Epistemica	1
Ritardo di openHAB	Il server risponde dopo un certo tempo / va in timeout	Contesto	Aleatoria	1 / 2

Figura 36: le incertezze associate al sistema SafeHome classificate secondo la tassonomia in [52].

A queste incertezze intrinseche del sistema sono state accuratamente aggiunte delle fonti estremamente comuni in controparti reali del modello in analisi: l'obiettivo è stato quello di fornire un contesto limitato e controllabile a tali fonti di incertezza e verificare il comportamento di SafeHome in ciascuna di queste situazioni, denominate **scenari di test**. Ognuna di queste è di primario interesse alla proposta della soluzione software perché è legata a **particolari requisiti non funzionali** come il tempo di risposta del sistema o la probabilità di raggiungimento dello stato *InAllarme*, caratteristiche che sono sicuramente d'interesse ad un ipotetico cliente di un progetto reale.

Sono state delineati per la precisione tre scenari, e in *Figura 36* è possibile notarne la classificazione:

- **Fallimento di un sensore**
- **Errore nella misurazione**
- **Ritardo di openHAB**

Per l'analisi di ognuna di queste situazioni **è stato realizzato un contesto standard** in cui determinati tipi di dato seguono un preciso andamento nel tempo, superando in specifiche occasioni i valori indesiderati di soglia. Lo scenario, definito *scenario di test standard*, viene analizzato nel dettaglio nella prossima sezione. Ne è stato osservato il **numero di raggiungimenti dello stato *InAllarme*** del sistema (cioè, il numero di chiamate di allarme prodotte dal server di openHAB), osservazione che è stata poi confrontata con i risultati dei numeri di raggiungimenti dello stesso stato in ciascuno dei quattro scenari.

Mentre per queste ultime è stato possibile condurre un esperimento finalizzato all'osservazione della suddetta proprietà, non è stato possibile farlo per ovvi motivi per l'incertezza 'configurazione erronea', il cui comunque livello 3 (*totale ignoranza*) non presentava la stessa significatività che presentano gli altri scenari di test.

Scenario	Fonte di incertezza	Proprietà osservata	Requisito non funzionale interessato
S1	Fallimento di un sensore	Raggiungimento dello stato InAllarme	Affidabilità
S2	Errore nella misurazione	Raggiungimento dello stato InAllarme, numero di falsi positivi e falsi negativi	Affidabilità, Performance
S3	Ritardo di openHAB	Raggiungimento dello stato InAllarme	Affidabilità, Performance

Figura 37: gli scenari di test proposti.

7.1 Contesto standard di esecuzione

Come accennato è stato costruito un contesto standard nel quale sono stati eseguiti i quattro scenari di test. Questo scenario standard, della durata di circa 9 minuti e 30 secondi, prevede che ogni tipo di dato rilevabile dal sistema SafeHome venga variato partendo da dei valori di default desiderabili in specifici istanti di tempo, fino a precisi valori obiettivo. Tali variazioni avvengono come descritto nella *sezione 3.2* e sono dunque soggetti all'incertezza di approssimazione appena descritta risultando in valori nell'intorno positivo del target, aumentando, e negativo, diminuendo.

Gli andamenti sono stati realizzati ad hoc **per permettere una possibilità di riduzione del numero di raggiungimenti dello stato *InAllarme*** (la proprietà osservata), che altrimenti non sarebbe significativo osservare in situazioni di non variazione dei dati.
Ad esempio, in molte occasioni i dati sono stati fatti avvicinare ai valori soglia, o fatti fluttuare attorno per brevi periodi di tempo, proprio per permettere la possibilità a tali avvenimenti di accadere: in uno scenario di errore di misurazione infatti una situazione simile potrebbe occasionalmente produrre uno scostamento nella misura tale da provocare il trigger dell'allarme quando non si sarebbe dovuto (o non provocarlo, nel caso inverso).

L'andamento di ciascuno dei dati è riportato nei grafici seguenti (*Figure 38 - 43*), nei quali vengono evidenziati gli istanti di tempo d'inizio delle alterazioni e i corrispondenti valori obiettivo. Vengono inoltre sempre evidenziati i valori di soglia indesiderati. Dopo che anche l'ultima variazione di un dato ha raggiunto il proprio valore obiettivo, tale dato è considerato stabile fino alla fine dello scenario di test.

Gli andamenti dei vari dati sono presentati separatamente, ma sono da considerarsi **contemporanei**: un superamento di un qualsiasi valore di soglia causa il raggiungimento dello stato *InAllarme*, in cui il sistema permane per il *tempo di reset*, assunto in tutti gli scenari pari a 10 secondi. Eventuali rilevazioni superiori alla norma in questo periodo di tempo **vengono ignorate**, ovvero non causano la transizione verso lo stato *InAllarme*, **anche se causate da un diverso tipo di dato**. Ad esempio, al tempo t = 460 s, avvengono contemporaneamente un allagamento in bagno (*Figura 39*) e un superamento della temperatura desiderata in soggiorno (*Figura 40*): l'allarme può scattare per uno qualsiasi dei due motivi, a seconda di quale sia la prima misurazione a raggiungere il server di openHAB.
Il sistema viene automaticamente posto nello stato *Normale* dopo quello stesso tempo di reset, dopo il quale può avvenire una successiva situazione indesiderata.

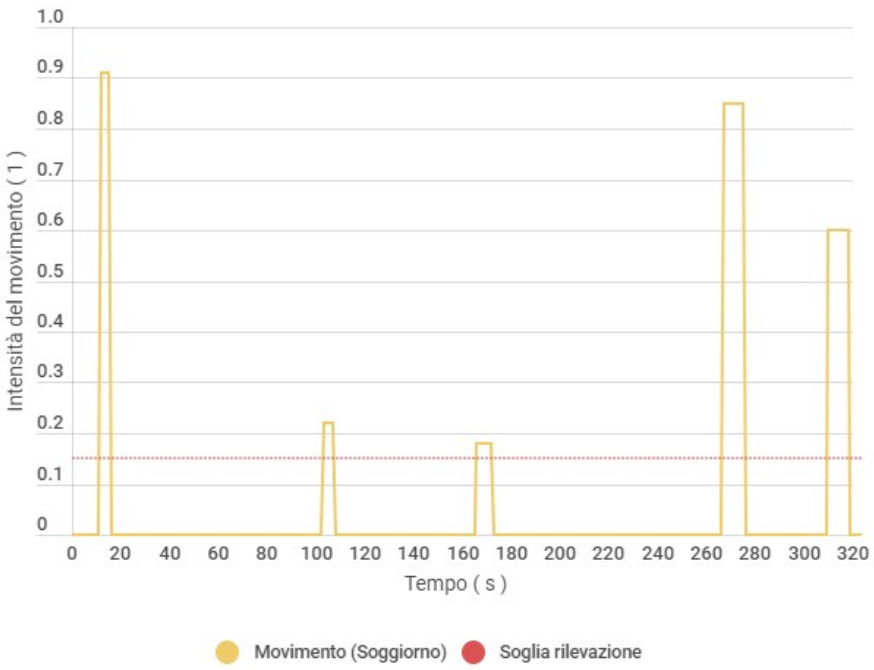

Figura 38: andamento nel contesto standard del dato *Movimento* nella stanza *Soggiorno*.

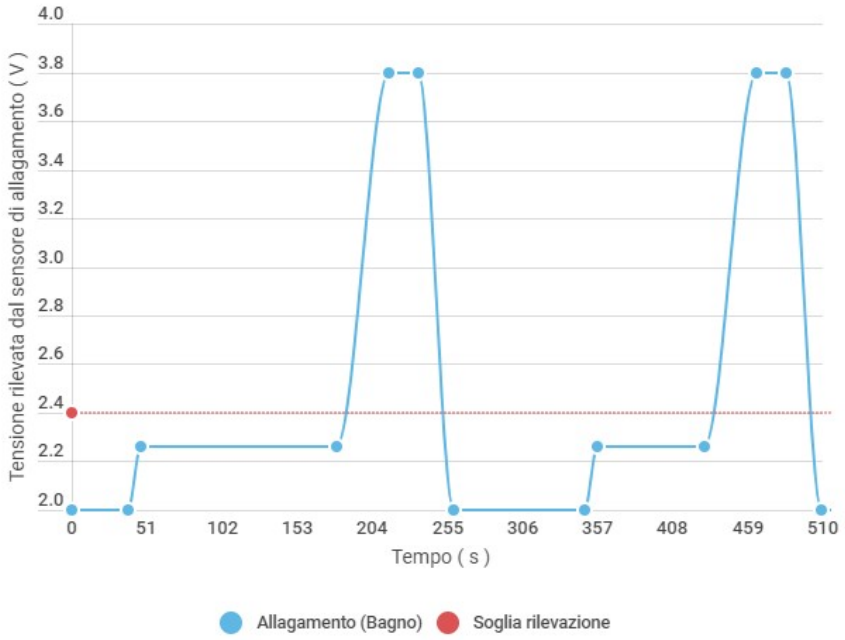

Figura 39: andamento nel contesto standard del dato *Allagamento* nella stanza *Bagno*.

Inoltre, è da evidenziare come un dato possa **rimanere stabile** per un certo periodo di tempo, entro il quale possono avvenire **nuove misurazioni** da parte del sistema Sensori, se questo periodo supera il *refresh time* del sensore, che in tutti gli scenari è stato posto pari a 10 s per tutti i misuratori; se tale stabilità avviene con un valore che è oltre una soglia, superato il tempo di reset dell'allarme sicuramente già scattato, la successiva misurazione porterà immediatamente il sistema di nuovo nello stato *InAllarme*. Ad esempio, analizzando l'andamento del valore di tensione fornito dal sensore di allagamento posto in bagno, si nota che tra i tempi t1 = 205 s e t2 = 235 s (circa) il dato rimane stabile ad un valore di 3.8 V. Il refresh time del sensore, come detto di 10 s, permette circa 3 misure, ognuna delle quali provocherà la transizione verso lo stato *InAllarme*.

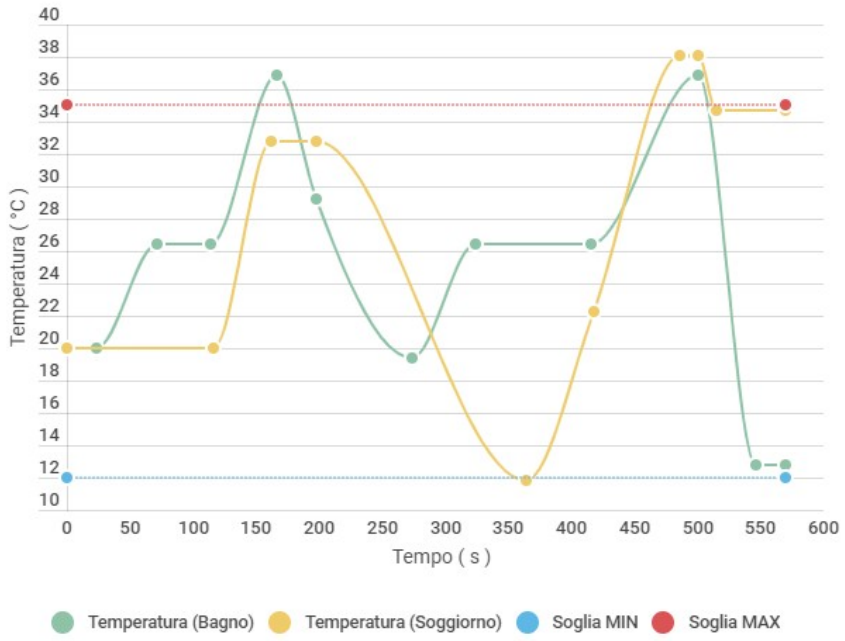

Figura 40: andamento nel contesto standard del dato *Temperatura* nelle due stanze *Soggiorno* (in giallo) e *Bagno (in verde)*.

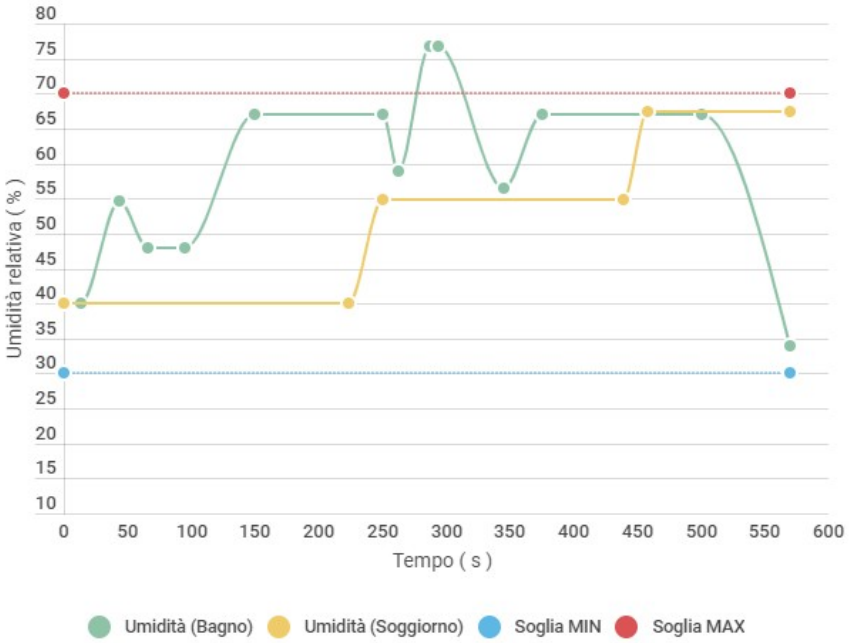

Figura 41: andamento nel contesto standard del dato *Umidità* nelle due stanze *Soggiorno* (in giallo) e *Bagno (in verde)*.

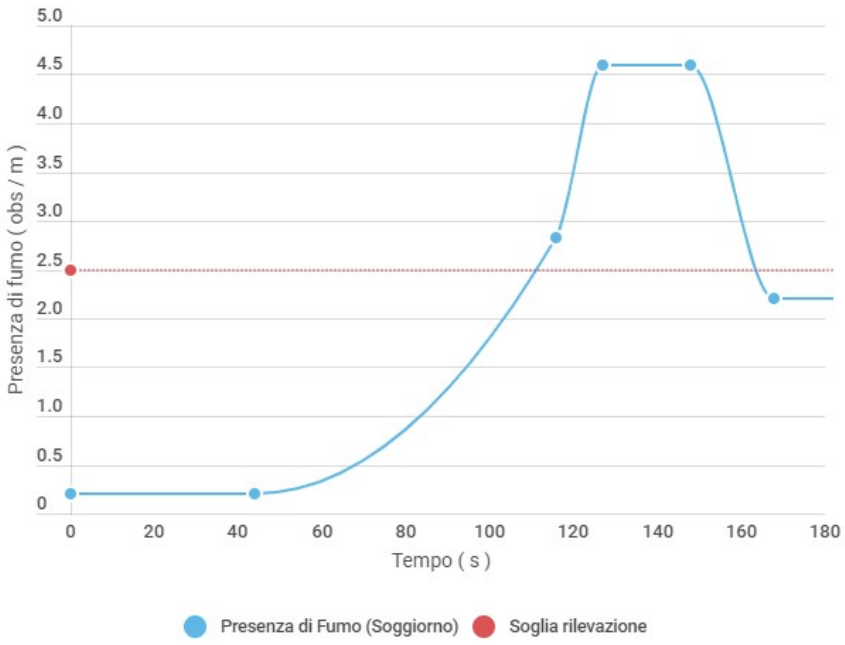

Figura 42: andamento nel contesto standard del dato *Presenza di fumo* nella stanza *Soggiorno*.

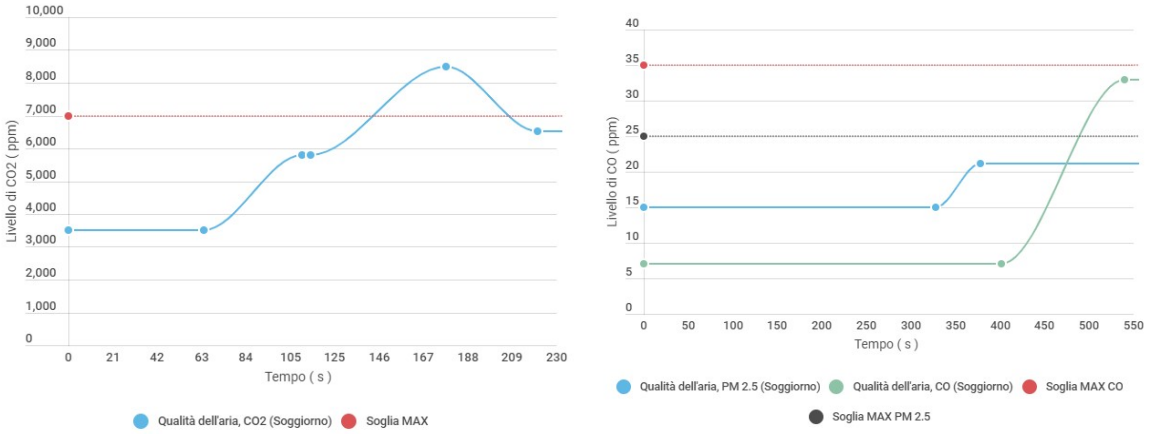

Figura 43: andamento nel contesto standard di ogni quantità del dato *Qualità dell'aria* nella stanza *Soggiorno*. Nello specifico, andamento dei livelli di CO_2 (a sinistra) e PM 2.5 (a destra, in azzurro, la cui soglia è in nero) e CO (a destra, in verde, la cui soglia è in rosso). Non viene mai variato il livello di TVOC.

L'incertezza intrinseca legata alla variazione dei dati sopra descritta non ha influito sul numero osservato, ma nel creare lo scenario **è stata introdotta involontariamente una nuova incertezza**: lo scenario infatti per iniziare necessita dell'avvio di entrambe le applicazioni Sensori ed Ambiente. L'applicazione Sensori, però, effettua lo scheduling dei job che si occupano di richiedere la misurazione non appena un dispositivo (e quindi un sensore al suo interno) viene acceso. A seconda dell'esatto momento in cui viene fatta l'accensione **si verifica uno sfasamento** dei tempi di misurazione che per quanto piccolo non è trascurabile, a causa del fatto che anche Ambiente **ha uno sfasamento legato al momento di avvio dello scenario**; a

seconda di tali shift temporali, valori diversi verrebbero quindi rilevati dall'applicativo e questo in uno scenario sufficientemente lungo come quello in analisi, porta inevitabilmente ad avere una certa variabilità nei risultati.

Questa incertezza è stata verificata mediante 11 misure della proprietà in osservazione *numero di raggiungimenti dello stato InAllarme* che hanno fornito la distribuzione riportata in *Figura 44*, la cui media è pari a **μ = 20.36 allarmi** con una deviazione standard di **σ = 1.03 allarmi**.

	1	2	3	4	5	6	7	8	9	10	11	Media	Dev. Std
Contesto standard	20	20	20	22	21	19	22	20	20	21	19	*20.36*	*1.03*

Figura 44: risultati nella proprietà *Numero di Allarmi* degli 11 test effettuati nel contesto standard.

7.2 Scenari di test

In questa sezione vengono descritti i tre scenari di test appena definiti. Tutti e tre utilizzano come base lo stesso contesto standard.
Ogni scenario ha una breve introduzione nella quale vengono analizzati i parametri che lo caratterizzano, dopodiché viene riportata una tabella contenente i risultati delle osservazioni della proprietà *raggiungimento dello stato InAllarme* del sistema SafeHome al variare dell'input, e una rappresentazione grafica equivalente.
Ad ogni scenario è inoltre associato un requisito non funzionale, il cui razionale è spiegato anch'esso nell'introduzione.

L'avvio di uno scenario di test viene fatto attraverso l'omonima pagina nella GUI, raggiungibile dal menù laterale delle due applicazioni: in Ambiente si sceglie quando avviare il contesto standard, mentre in Sensori si immettono gli input necessari e, se corretti, si può attivare lo scenario che si desidera.

I risultati in termini di *numero di raggiungimenti dello stato InAllarme*, per qualunque caso di test si scelga di eseguire, sono osservabili tramite l'interfaccia remota del sistema SafeHome (cioè l'istanza del server di openHAB), precisamente nella *sitemap* "SafeHome Demo" presentata assieme a questo elaborato - e reperibile assieme all'intero codice del sistema all'indirizzo *https://github.com/aleverga10/SafeHome* .
All'apertura di quella pagina, infatti, l'utente troverà un'area dedicata agli scenari di test (chiamata per l'appunto "*Testing scenarios*"), nella quale potrà visualizzare unitamente ad informazioni sul caso incerto in osservazione, i risultati nel *numero di allarmi* prodotti dalle situazioni considerate potenzialmente pericolose. La fine dello scenario, o del contesto standard egualmente, della durata complessiva di circa 9 minuti e 30 secondi, viene sempre segnata dalla staticità dei dati in analisi: dopo che tutte le variazioni predefinite dal contesto standard avranno avuto luogo, infatti, l'andamento di ogni dato in ogni stanza sarà indefinitamente costante all'ultimo valore assunto.

La costruzione della smart home utilizzata in ognuno dei casi analizzati è la seguente:

- stanza *LivingRoom*, nella quale è stato installato un dispositivo *modello THASM*, capace cioè di effettuare rilevazioni di temperatura, umidità, qualità dell'aria in tutte e quattro le quantità caratteristiche, presenza di fumo e di movimento.

- stanza *Bathroom*, nella quale è stato installato un dispositivo *modello THF*, capace cioè di effettuare rilevazioni di temperatura, umidità e presenza di allagamento.

Di nuovo, tutti e tre i test sono stati eseguiti nel contesto standard, perciò le misurazioni effettuate da tali dispositivi hanno rispecchiato - a meno dell'incertezza analizzata - gli andamenti appena descritti.

Fallimento di un sensore (Scenario S1)

In questo primo scenario ad ogni sensore è **associata una probabilità di fallimento**, *p(F)*.
L'evento di fallimento viene verificato poco prima di effettuare una nuova misurazione su un dato appena ricevuto.
L'implementazione della verifica dell'evento è banale: viene randomizzato un float (utilizzando la libreria *java.util.Random*) e controllato che sia minore o uguale a p(F). In tal caso, l'evento si verifica.

Il fallimento di un sensore ha le seguenti conseguenze:
- il sensore non può effettuare la misurazione sul dato appena ricevuto
- il valore del corrispettivo campo sull'interfaccia remota di openHAB rimarrà stabile all'esito della misurazione precedente (anche se è un valore sopra una soglia)
- il sensore non può più richiedere nuove misurazioni
- il sensore passa dallo stato *Online* allo stato *Failure*, che è uno stato finale

La **proprietà osservata** è il ***numero di raggiungimenti dello stato InAllarme*** del sistema, che viene poi raffinata nella *probabilità di non attivazione dell'allarme*.

Le probabilità in input sono state scelte appositamente a seguito dell'applicazione inversa della seguente formula, nella quale l'incognita è p(f), e p(Fs) rappresenta la probabilità di fallimento di un singolo sensore per tutto lo scenario.

$$p(Fs) = 1 - [1 - p(f)]^{57}$$

dove 57 è il numero totale di misurazioni che ogni sensore (tranne il rilevatore di movimento) effettua nel contesto standard, in cui essi sono infallibili. Il numero è ricavabile dalla durata dello scenario (= 570 s) e dal *refresh time* dei sensori, impostato per ognuno pari a 10 s.

La formula è ricavata dall'espressione della probabilità di una **variabile casuale binomiale** in cui si assume come 'successo' il verificarsi dell'evento di fallimento in un numero D di tentativi assunti indipendenti, la seguente

$$p(x = n) = \frac{D!}{n! * (D - n)!} * p^n * (1 - p)^{D-n}$$

Se si verificano n = 0 successi, la formula si riduce a

$$p(x = 0) = (1 - p)^D$$

che nello scenario in analisi sarebbe la probabilità in cui un singolo sensore non fallisca mai per tutta la durata dello stesso; la sua inversa, quindi, ci dà la probabilità che un sensore fallisca almeno una volta (o nel caso in esame, che fallisca).

La tabella seguente mostra i valori di p(Fs) desiderati e i conseguenti valori di p(f) scelti per essere testati.

p(Fs)	p(f) effettiva	p(f) usata nei test
25%	0.005035	0.005
35%	0.007530	0.0075
45%	0.01045	0.01
55%	0.01395	0.014
70%	0.021	0.02
95%	0.0517	0.05

Figura 45: valori di probabilità di fallimento desiderata per l'intero scenario, p(Fs), associati ai valori ricavati dalla formula della binomiale, p(f). A fianco, i valori di p(f) usati nei test.

La probabilità di fallimento di un sensore di movimento è soggetta ad un fattore di correzione moltiplicativo, pari ad esattamente *Fm = 0.565*. Questa correzione tiene conto del fatto che un sensore di movimento, per sua costruzione, non effettua il polling del dato ad Ambiente, come gli altri rilevatori, ma gli viene 'consegnato' il valore di intensità non appena in Ambiente si verifica una variazione del dato Movimento. Avvenendo esattamente 5 rilevazioni per tutta la durata dello scenario, più le 5 misure automatiche al termine dell'attività in cui il dato viene resettato a zero (che possono comunque far fallire il sensore), il numero D di tentativi è quindi 10. Il valore è stato ricavato dall'applicazione della stessa formula di cui sopra, alla quale a p(f) è stato sostituito [*p(f) * Fm*].

I risultati delle osservazioni per ciascuna delle probabilità elencate è riportato nella tabella seguente, nella quale viene indicato tra parentesi il numero di fallimenti verificati, mentre nel grafico in *Figura 47* si omette questo parametro e si rappresenta solo l'andamento della proprietà osservata:

p(Fs)	p(f)	1	2	3	4	5	6	7	8	9	10	Media	Dev. Std
25%	0.005	19	14	20	19	18	19	19	17	19	18	*18.2*	*1.69*
35%	0.0075	19	18	19	21	17	19	13	17	12	21	*17.6*	*3.03*
45%	0.01	14	14	20	17	19	15	13	18	17	19	*16.6*	*2.46*
55%	0.014	16	19	11	12	12	18	17	19	12	18	*15.4*	*3.27*
70%	0.02	15	14	7	11	4	5	5	14	14	6	*9.5*	*4.5*
95%	0.05	5	2	5	3	5	2	5	8	1	6	*4.2*	*2.14*

Figura 46: risultati nel *Numero di Allarmi* dei 10 test effettuati nello scenario *Fallimento di un sensore*.

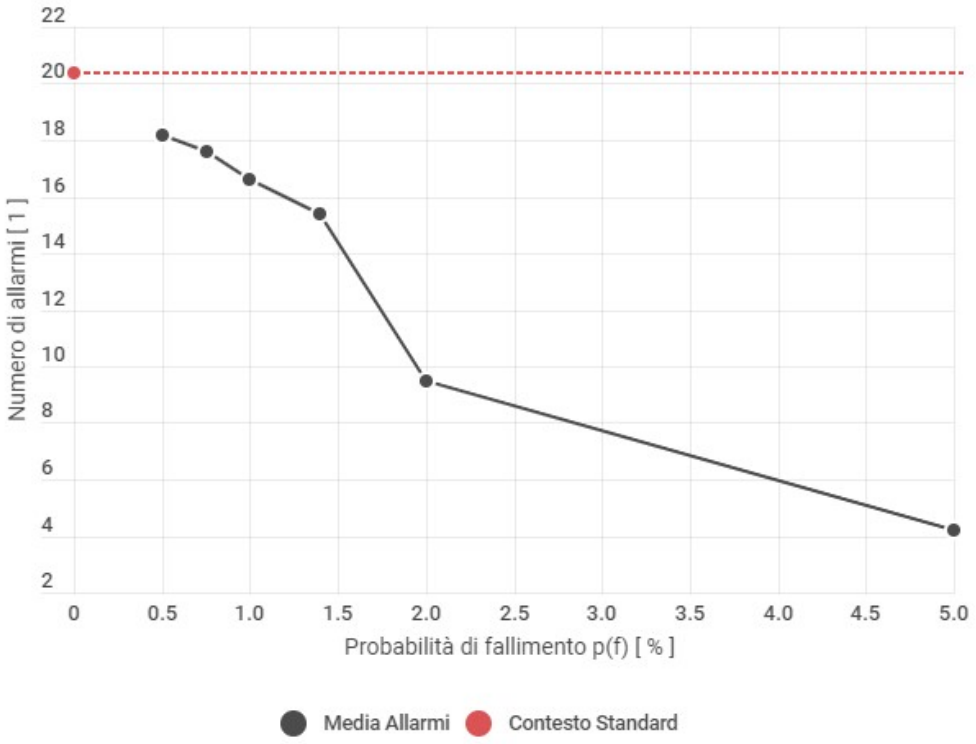

Figura 47: andamento della proprietà *Numero di allarmi* al variare del parametro p(f), graficato in %.

p(Fs)	p(f)	1	2	3	4	5	6	7	8	9	10	Media	Dev. Std
25%	0.005	2	2	0	1	2	3	1	2	2	1	*1.6*	*0.84*
35%	0.0075	1	4	0	0	2	3	4	2	5	3	*2.3*	*1.70*
45%	0.01	5	4	1	2	1	4	6	2	4	2	*3.1*	*1.73*
55%	0.014	5	1	4	5	4	3	3	4	6	5	*4.0*	*1.41*
70%	0.02	5	5	6	6	8	7	7	3	4	6	*5.7*	*1.49*
95%	0.05	7	8	7	8	7	8	7	7	8	7	*7.4*	*0.52*

Figura 48: risultati della proprietà *Numero di Fallimenti* nei 10 test effettuati nello scenario S1.

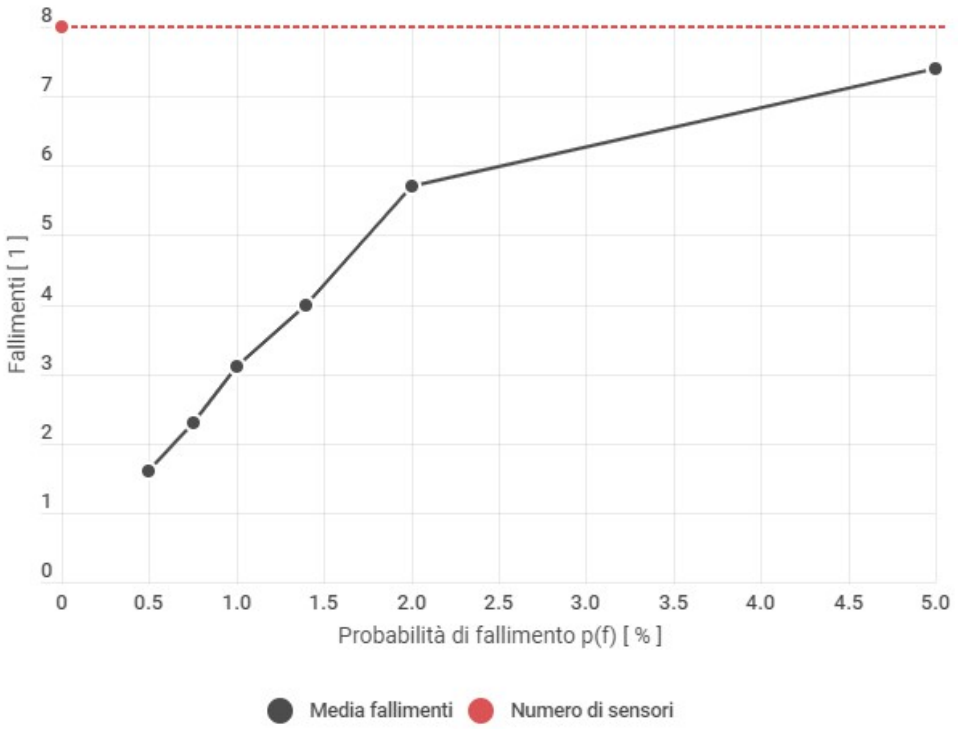

Figura 49: andamento della proprietà *Numero di fallimenti* al variare del parametro p(f), graficato in %.

Si può osservare come un numero elevato di fallimenti sia correlato ad un numero inferiore di allarmi ma non ne sia la sua diretta causa: infatti, ovviamente, è l'istante del fallimento a determinare l'abbassamento del risultato, perché inibire le misurazioni all'inizio dello scenario è chiaramente peggiore che farlo alla fine.

Il **requisito non funzionale associato** riguarda **l'affidabilità** del sistema SafeHome, e potrebbe essere espresso in funzione della differenza di probabilità di attivazione dell'allarme nel caso testato e nel caso standard. Se infatti si calcola la probabilità di attivazione dell'allarme come segue

$$p(A) = \frac{\mu_A}{M}$$

dove μA è la media del numero di allarmi e M è il numero di misure totali (pari, nello scenario, a 409 , derivato da 57 misure * 7 sensori + 10 rilevazioni di movimento)

e la si confronta con il contesto standard, che ha p(A) pari a

$$p(As) = \frac{20.36}{57 * 7 + 10} = \frac{20.36}{409} = 0.0498$$

si può attribuire la differenza totalmente ai casi di fallimento dei sensori.
Questo scostamento può quindi essere quantificato e scritto come requisito non funzionale, ad esempio nella formula seguente:

La probabilità di attivazione dell'allarme dovuta a fallimenti non deve mai superare la soglia di X %.

Fissato questo valore soglia in accordo con il cliente, si può verificare - ad esempio con valori di probabilità di fallimento presi da una situazione reale - che il sistema sia sufficientemente affidabile.

A titolo di esempio si calcola la probabilità per il valore di p(f) = 0.01 pari a

$$p(A|p(f) = 0.01) = \frac{16.6}{409} = 0.0406$$

che quindi comporta una differenza di 0.0092, o volendo esprimere la probabilità in percentuale 0.92%, equivalente ad una diminuzione del 18.5% dal contesto standard. Questo valore per esempio potrebbe già essere considerato troppo elevato per molte applicazioni reali, il che potrebbe significare problemi nel funzionamento dei sensori - ad esempio batterie inaffidabili - o che si è scelta una soglia irrealisticamente piccola.

Si precisa inoltre che il valore M = 409, somma di tutte le misurazioni effettuate nel contesto standard, è espresso con una incertezza di ± 7 misure, derivante dallo shift temporale di avvio delle due applicazioni Sensori ed Ambiente.

Errore nella misurazione (Scenario 2)

In questo scenario nell'ambiente controllato e quasi del tutto naturalmente esente da incertezza è stato introdotto un **errore di misurazione**. Vengono proposti due test, che si differenziano per il tipo di scostamento: il primo fa uso di un errore che segue una **distribuzione gaussiana** centrata attorno al valore reale da misurare e avente come varianza il **valore di accuratezza del sensore diviso 3**.
Il valore di accuratezza è ricavato per ogni tipo di dato dalla tabella in *Figura 6* (*sez. 3.1*), mentre la divisione per tre deriva dal fatto che il valore è espresso in tutte le fonti consultate come massimo errore possibile (ad esempio, ± 4% per il modello di igrometro [32]) e quindi deve essere il valore più alto possibile della curva gaussiana; l'estremo superiore però è impraticabile da ottenere, essendo la normale Z standard tendente ad 1, e quindi si è scelto come valore **3σ** (dove σ è la varianza) che rappresenta in tutte le distribuzioni gaussiane **il 99.5-esimo percentile**.Posto dunque **3σ = A** (con A valore di accuratezza di un sensore), è stato possibile ricavare il suddetto parametro di varianza.
Questo errore di misurazione avviene perciò ad ogni nuova rilevazione, e avrà un valore compreso nell'insieme (0, A), con le probabilità caratteristiche della normale.

Il secondo tipo di scostamento segue invece una **distribuzione uniforme** (cioè random). I valori di probabilità di errore random, detta p(er), sono stati scelti seguendo la stessa binomiale dello scenario 1, mentre lo scostamento sulla misurazione è stato posto costante a **±35% del valore corrente**. La scelta di non lasciar variare questo parametro all'utente è intenzionale, fatta per poter isolare il più possibile la variazione della proprietà osservata in termini di probabilità; è ricaduta inoltre su questo preciso valore, invece che fornire un numero completamente sballato, perché è possibile che dipendentemente dalla misura corrente si osservi, oppure no, un cambiamento del numero di allarmi del sistema. Scegliendo invece un valore completamente fuori scala si sarebbe inevitabilmente osservato un incremento nel numero di allarmi tendente esattamente al numero di errori, il che non sarebbe stato ugualmente significativo.

La proprietà osservata in entrambi i casi è, come nel caso 1, sempre il **numero di raggiungimenti dello stato *InAllarme*** del sistema SafeHome.
I risultati sono riportati in *Figura 50* e nel grafico in *Figura 51* viene mostrato l'andamento del valore medio al variare del parametro p(er).

p(er)	Distribuzione	1	2	3	4	5	6	7	8	9	10	Media	Dev. Std
/	Gaussiana	24	23	24	24	26	24	24	27	24	25	*24.50*	*1.17*
0.01	Uniforme	22	22	22	25	22	22	23	19	20	22	*21.90*	*1.60*
0.02	Uniforme	22	25	22	22	20	23	24	23	22	20	*22.30*	*1.57*
0.05	Uniforme	25	23	22	25	23	24	25	23	25	26	*23.91*	*1.38*
0.075	Uniforme	24	30	27	26	31	30	22	22	26	28	*26.60*	*3.24*

Figura 50: risultati nella proprietà *Numero di allarmi* nei 10 test effettuati per ogni valore di p(er) nello scenario S2. Si riportano inoltre i risultati ottenuti con un errore che segue una distribuzione gaussiana.

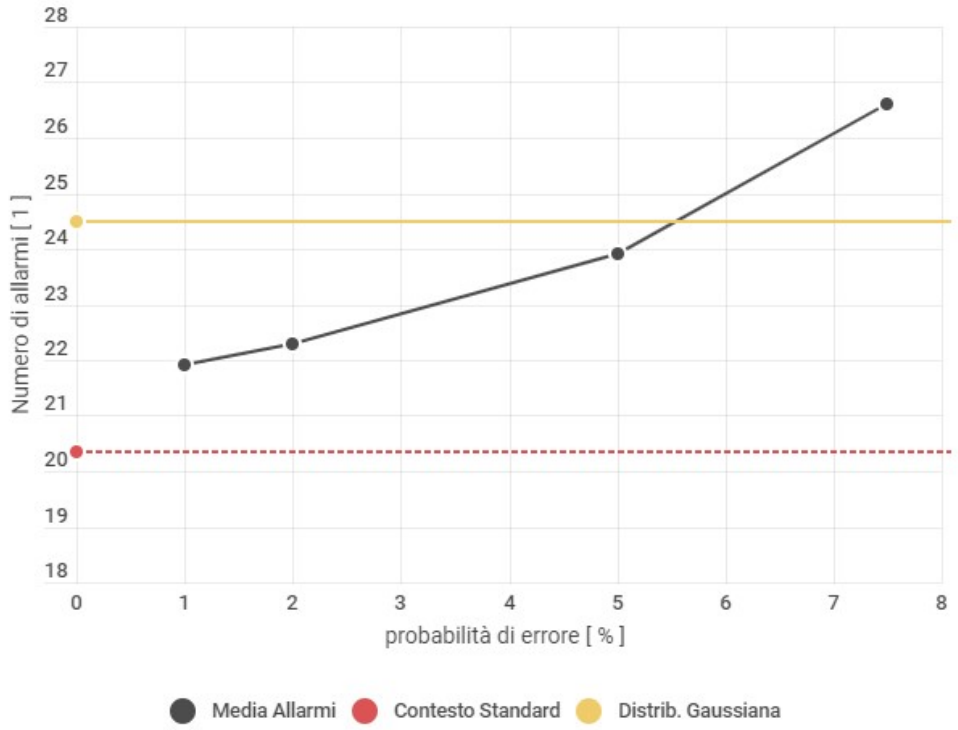

Figura 51: andamento nella proprietà *Numero di allarmi* al variare del parametro p(er), graficato in %. Si riportano per completezza anche l'analogo dato osservato nel contesto standard (in rosso) e nel caso di errore con distribuzione gaussiana (in giallo), i quali non variano con p(er).

Nel caso di errore seguente la distribuzione gaussiana è stato osservato un incremento sostanziale del numero di allarmi (24.5 contro i 20.36 del contesto standard) e tale scostamento è da attribuirsi ai valori di accuratezza dei sensori presi in esame. Se infatti è vero che, ad esempio, igrometro e termometro sono altamente precisi (con rispettivamente un errore *massimo* di ± 4 % e ± 0.4 °C), lo stesso non si può dire ad esempio del rilevatore di monossido di carbonio, che ha un'accuratezza addirittura pari a quasi la metà della soglia pericolosa istantanea, o del rilevatore di fumo e di allagamento la cui situazione è molto simile. Tra le cause di allarme che non erano presenti nel contesto standard originale, infatti, si sono registrati eccessi di CO e presenze di fumo non veritiere, che comportano dei **falsi positivi del sistema.**

Allo stesso modo, è facile immaginare che si siano verificati dei falsi negativi, in cui SafeHome sarebbe dovuto transitare nello stato *InAllarme* ma non lo ha fatto a causa dell'errore di misurazione. Tuttavia, il loro impatto è stato quantificato decisamente inferiore al numero di falsi positivi, in questo contesto, altrimenti il numero di allarmi si sarebbe mantenuto stabile. Una possibile causa per questo fenomeno è da attribuire all'andamento dei dati stessi, precisamente al fatto che in media il tempo in cui i dati fluttuano sufficientemente sotto la soglia limite è maggiore di quello in cui fluttuano appena sopra, e questo dà più possibilità di accadere ai falsi positivi. Non è stata comunque condotta un'indagine più approfondita sulle cause di questa anomalia, perché si sospetta appunto sia dipendente dal caso standard.

Per un errore con distribuzione uniforme, invece, sono stati presi in analisi alcuni parametri derivati dalla stessa formula per la binomiale mostrata nel caso 1, con l'aggiunta del valore p(er) = 0.075, corrispondente ad una probabilità di avvenimento di almeno un errore pari al 98,82%, contro ad esempio il 95% del valore p(er) = 0.05 già mostrato in *Figura 48*.
Anche in questo caso, analogamente al caso gaussiano, è stato osservato che il numero di falsi positivi supera quello di falsi negativi, probabilmente a causa degli stessi fattori.

Il **requisito non funzionale** che si associa a questo scenario riguarda sia la performance che l'affidabilità, e espresso similmente al caso 1, in cui però la differenza tra la probabilità di allarme osservata e quella standard dipende dal bilancio tra casi positivi e negativi derivati dagli errori di misura:

La probabilità di attivazione dell'allarme dovuta ad errori di misurazione non deve superare il valore limite X %.

Calcolata nello stesso modo, la probabilità viene ad esempio fornita per il caso gaussiano, pari a

$$p(A_{Eg}) = \frac{24.5}{409} = 0.0599$$

e per il caso uniforme con p(er = 0.075), pari a

$$p(A_{Eu}|p(er) = 0.075) = \frac{26.6}{409} = 0.0650$$

che fornirebbero seguendo il calcolo del caso 1 rispettivamente un aumento del 20.3% e del 30.5% rispetto al caso standard. Se questi valori fossero considerati troppo elevati, potrebbe significare che i sensori sono troppo imprecisi (nel caso gaussiano) o troppo proni ad errori sistematici (nel caso uniforme): una soluzione a questo problema potrebbe stare nella loro sostituzione.

Ritardo di openHAB (Scenario 3)

In questo scenario si vuole testare il caso in cui l'interfaccia remota abbia un certo **ritardo nella ricezione del valore della nuova misurazione** e nel conseguente controllo delle soglie di situazione indesiderata. Lo scenario è realistico e particolarmente significativo in quanto garantire l'affidabilità del sistema è una sfida frequentemente riscontrata nel processo di design e manutenzione di un software.
Siccome però il runtime di openHAB, per il caso d'uso dimostrativo presentato, è stato installato su un'istanza locale, non è possibile che si verifichino dei rallentamenti legati ai suddetti problemi: così, allo scopo di eseguire comunque questo scenario di test, il ritardo di comunicazione è stato iniettato nel codice

e implementato tramite un *thread* messo in *schedule* con un certo *delay*, avente come unico compito quello di effettuare la solita chiamata all'API di aggiornamento fornita da openHAB. Il delay di schedule del thread è stato preso a random seguendo una distribuzione uniforme entro un massimo che viene inserito in input dall'utente.
È stato poi introdotto **un valore limite**, pari a 4 secondi, oltre il quale la web app va virtualmente **in stato di timeout**. Nessun aggiornamento viene fatto in tale caso.

Durante l'esecuzione del sistema in questo scenario, potrebbe succedere che il ritardo di comunicazione avvenga proprio mentre si sta aggiornando un valore che avrebbe altrimenti provocato un allarme; se un altro valore (di un altro tipo di dato o sensore) oltre la propria soglia viene misurato durante questo periodo temporale, l'allarme scatterà ugualmente ma sarà causato dal secondo dato, mentre il primo andrebbe 'perso' (cioè, non sarebbe rilevato) perché il sistema si troverebbe già nello stato InAllarme e ci rimarrebbe per il periodo di reset (sempre pari a 10 secondi).
In assenza di ritardo, la stessa sorte sarebbe toccata invece al secondo dato.

Va da sé che in questo scenario esiste un caso analogo in cui è il secondo aggiornamento a subire un ritardo tale da essere effettivamente rilevato come potenziale situazione pericolosa, se nel frattempo è trascorso il reset time e il sistema è tornato nello stato Normale, e il numero di raggiungimenti dello stato InAllarme (la proprietà osservata) sale di uno.

Analogamente, potrebbe succedere che openHAB vada in timeout, quando il delay del thread supera il limite di 4 secondi, proprio mentre sta cercando di segnalare una misurazione oltre la soglia, e la proprietà osservata di nuovo subirebbe una modifica (in questo caso, scenderebbe di un allarme).

Non vi sarebbe alcuna differenza nella proprietà osservata invece nel caso in cui un allarme rimanesse stabile più tempo del normale in un valore oltre la soglia, perché per costruzione delle *rules* del sistema il controllo sulla situazione indesiderata viene fatto ad ogni aggiornamento del dato.

Il test è stato condotto nel contesto standard, e compiuto 10 volte per ogni diverso valore del parametro in input (*Max Delay*). I risultati sono riportati in *Figura 52* e graficati in *Figura 53*.

Sono stati scelti quasi solo valori oltre la soglia limite di timeout tranne per il primo test, in cui però la proprietà in assenza di timeout non ha esibito significative differenze.
Nonostante infatti le particolari situazioni sopra descritte potrebbero o non potrebbero essersi verificate, il numero totale di raggiungimenti dello stato *InAllarme* si è mantenuto sufficientemente simile al caso standard, scoraggiando l'ipotesi di condurre altri test con delay minori di 4 secondi.

max delay (ms)	1	2	3	4	5	6	7	8	9	10	Media	Dev. Std
3500	21	23	18	22	23	22	20	20	21	19	*20.9*	*1.66*
4010	22	18	22	18	22	20	24	23	24	25	*21.8*	*2.44*
4250	22	20	19	20	19	23	24	20	19	19	*20.5*	*1.84*
4750	21	19	19	16	21	19	18	21	19	22	*19.5*	*1.78*
5000	15	21	18	18	20	20	17	21	19	17	*18.6*	*1.86*
5500	16	18	16	17	17	20	17	17	19	18	*17.5*	*1.26*

Figura 52: risultati nel *Numero di Allarmi* dei 10 test effettuati nello scenario *Ritardo di openHAB*.
Si tenga presente che la soglia di *timeout* è stata impostata pari a 4000 ms.

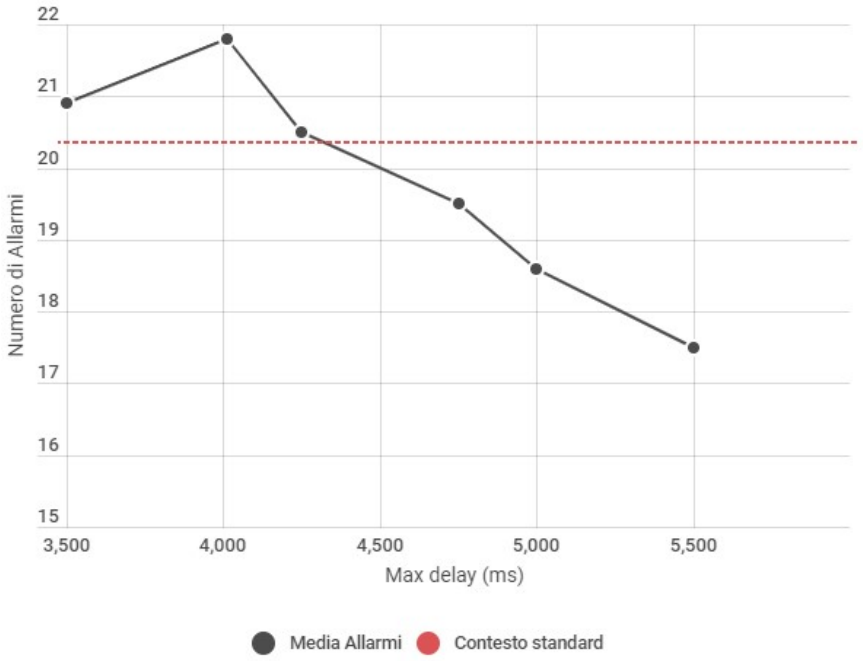

Figura 53: andamento della proprietà *Numero di allarmi* al variare del ritardo massimo impostato. Si riporta per confronto l'analoga proprietà nel contesto standard (in rosso).

Come previsto, si osserva una differenza sostanziale all'aumentare del delay massimo perché la distribuzione secondo la quale viene scelto ogni ritardo è uniforme; non sorprende invece che con un valore Max delay di 3500 ms, inferiore al timeout pari a 4000 ms, si sia verificata una variazione nel numero medio di allarmi molto piccola.

Il **requisito non funzionale** associato a questo ultimo caso riguarda ancora sia affidabilità che performance, e può essere espresso nuovamente in funzione della differenza tra probabilità di allarme dello scenario 3 e del contesto standard, con una formula simile a

La probabilità di attivazione dell'allarme dovuta al ritardo di aggiornamento del sistema non deve superare il valore limite di X %.

Al solito, come negli altri casi di test, si può attribuire l'intero scostamento di probabilità al fattore testato.

Viene fornita a titolo di esempio la probabilità calcolata per d = 5000 ms, pari a

$$p(A|d = 5000) = \frac{18.6}{409} = 0.0455$$

che seguendo il calcolo del caso 1, dà una diminuzione dell' 8.6%.
Se il cliente considerasse questo scostamento troppo elevato, significherebbe che le performance del sistema non sono sufficienti, e si dovrebbe quindi indagare sulle cause di tale ritardo; ad esempio, in un caso di sovraffollamento del server, si potrebbe risolvere migliorando quell'aspetto della soluzione.

8. Conclusioni

La soluzione presentata in questo lavoro di tesi rappresenta un modello del funzionamento di una tipica smart home, realizzata sulla base del noto caso di studio *SafeHome,* inventato a scopi didattici nel libro *Software Engineering: a Practitioner's Approach* [1], e sviluppata utilizzando il software open source di domotica *openHAB* [5].
Nella produzione della soluzione si sono seguiti tutti i processi tipici dell'ingegneria del software riguardo alle fasi di design, elicitazione dei requisiti ed architettura di un applicativo.

Il sistema così prodotto è stato poi utilizzato come ambiente virtuale isolato e controllato nel quale sono state modellate ed inserite fonti di incertezza estremamente comuni in controparti reali dell'ambiente simulato e ne è stato osservato il comportamento in risposta a tali situazioni non deterministiche.

Nello specifico sono stati creati tre scenari di test, nei quali sono state introdotte rispettivamente le eventualità di fallimento di un sensore dovuto a malfunzionamenti, di errori nella misurazione e di ritardo nell'aggiornamento dell'applicativo remoto. Questi tre esperimenti sono stati condotti in un contesto standard specifico, accuratamente creato per permettere la possibilità ai suddetti eventi di verificarsi.
È stato studiato quindi il comportamento del sistema, in termini di numero di raggiungimenti dello stato *InAllarme* (cioè, il numero di attivazioni del sistema di allarme dell'abitazione), al variare di precisi valori di probabilità di avvenimento dell'anomalia (per gli scenari 1 e 2) e del ritardo massimo dell'applicazione (per lo scenario 3): i valori in output dei sensori sono stati confrontati con delle soglie predeterminate, oltre le quali fonti spesso autorevoli nel campo della sanità classificano la situazione come potenzialmente pericolosa per la salute delle persone, e quindi degli ipotetici utenti del sistema.

I risultati, anche nell'ambiente virtuale, sono in linea con le aspettative ricavate dall'esperienza nelle analoghe situazioni reali: ad esempio, dal grafico in *Figura 47* è risultato esserci una possibile relazione di proporzionalità diretta tra la probabilità per un sensore di fallire a causa di un malfunzionamento e il numero di attivazioni dell'allarme della smart home, ma in *Figura 49* si può notare che l'analoga relazione attesa con il numero di sensori malfunzionanti non è stata riscontrata - questo perché è presente piuttosto una relazione con il *momento di fallimento*, perché inibire indefinitamente le misurazioni all'inizio dell'esperimento causa più scostamento nella proprietà osservata che lo stesso evento avvenuto a fine scenario.

Riguardo all'incertezza dovuta ad errori di misurazione, è possibile notare dai risultati in *Figura 51* che nel caso in cui l'errore di misura era legato all'accuratezza dei sensori (che è stata presa da specifiche tecniche o datasheet di dispositivi reali), cioè nel caso chiamato *gaussiano*, si è verificato un incremento sostanziale nel numero di allarmi rispetto al contesto standardizzato privo di errori (+20.3 %) che si attribuisce alla poca accuratezza dei misuratori presi in esame (di quantità di monossido di carbonio, di presenza di fumo ed allagamento).
Per quanto concerne lo stesso studio condotto questa volta su errori di misurazione *sistematici*, in cui cioè i sensori hanno occasionalmente fornito valori completamente sbagliati - eventi legati ad una probabilità di avvenimento, immessa in input - denominato *uniforme*, ci si è invece riallineati con le aspettative, con un andamento della proprietà osservata che si sospetta sia leggermente parabolico al variare della suddetta probabilità (visibile anch'esso in *Figura 51*).

Infine, nell'ultimo caso di test è stato analizzato il comportamento del sistema in merito a situazioni di ritardo nell'aggiornamento dell'interfaccia remota, nei quali la lunghezza della finestra temporale di update è stata simulata con valori presi seguendo una distribuzione uniforme entro un massimo, variabile, prestabilito, ed è stato riscontrata l'attesa dipendenza tra tale valore massimo e il numero di attivazioni del sistema di allarme in *Figura 53*.

A seguito di queste rilevazioni è stato poi proposto per ciascuno scenario di test un requisito non funzionale associato che in un'applicazione reale del software sarebbe di estremo interesse per il cliente o utente finale per garantire la qualità della soluzione proposta.

L'ambiente simulato, nonché l'intero codice, viene fornito per scopi didattici e di ricerca al seguente indirizzo web: *https://github.com/aleverga10/SafeHome* .

9. Lavoro futuro

In questo elaborato si è seguito per intero il processo di design del sistema Safehome e il suo conseguente utilizzo come ambiente isolato e controllato per la valutazione di incertezze, ma esistono ancora degli aspetti sui quali è possibile agire, in futuro.

Una prima direzione sarebbe quella di sostituire, con un adattamento minimo del codice, una delle due simulazioni con la sua controparte reale ed osservare il conseguente comportamento del sistema: ad esempio, sostituendo Ambiente con un servizio che fornisca in qualche modo dati da una vera smart home, si potrebbe valutare il comportamento del sistema Sensori, conducendo gli stessi tre tipi di test effettuati nella simulazione e osservando la differenza nel numero di allarmi prodotti.

Allo stesso modo, potrebbe essere interessante definire nuove situazioni incerte presenti nei sistemi reali e iniettare accuratamente il loro modello come nuovi scenari di test da eseguire nel sistema controllato, così da avere una verifica sperimentale della validità del modello dell'incertezza sviluppato.

Un'altra strada da percorrere potrebbe invece essere quella di utilizzare l'ambiente simulato controllato come si è fatto finora per i tre scenari di test proposti, ma affiancare SafeHome ad un sistema *self-adaptive* con un'intelligenza artificiale in grado di accorgersi dell'avvenire di situazioni indesiderate e riconfigurare i propri parametri in risposta, ad esempio aggiungendo nuove *rules* in openHAB o modificando automaticamente quelle già presenti.

Infine, si potrebbe propendere per cercare un adattamento del caso di studio ancora più fedele di quello proposto qui, o più moderno, per esempio implementando un sistema di telecamere di cui questa soluzione è sprovvista oppure installando SafeHome su un server dedicato.

Appendice A: Ambiente hardware e software utilizzati

La versione di openHAB utilizzata durante tutto questo lavoro di tesi è la *stable release 2.5.2* [53], seguendo i tutorial ufficiali [6] e [47].

L'intero sistema SafeHome è stato scritto in Java.
È stata utilizzata la release 8 della versione OpenJDK (*open source Java Development Kit*) con il bundle *JavaFX* necessario per lo sviluppo di interfacce grafiche.
La build di OpenJDK utilizzata è Zulu, precisamente nella versione Zulu 8.44.0.13 con la versione OpenJFX 8 come build del bundle FX, che è fornita da Azul Systems [54].

Come ambiente di sviluppo IDE è stato usato Visual Studio Code (nella versione 1.41), con l'Extension Pack apposito per Java [55].
Apache Maven 3.6.3 [56] è stato utilizzato per gestire l'intera build del progetto tramite il POM (*Project Object Model*) creato dal software in automatico, il quale ha installato i seguenti plugin (riportati con la dicitura [*groupID*] - [*artifactID*] - [*version*]) :
org.openjfx - javafx.maven.plugin - 0.0.4;
org.apache.maven.plugins - maven-checkstyle-plugin;
org.jacoco - jacoco.maven.plugin - 0.8.1;

Oltre a questi, installati automaticamente, sono state utilizzate le seguenti **dipendenze** di JUnit, PubNub e slf4j, dipendenza transitiva richiesta da PubNub (anche qui, [*groupID*] - [*artifactID*] - [*version*]):
junit - junit - 4.12;
com.pubnub - pubnub.gson - 4.31.0;
org.slf4j - slf4j-jdk14 - 1.7.25;

Allo scopo di costruire la GUI, sono state installate le seguenti **dipendenze** di JavaFX:
base, controls, fxml, graphics, media, swing, web, tutte con groupID *org.openjfx*, versione 14
Per la costruzione di una GUI nativa e responsiva è stata usata la libreria JFoenix [57], che implementa il material design di Google usando componenti Java, e il software Scene Builder, prodotto open source fornito da Gluon [58], che permette il drag and drop dei componenti di JavaFX (e di JFoenix, una volta importato il .jar). Le icone utilizzate sono state prese dalla libreria fontawesome-fx [59], i cui jar sono stati importati nelle cartelle */lib* e nei file *.classpath* dei progetti Sensori e Ambiente.

Infine GitHub [60] è stato usato per **la condivisione del codice del sistema SafeHome**, che è reperibile all'indirizzo *https://github.com/aleverga10/SafeHome*.

Per la stesura di questo documento sono stati usati i seguenti software:
Google Docs, per la scrittura;
draw.io, per disegnare i modelli UML;
infogram.com, per realizzare ogni tabella e grafico;
sciweavers.org, per le formule rappresentate in forma grafica.

10. Bibliografia e sitografia

[1] Pressman R. S., *Software Engineering. A Practitioner's Approach*, New York, McGraw-Hill, 2010, seventh edition

[2] RS Pressman & Associates, Inc., http://www.rspa.com
(ultima visita: Maggio 2020)

[3] Google Books reviews, https://books.google.it
(ultima visita: Maggio 2020)

[4] openHAB Foundation, *Our Mission*, https://www.openhabfoundation.org/
(ultima visita: Maggio 2020)

[5] openHAB Foundation, *Who we are*, https://www.openhab.org/about/who-we-are.html
(ultima visita: Maggio 2020)

[6] openHAB Foundation, *Documentation*, https://www.openhab.org/docs/
(ultima visita: Maggio 2020)

[7] Franchitti J., *Software Engineering Session 6 Slides – Main Theme Detailed-Level Analysis and Design*, Computer Science Department Courant Institute of Mathematical Sciences, New York

[8] Southern Illinois University, *Lecture 8*, School of Computing, Illinois

[9] Kahn F., *Case Study: Safe Home Security System*, Interdisciplinary Information Studies, University of Tokyo, Tokyo

[10] Lethbridge T. C., Pressman R. S., *Software Engineering: a Practitioner's Approach 7th Edition Pressman Solutions Manual*

[11] SafeHome Products, *Case Study. Overview*, http://www.safehomeassured.com/casestudy/index.htm
(ultima visita: Maggio 2020)

[12] Pressman R. S., Lowe D., *Web Engineering. A Practitioner's Approach*, New York, McGraw-Hill, 2008

[13] Alshammari N., Alshammari T., Sedky M., Champion J., Bauer C., *OpenSHS: Open Smart Home Simulator*, in "Sensors 2017", 2017, 1003

[14] Bouchard K., Ajroud A., Bouchard B., Bouzouane A., *SIMACT: A 3D Open Source Smart Home Simulator for Activity Recognition*, in "Advances in Computer Science and Information Technology", 2010, 524-533

[15] Real Games Unipessoal Lda, *Home I/O*, https://realgames.co/home-io/
(ultima visita: Maggio 2020)

[16] Vasilateanu A., Popescu A., Cergan A., Goga N., *Smart home simulation system*, 2016 IEEE International Symposium on Systems Engineering (ISSE), 2016

[17] Fomin S. P., Orlov A. A., *Simulation Modeling of "Smart Home" Operations using the Data on a Man's Position*, disponibile online, 2015

[18] MIT Scratch Team, *Scratch*, https://scratch.mit.edu/ (ultima visita: Maggio 2020)

[19] Alhafidh B. M. H., Allen W., *Design and Simulation of a Smart Home managed by an Intelligent Self-Adaptive System*, in "International Journal of Engineering Research and Applications", 2016

[20] Kouda S., Dibi Z., Faycal M., *Modeling of a Smart Humidity Sensor*, disponibile online, 2008

[21] Xiumei X., Jinfeng P., *The simulation of temperature and humidity control system based on PROTEUS*, in "2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC)", Jilin, 2011, pp. 1896-1898,

[22] The MathWorks, Inc., *Simulink*, https://www.mathworks.com/products/simulink.html (ultima visita: Maggio 2020)

[23] Pressman R. S., *Software Engineering. A Practitioner's Approach*, New York, McGraw-Hill, 2010, seventh edition, p. 130

[24] Good Home Automation, *Different Kinds of Smart Home Sensors: The Definitive List*, https://goodhomeautomation.com/different-kinds-of-smart-home-sensors/ (ultima visita: Maggio 2020)

[25] Cleland-Huang J., Settimi R., Zou X., Solc P., *The Detection and Classification of Non-Functional Requirements with Application to Early Aspects*, in "14th IEEE International Conference on Requirements Engineering (RE 2006)", September 2006, Minnesota

[26] Old Dominion University, *Lecture 3: Non Functional Requirements*, Department of Computer Science, Norfolk, Virginia

[27] Odeh Y., Odeh M., A New Classification of Non-Functional Requirements for Service-Oriented Software Engineering, in "Naif Arab University for Security Sciences", 2011, Riyadh, Saudi Arabia

[28] Pressman R. S., *Software Engineering. A Practitioner's Approach*, New York, McGraw-Hill, 2010, seventh edition, p. 131

[29] Pressman R. S., *Software Engineering. A Practitioner's Approach*, New York, McGraw-Hill, 2010, seventh edition, p. 133

[30] Pressman R. S., *Software Engineering. A Practitioner's Approach*, New York, McGraw-Hill, 2010, seventh edition, p. 129

[31] IEEE Computer Society, *29148-2018 - ISO/IEC/IEEE International Standard - Systems and software engineering -- Life cycle processes -- Requirements engineering*, 2018

[32] Weather Station Advisor, *Best Hygrometer Reviews in 2020: We Found the Most Accurate Digital Hygrometer*, 2020, https://www.weatherstationadvisor.com/best-hygrometer-reviews/ (ultima visita: Maggio 2020)

[33] Airthings, *Airthings Wave Plus. Operating Manual*, 2018, version 4

[34] Kidde, Carrier Company, *Nighthawk™ AC Plug-in Operated Carbon Monoxide Alarm with Digital Display. Technical Specifications, https://www.kidde.com/home-safety/en/us/products/fire-safety/co-alarms/kn-copp-3/* (ultima visita: Maggio 2020)

[35] Smart Air, *How accurate are common particle counters?*, 2016, https://smartairfilters.com/en/blog/how-accurate-are-common-particle-counters-comparison-test/ (ultima visita: Maggio 2020)

[36] Bukowski R. W, Peacock R. D., Averill J. D., Cleary T. G., Bryner N. P., Walton W. D., Reneke P. A., Kuligowski E. A., *Performance of Home Smoke Alarms. Analysis of the Response of Several Available Technologies in Residential Fire Settings*, in "NIST Techincal Note", 2008, p. 29

[37] Reolink, *How to Configure Motion Detection Sensitivity for Reolink Cameras*, 2020, https://support.reolink.com/hc/en-us/articles/360006992493-How-to-Configure-Motion-Detection-Sensitivity-for-Reolink-Cameras (ultima visita: Maggio 2020)

[38] OMRON, *Technical Explanation for Liquid Leakage Sensors*

[39] Healthline Media, Red Ventures, *Hot and Cold: Extreme Temperature Safety*, https://www.healthline.com/health/extreme-temperature-safety (ultima visita: Maggio 2020)

[40] Gilmore C. P., *More Comfort for Your Heating Dollar*, in "Popular Science", 1972, p. 99

[41] ASHRAE, Ventilation for Acceptable Indoor Air Quality (ANSI Approved). *Standard 62.1-2019*, 2019

[42] National Comfort Institute, *Carbon Monoxide Levels & Risks*, https://www.myhomecomfort.org/wp-content/uploads/2015/09/CO_Levels_Risk_Chart.pdf (ultima visita: Maggio 2020)

[43] The World Air Quality Project, *Revised PM2.5 AQI breakpoints*, 2013 https://aqicn.org/faq/2013-09-09/revised-pm25-aqi-breakpoints/ (ultima visita: Maggio 2020)

[44] TechHeroes Inc., *The Eve Room, 2nd Gen: Comparison and Review. Air Quality Scale*, 2019 http://techheroesinc.blogspot.com/2019/02/the-eve-room-2nd-gen-comparison-and.html (ultima visita: Maggio 2020)

[45] PubNub Inc., *PubSub Messaging APIs*, https://www.pubnub.com/features/pub-sub-messaging/ (ultima visita: Maggio 2020)

[46] Callmebot.com, *CallMeBot API*, https://www.callmebot.com/
(ultima visita: Maggio 2020)

[47] openHAB Foundation, *Installation Overview*, https://www.openhab.org/docs/installation/
(ultima visita: Maggio 2020)

[48] openHAB Foundation, *Icons*, https://www.openhab.org/docs/configuration/iconsets/classic/
(ultima visita: Maggio 2020)

[49] openHAB Foundation, *openHAB REST API*, http://localhost:8080/doc/index.html#/

[50] Autili M., Cortellessa V., Di Ruscio D., Inverardi P., Pelliccione P., Tivoli M., *Integration architecture synthesis for taming uncertainty in the digital space*, in "Computer Science vol. 7539", Springer, Heidelberg 2012

[51] Funtowicz S., Ravetz J., *Uncertainty and Quality in Science for Policy*, in "Springer", 1990

[52] Mirandola R., Perez-Palacin D., *Uncertainties in the modeling of self-adaptive systems: A taxonomy and an example of availability evaluation,* in "Conference: Proceedings of the 5th ACM/SPEC international conference on Performance engineering", 2014

[53] openHAB Foundation, *Download openHAB*, https://www.openhab.org/download/
(ultima visita Marzo 2020)

[54] Azul Systems Inc., Zulu Community Downloads: Open Source Java Built by Azul,
https://www.azul.com/downloads/zulu-community
(ultima visita Marzo 2020)

[55] Microsoft, *Java in Visual Studio Code*, https://code.visualstudio.com/docs/languages/java
(ultima visita Marzo 2020)

[56] The Apache Software Foundation, *Download*, https://maven.apache.org/download.cgi
(ultima visita Marzo 2020)

[57] JFoenix, *Documentation*, http://www.jfoenix.com/documentation.html
(ultima visita Marzo 2020)

[58] Gluon, *Scene Builder*, https://gluonhq.com/products/scene-builder/
(ultima visita Marzo 2020)

[59] FontIcons Inc., *Font Awesome*, https://fontawesome.com/?from=io
(ultima visita Marzo 2020)

[60] GitHub Inc. *GitHub*, https://github.com/