VIRTUAL SMART HOME

Progetto INFO3B A.A. 2018/2019

Alberto Arrigoni, Daniele Bosc, Rita Pedercini

Sommario

Introduzione

Virtual Smart Home project fa parte di un progetto avente come scopo la creazione di una smart home virtuale a cui è applicato un controllo adattativo.
Tale progetto si basa quindi sulla pratica della Smart Home automation, cioè sull'utilizzo di devices connessi ad Internet per il controllo remoto e automatico degli elettrodomestici.
Il nostro contributo a tale progetto consiste nella creazione di una casa virtuale i cui devices non sono presenti fisicamente, ma per i quali viene simulato il comportamento. Tale casa virtuale è inoltre controllata per mezzo di un Management System, cioè è stata effettuata l'implementazione di un controllore di base (managed system) e di un controllore più avanzato (managing system).
In generale i Management Systems per smart home hanno stili architetturali simili, essi sono caratterizzati da un gateway (hub) che svolge il compito di mediatore tra l'applicazione in cloud e i devices della smart home. Il gateway ha quindi il compito di raccogliere dati dai diversi devices della casa e trasferirli all'applicazione di controllo; per tale motivo esso conosce una vasta varietà di protocolli di comunicazione.
L'utente ha la possibilità di controllare il sistema per mezzo di una semplice interfaccia disponibile sia per smartphones che tablets.
Inoltre, molte piattaforme di home automation offrono la possibilità all'utente di creare delle semplici regole personalizzate (Event-Condition-Actions rules). Ogni regola, azionata dal verificarsi di specifici eventi e in base allo stato del sistema, consente l'invio di specifici comandi e la modifica dello stato dell'ambiente.
In particolar modo, il Managed System implementato si basa su tre principali obiettivi: controllo del sistema di riscaldamento allo scopo di massimizzare il comfort dell'utente, controllo di qualità dell'aria indoor e controllo antincendio. È tuttavia difficile l'implementazione di logiche di controllo che permettano di massimizzare il comfort in ogni situazione e cioè di garantire la massima efficienza in tutti gli obiettivi proposti. Infatti, si possono verificare dei conflitti tra le varie regole di controllo; per esempio questo può accadere se requisiti di riscaldamento e requisiti di qualità dell'aria si riferiscono ai medesimi devices e impongono delle azioni che sono in contrasto tra loro.
Proprio allo scopo di gestire nel modo corretto tali conflitti, evitare malfunzionamenti del sistema di controllo o situazioni indesiderate, il Managed System è affiancato da un Managing System. Quest'ultimo realizza su un meccanismo self-adaptive per le operazioni e i processi più avanzati.
Nel nostro progetto è stato creato un Managing System molto semplice che si occupa solamente di risolvere alcuni controlli avanzati, ma non effettua la gestione dei conflitti vera e propria.
Il sistema di controllo implementato è costituito quindi da due layers: il Managed Layer, per operazioni di controllo di base, e il Managing Layer, per operazioni di controllo avanzate.

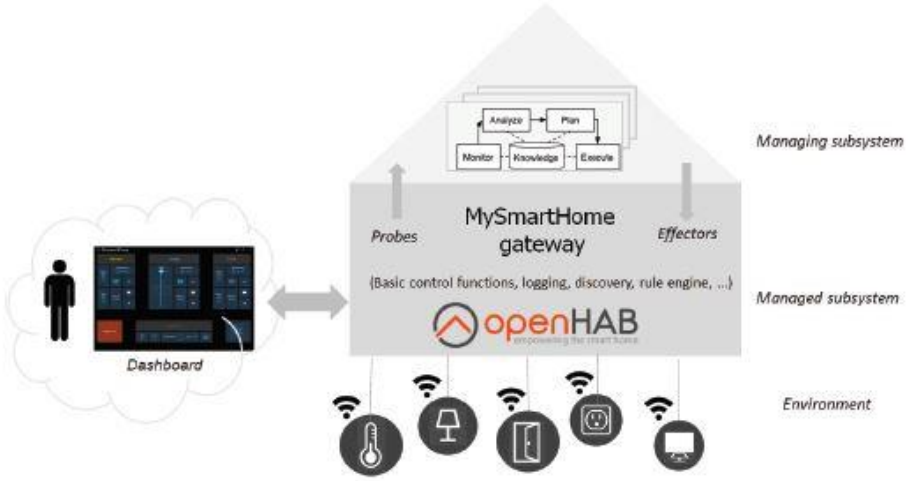


Figura 1: Structural architecture of the smart home control system

Sia il Managed che il Managing Layer sono stati implementati utilizzato OpenHAB[1], una piattaforma open source (scritta in Java con un'architettura OSGi per la modularizzazione) per home automation.
Esso consente di connettere diversi devices e fornisce diverse tipologie di interfaccia utente, attraverso le quali gli utenti possono modificare lo stato dei devices.
Oltre alla realizzazione del Management system, il nostro contributo al progetto comprende la creazione di un Gateway/Broker che connetta il Management system realizzato in OpenHAB con un controllore avanzato (Managing system) realizzato per mezzo di MAPE-K feedback loops.
Tale componente è stata realizzata in Java, essa comunica con OpenHab tramite Rest API ed è utilizzabile da parte di un generico Managing system per semplice invocazione di metodo.

[1] OpenHAB: https://www.openhab.org/

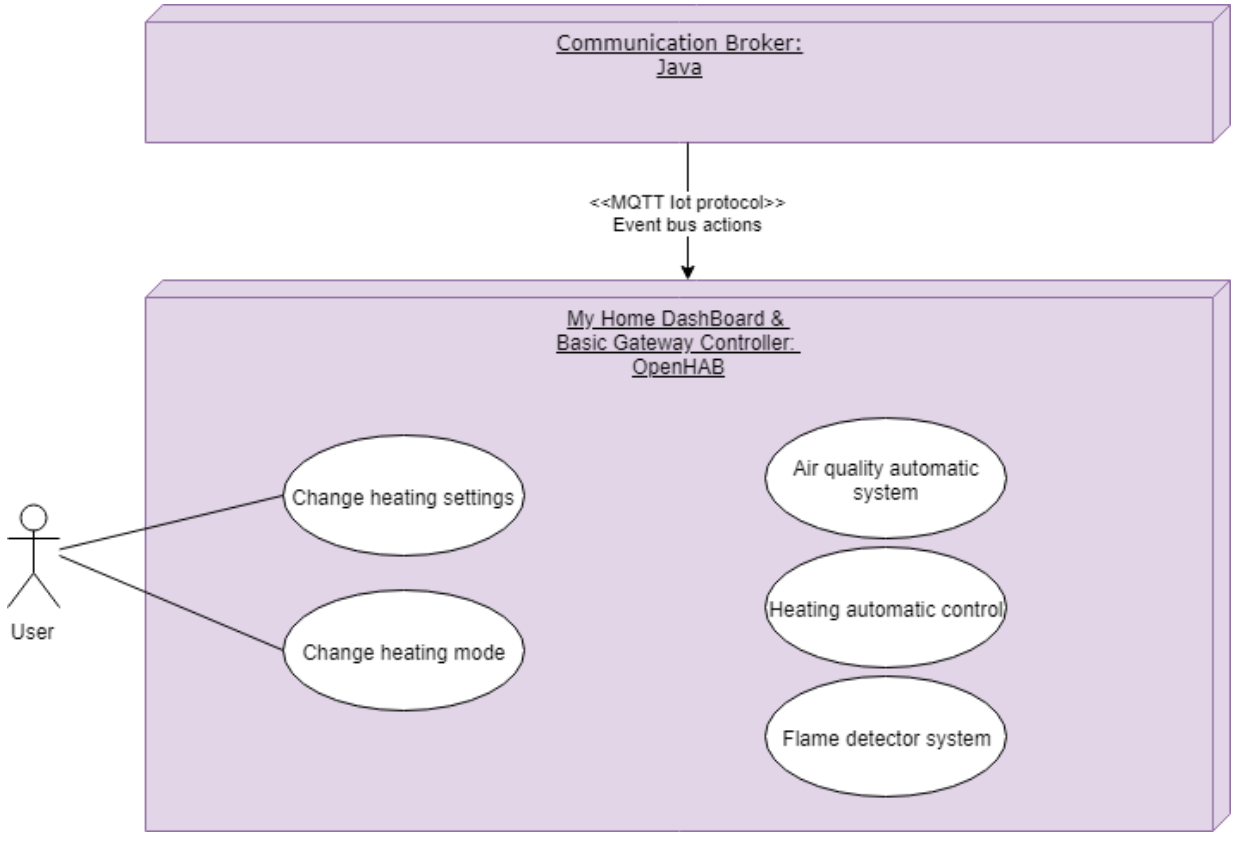
Iterazione 0

Il progetto Virtual Smart Home si basa sulla realizzazione di un sistema di controllo di base (Managed System) per mezzo della piattaforma OpenHab. Inizialmente inoltre avevamo pensato di gestire la simulazione del comportamento dell'ambiente per mezzo del tool OpenSHS (Open Smart Home Simulator).
Si tratta di una piattaforma 3d per la simulazione di smart home che consente la generazione di datasets. OpenSHS offre quindi un'opportunità nel campo dell'Internet of Things.
Inoltre, nella versione iniziale del progetto era prevista la realizzazione di un Communication Broker che doveva permettere il trasferimento del dataset di eventi, prodotto da OpenSHS (espresso come file .CSV), all'interno del bus di eventi di OpenHab per mezzo del protocollo MQTT.
I requisiti funzionali iniziali erano quindi:

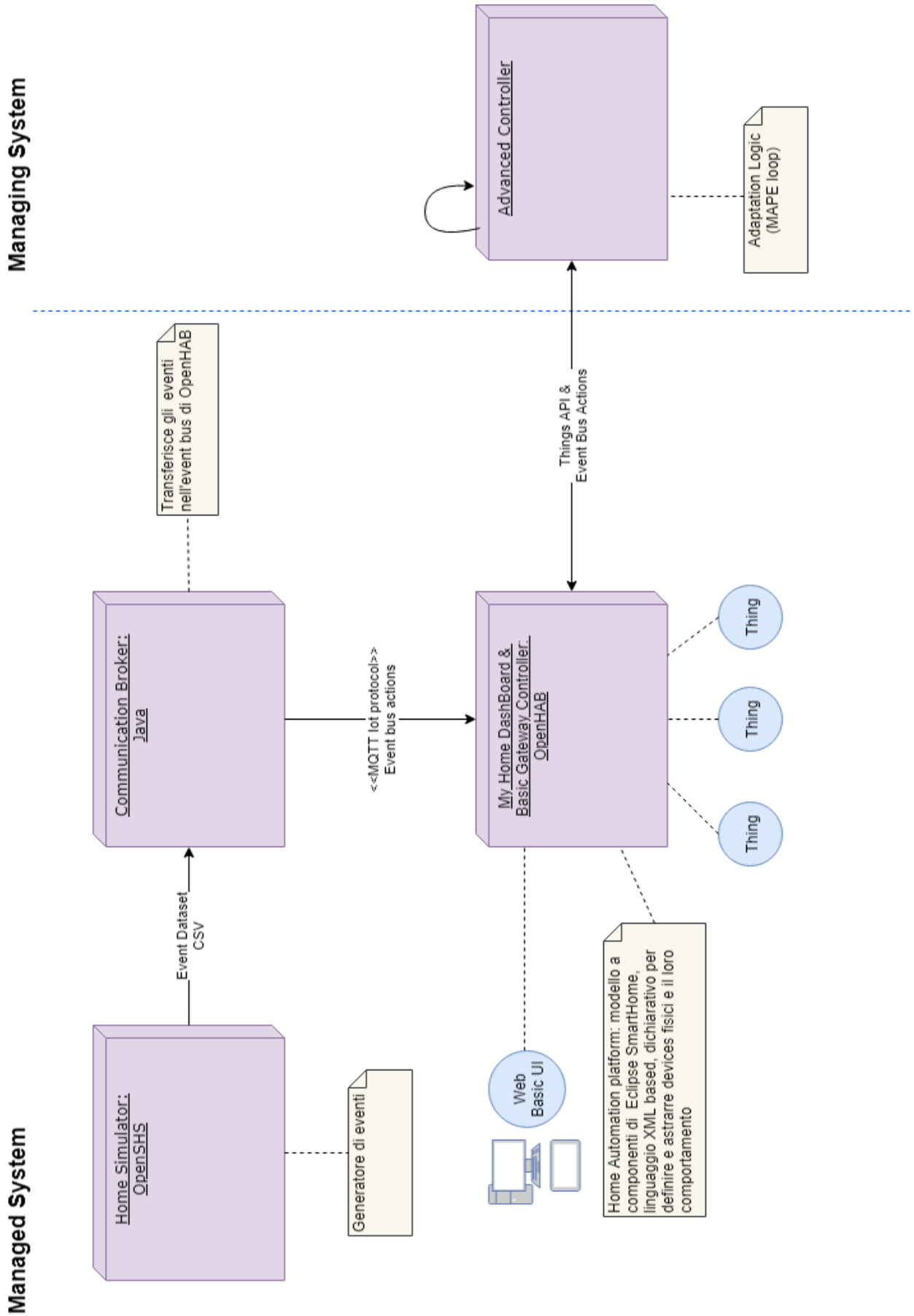
- Implementazione di un sistema di controllo di base per la gestione di un sistema di riscaldamento che massimizzi il comfort, di un sistema di qualità dell'aria e di un sistema di controllo di incendio;
- Realizzazione di un simulatore della casa virtuale coerente con il modello su cui si basa il controllo effettuato tramite OpenHab;
- Realizzazione di un Communication Broker che inserisca gli eventi generati da OpenSHS nell'event bus di OpenHab.

1.1 Use Cases

I casi d'uso iniziali possono essere riassunti quindi con il seguente schema:



1.2 Topology Diagram



Iterazione 1

L'iterazione 1 dello sviluppo del progetto Virtual Smart Home è incentrata sull'implementazione del sistema di controllo della Virtual Smart Home per mezzo di OpenHab, in particolar modo durante quest'iterazione abbiamo sviluppato sia il sistema di controllo di base che quello avanzato. Abbiamo inoltre gestito la simulazione dell'ambiente da controllare.

2.1 Smart Home environment description

Per mezzo della piattaforma OpenHab è stato quindi possibile rappresentare per mezzo di Items [2] una casa virtuale. La virtual smart home utilizzata in questo progetto è costituita da due piani. Ogni piano ha un termostato smart che riceve informazioni da diversi sensori di temperatura distribuiti tra le diverse stanze del piano (almeno uno per stanza) e da valvole termostatiche smart del sistema di riscaldamento presenti nelle stanze (almeno una per stanza). Ci sono inoltre sensori di qualità dell'aria, devices per l'apertura delle finestre e sensori antifumo (almeno uno per piano).

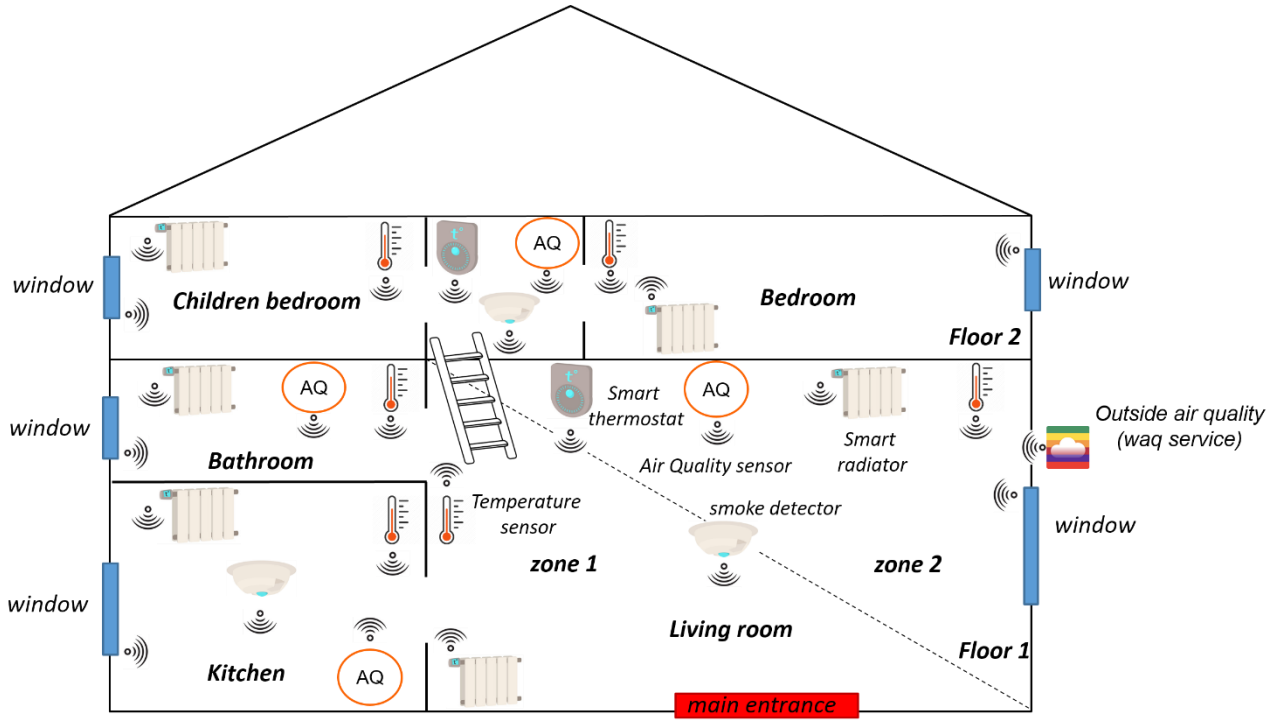
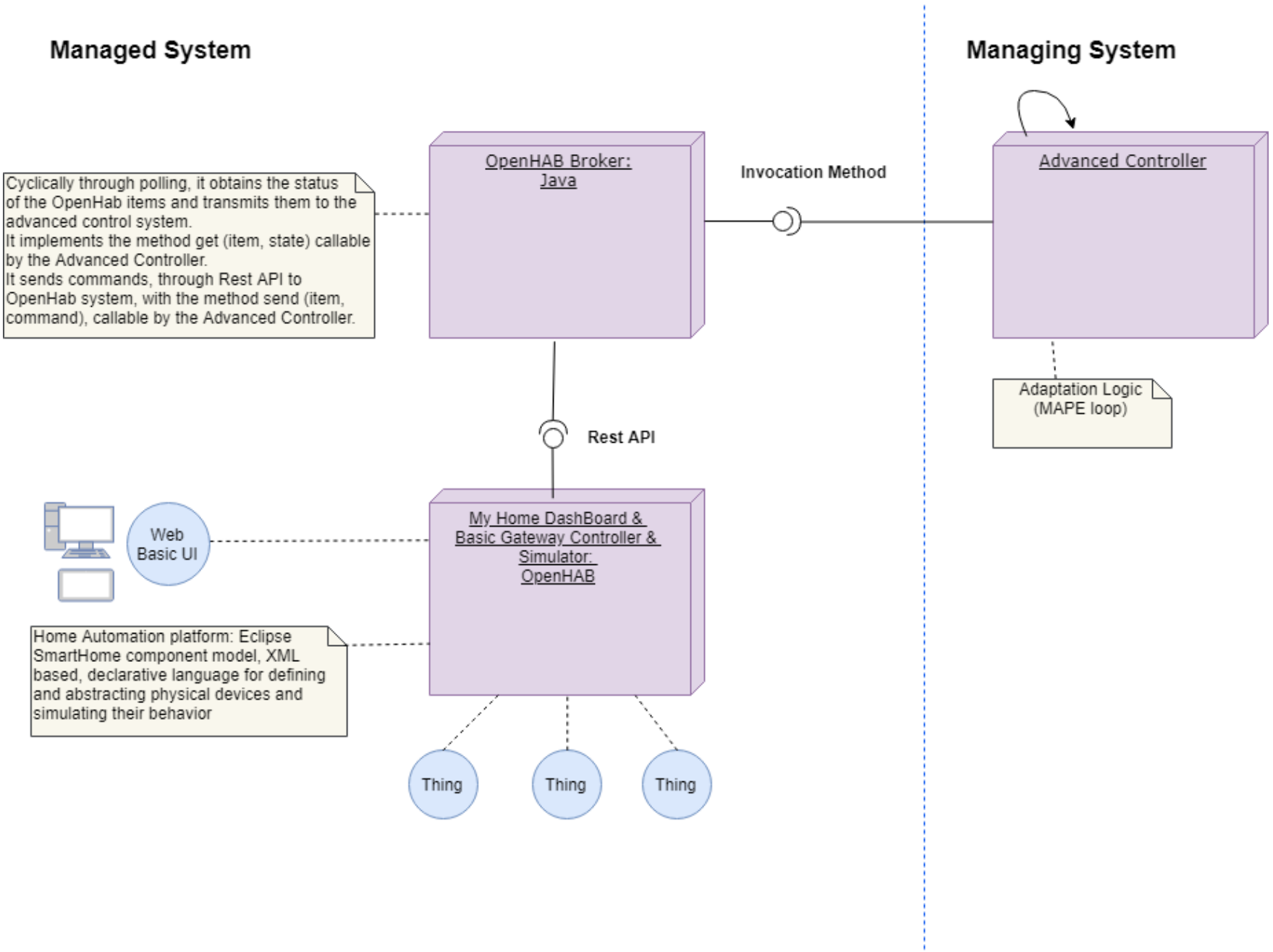


Figura 2: Virtual Smart Home

[2] Items: rappresentano il layer virtuale, sono quindi la virtualizzazione delle Things, cioè del mondo fisico, dei devices. Per maggiori informazioni: https://www.openhab.org/docs/concepts/items.html

2.2 Deployment Diagram

Durante la fase iniziale del progetto abbiamo effettuato delle modifiche all'architettura del progetto e di conseguenza anche ai requisiti funzionali su cui si basa.
In particolar modo dato che la simulazione dell'ambiente, necessario per lo sviluppo di Virtual Smart Home, non necessita di un simulatore come OpenSHS, il quale è principalmente utilizzato per la simulazione del funzionamento di motion sensors e quindi la presenza di persone nella casa, situazioni non di interesse per l'attuale progetto. Abbiamo deciso di simulare il comportamento dell'ambiente, che comprende principalmente la variazione di temperatura, la variazione di qualità dell'aria nelle varie stanze della casa ed eventi particolari quali la presenza di un incendio, utilizzando anche in questo caso il sistema di regole di OpenHab.
Il progetto Virtual Smart Home si basa quindi sulla realizzazione di un sistema di controllo costituito da un controllore di base e da un sistema di controllo avanzato oltre che l'implementazione di un simulatore dell'ambiente della virtual smart home per mezzo della piattaforma di OpenHab.
Un altro requisito del progetto riguarda inoltre l'implementazione di un OpenHab Broker in Java che permetta la comunicazione tra il sistema di controllo realizzato in OpenHab e un controllore avanzato realizzato per mezzo di mape loop.


Managed System
Managing System
OpenHAB Broker:
Java
Invocation Method
Advanced Controller
Cyclically through polling, it obtains the status of the OpenHab items and transmits them to the advanced control system.
It implements the method get (item, state) callable by the Advanced Controller.
It sends commands, through Rest API to OpenHab system, with the method send (item, command), callable by the Advanced Controller.
Adaptation Logic
(MAPE loop)
Rest API
Web
Basic UI
My Home DashBoard &
Basic Gateway Controller &
Simulator:
OpenHAB
Home Automation platform: Eclipse SmartHome component model, XML based, declarative language for defining and abstracting physical devices and simulating their behavior
Thing
Thing
Thing

2.3 Requisiti Funzionali

I requisiti funzionali inerenti all'implementazione del sistema di controllo per mezzo di OpenHab sono suddivisibili tra quelli che riguardano il Managed System, quelli del Managing System e i requisiti per la simulazione del comportamento dell'ambiente.

2.3.1 Managed System requirements (basic control functions)

I requisiti funzionali del Managed System sono:

- Modifica target del termostato al cambio della modalità: alla richiesta di cambiare modalità del termostato (sono presenti 4 modalità: normal mode, manual mode, holiday mode e stop mode) viene inviato ad esso il relativo target;
- Modifica target termostato: in orari prestabiliti il target associato allo stato del termostato "normal mode" viene modificato sulla base delle preferenze impostate dall'utente;
- Modifica impostazioni utente: l'utente può modificare le temperature target di ogni piano e per ogni modalità presente;
- Implementazione di un sistema di riscaldamento di base: in ogni piano della casa vengono accesi o spenti i radiatori confrontando la temperatura media sul piano e l'attuale target del termostato;
- Chiusura delle finestre: l'utente può impostare la chiusura forzata delle finestre per un certo periodo di tempo;
- Impostazioni utente per apertura finestre: l'utente può impostare un periodo di tempo in cui mantenere aperte le finestre;
- Controllo incendio: in caso di attivazione del sensore di fumo viene attivato l'allarma di incendio.

2.3.2 Managing System requirements

I requisiti funzionali del Managing System sono:

- Raggiungimento del target del termostato in ogni stanza della casa: sulla base delle temperature impostate dall'utente e in base ai diversi momenti della giornata vengono attivati e disattivati i radiatori in base all'output dei sensori di temperatura allo scopo di raggiungere lo specifico target;
- Apertura delle finestre in caso di aria interna non buona: sulla base dei parametri in output del sensore di qualità dell'aria indoor viene determinato attraverso un'opportuna scala di valori se la qualità dell'aria interna è buona, qualora essa non lo fosse e l'aria esterna risulti buona (controllo effettuato sulla base dei sensori di qualità dell'aria outdoor) verranno aperte le finestre;
- Chiusura automatica delle finestre se l'aria esterna non è buona;
- Controllo sensore di temperatura non funzionante: in caso un sensore di temperatura non funzioni, il controllo di incendio e il sistema di riscaldamento fanno riferimento all'ultima temperatura registrata;

- Controllo di incendio sulla base dei sensori di temperatura: qualora un sensore di temperatura riporti una misura superiore ai 45°C viene attivata l'allarme di incendio.

2.3.3 Requisiti per la simulazione del comportamento d'ambiente

La simulazione dell'ambiente è effettuata utilizzando sempre le rules di OpenHab, in particolar modo tali requisiti di simulazione possono essere suddivisi in due gruppi: requisiti per la simulazione di base e requisiti per simulazione di eventi particolari.
I requisiti per la simulazione di base sono:

- Simulazione temperatura di ogni stanza: in ogni stanza viene simulata una temperatura compresa tra i 17 e 23 gradi per simulare gli output dei sensori di temperatura nelle situazioni tipiche e verificare opportunamente il corretto funzionamento dell'attivazione/disattivazione dei radiatori e del sistema di riscaldamento;
- Simulazione qualità dell'aria di ogni stanza: in ogni stanza viene simulata la qualità dell'aria per mezzo della concentrazione di CO2, di Vocs, di Pm10 e di Rh;
- Simulazione della qualità dell'aria outdoor: periodicamente viene simulata la qualità dell'aria outdoor sulla base della concentrazione di Pm10.

Invece i requisiti per la simulazione di eventi particolari sono:

- Simulazione incendio: viene effettuata un'opportuna simulazione di incendio, caratterizzata dall'impostazioni di output dei sensori di temperatura elevati (>45°C) in una zona della casa o dall'attivazione del sensore di fumo;
- Simulazione di qualità non buona dell'aria interna;
- Simulazione dell'aria esterna non buona;
- Simulazione di sensore guasto.

2.4 Adaptation Concerns

Il sistema di controllo gestito per mezzo di OpenHab è stato realizzato allo scopo di implementare delle logiche di adattamento e quindi di gestire opportunamente gli Adaptation concerns richiesti.
Nonostante sarebbero disponibili molti adaptation concerns per questo caso di studio, sono stati gestiti solamente i seguenti:

- Offrire il massimo comfort del riscaldamento: in ogni periodo del giorno il sistema aggiusta la temperatura dei termostati in base alle impostazioni utente; garantendo il raggiungimento della temperatura ideale in ogni stanza.
- Garantire il rilevamento di un incendio: il sistema deve rilevare la presenza di un incendio, permettendo un controllo continuo dell'ambiente.
- Offrire la massima qualità dell'aria: in ogni stanza il sistema deve verificare la qualità dell'aria ed eventualmente gestire l'apertura e chiusura delle finestre.

Case	Adaptation Goal	Source of uncertainty	Type of adaptation
S1	Massimizzare il comfort di riscaldamento	Diverse impostazioni utente durante la giornata	Adattare la temperatura dei termostati in base al periodo del giorno; accendere e spegnere i radiatori in base alla temperatura di ogni stanza
S2	Garantire il rilevamento di un incendio	Numero variabile di sensori di fumo per stanza.	Controllo della presenza di un incendio per mezzo di altri devices, per esempio per mezzo dei sensori di temperatura
		Misure affette da rumore / sensore di fumo non funzionante	
S3	Massimizzare la qualità dell'aria indoor	Qualità dell'aria outdoor	Apertura/chiusura delle finestre sulla base della qualità dell'aria outdoor ottenuta per mezzo di servizi esterni, per esempio aqicn.org

2.4.1 Massimizzare il comfort di riscaldamento

Allo scopo di gestire un sistema di riscaldamento che offra il massimo comfort (case S1), i termostati aggiornano costantemente la temperatura target di ogni stanza sulla base delle preferenze dell'utente, permettendo di mantenere la temperatura ideale quando gli utenti sono in casa e risparmiando energia quando nessuno è in casa. A tale fine, i termostati accendono e spengono i diversi radiatori in base alle temperature raggiunte in ogni stanza.

2.4.2 Garantire il rilevamento di un incendio

Tale concern (case S2) ha l'obiettivo di garantire un sistema di controllo di incendio più robusto. Infatti, tale sistema di controllo attiva l'allarme antincendio non solo quando i sensori di fumo hanno rilevato del fumo/fiamme, ma anche quando un sensore di temperatura riporta una temperatura maggiore di 45°C.

2.4.3 Massimizzare la qualità dell'aria indoor

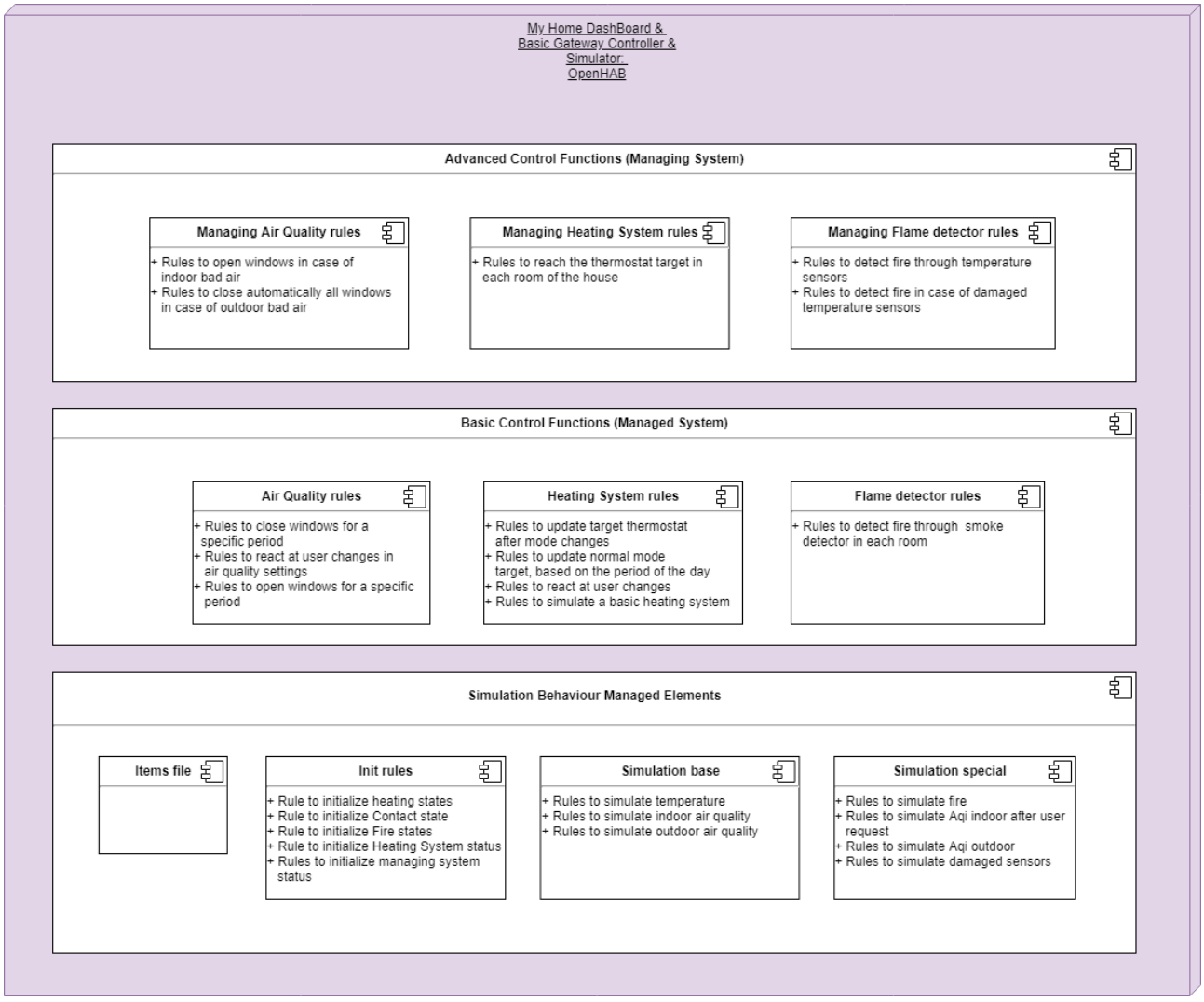
Allo scopo di massimizzare la qualità dell'aria indoor in ogni stanza della casa (case S3), il sistema di controllo non si basa solo sulle informazioni di qualità dell'aria ottenute per mezzo dei diversi sensori di qualità dell'aria presenti nelle varie stanze, ma fa riferimento anche alla qualità dell'aria esterna, informazione offerta da appositi servizi, come per esempio aqicn.org. In particolar modo, il sistema aprirà le finestre in caso di aria interna non buona solamente se l'aria esterna è buona, in caso contrario verranno mantenute chiuse.

2.5 Component Diagram

La componente OpenHab ha quindi una struttura a layer e la sua realizzazione è stata effettuata proprio in base ad essi; tale componente è quindi formata da un layer rappresentante il simulatore dell'ambiente di OpenHab, dal layer del sistema di controllo di base (Managed system) e dal layer del sistema di controllo avanzato (Managing system).

Inoltre, l'implementazione dei due sistemi di controllo è stata effettuata per concern, suddividendo quindi i controlli che regolano i tre principali concerns su cui si basa tale progetto: Comfortabel Heating, Air Quality system, Fire Detection system.

Nel diagramma a componente seguente è possibile vedere le principali regole che sono presenti in ogni componente e che consentono la gestione dei diversi requisiti funzionali precedentemente descritti.



2.6 Use Cases

Dato che l'implementazione della componente in OpenHab è stata effettuata sulla base dei tre layer: simulatore, Managed system e Managing system; anche i casi d'uso sono stati realizzati mantenendo tale separazione.

2.6.1 Simulator Use Cases

Per la corretta gestione e verifica del sistema di controllo implementato è necessario simulare il comportamento di tutti i sensori e dispositivi presenti nella casa virtuale, realizzata tramite OpenHab.
La simulazione è stata gestita distinguendo tra la simulazione dei comportamenti standard dei differenti sensori e la simulazione di eventi particolari che si verificano solo in specifiche situazioni, per esempio l'incendio di una zona della casa.
In particolar modo mentre la simulazione di base della casa è effettuata tramite rules automatiche gestite dal sistema di OpenHab, la simulazione specifica può essere invocata dall'utente il quale può quindi verificare le reazioni dei diversi devices della casa in base all'evento critico invocato.

Simulazioni di base

UC1: Simulazione delle temperature in ogni zona della casa

Descrizione: il sistema deve simulare periodicamente gli output dei sensori di temperatura di ogni zona della casa.

Attori: sistema OpenHab

Passi base:

Il sistema periodicamente (ogni minuto/ogni trenta secondi) modifica lo stato dei sensori di temperatura di ogni stanza, simulando la temperatura misurata da essi. La nuova temperatura impostata è scelta in modo casuale all'interno di un intervallo tra 17 e 23 gradi.

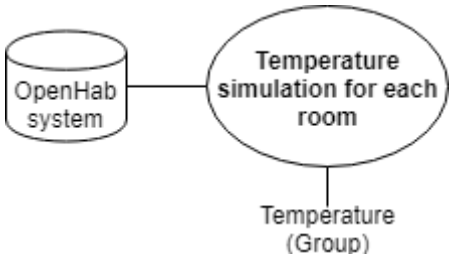


Figura 3: UML use case Temperature simulation

UC2: Simulazione della qualità dell'aria in ogni stanza della casa

Descrizione: il sistema deve simulare la qualità dell'aria di ogni stanza, cioè l'output dei rilevatori della qualità dell'aria indoor.
Avendo effettuato una ricerca dei possibili dispositivi acquistabili siamo giunti alla conclusione che attualmente sono presenti sul mercato sia dispositivi che consentono di fare una stima base della qualità

dell'aria tenendo in considerazione solo il livello di CO_2 presente e la relativa umidità, sia sensori più precisi e dettagliati che rilevano anche il livello di CO_2, Rh (umidità), VOC_s (Volatile Organic Compounds) , PM2.5, PM10, Radon, NO_2, O_3 etc.

Per tale motivo abbiamo deciso di simulare la qualità dell'aria e gestirne il suo controllo ipotizzando di avere a disposizione un dispositivo di precisione media che permetta di rilevare il livello di CO_2, VOC_s, PM10 e Rh dell'aria.

In particolar modo ci siamo basati sui dati pubblicati da diverse organizzazione EPA(United States Environmental Protection Agency), WHO(World health organization) confrontandoli con quanto pubblicato negli articoli inerenti all'indice di qualità dell'aria indoor presenti sul www.researchgate.net.

Abbiamo quindi fatto riferimento ai seguenti valori, limitando l'utilizzo delle sole colonne CO_2, VOC_s, PM10 e Rh:

CO_2 (ppm)	**CO (ppm)**	**NO_2(ppm)**	**O_3(ppm)**	**IAQI**	**IAQI Status**	**TCI**	**TCI Status**
340-600	0.0 – 1.7	0.000-0.021	0.000 – 0.025	100-76	Good	100-76	Most Comfort
601 – 1000	1.8-8.7	0.022-0.08	0.026 – 0.05	75-51	Moderate	75-51	Comfort
1001-1500	8.8-10.0	0.09-0.17	0.051-0.075	50-26	Unhealthy	50-26	Not Comfort
1501-5000	10.1-50	0.18-5	0.076-0.1	25.0	Hazardous	25-0	Least Comfort

PM10 (mg/m^3)	**VOC_s(ppm)**	**O_2(%)**	**Temp(°C)**	**Rh(%)**	**IAQI**	**IAQI Status**	**TCI**	**TCI Status**
0.000-0.020	0.000-0.087	23.5-20.9	20.0-26.0	40.0-70.0	100-76	Good	100-76	Most Comfort
0.021-0.150	0.088-0.261	20.8-19.5	26.1-29.0	70.1-80	75-51	Moderate	75-51	Comfort
0.151 – 0.180	0.262-0.43	19.4-12.0	29.1-39.0	80.1-90.0	50-26	Unhealthy	50-26	Not Comfort
0.181-0.600	0.44-3.00	11.9-10.0	39.1-45.0	90.1-100.0	25-0	Hazardous	25-0	Hazardous

Fonti:

- http://apps.who.int/iris/bitstream/handle/10665/69477/WHO_SDE_PHE_OEH_06.02_eng.pdf;jsessionid=A9DB3AD4EC5FEB3A49CE145500C8EA6E?sequence=1
- https://www3.epa.gov/ttn/naaqs/standards/pm/s_pm_history.html
- https://www.researchgate.net/publication/315006128_Development_of_indoor_environmental_index_Air_quality_index_and_thermal_comfort_index

Attori: sistema OpenHab

Passi base: il sistema periodicamente (ogni minuto/ogni trenta secondi) modifica i diversi parametri di qualità dell'aria delle diverse stanze della casa (rimanendo nel range dell'ultima simulazione effettuata in quella stanza, inizialmente tutte sono a Good).

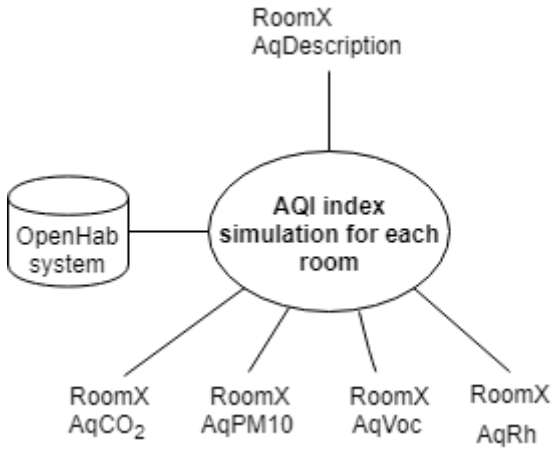


Figura 4: UML use case AQI simulation

UC3: Simulazione della qualità dell'aria outdoor

Descrizione: il sistema di controllo di qualità dell'aria prende decisioni sull'apertura o chiusura delle finestre anche in base alla qualità dell'aria outdoor per la quale ci siamo basati sulle informazioni trasmesse da *aqicn.org*.

In particolar modo il sistema OpenHab periodicamente modifica la qualità dell'aria outdoor per simulare un corretto funzionamento del sistema applicato nella realtà.

Per la simulazione ci siamo basati sull'indice di qualità dell'aria di aqicn.org:

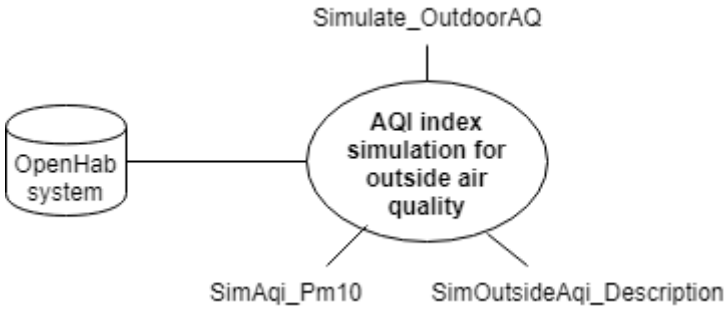
AQI	Air Pollution Level	Health implications
0 - 50	Good	Air quality is considered satisfactory, and air pollution poses little or no risk
51 -100	Moderate	Air quality is acceptable; however, for some pollutants there may be a moderate health concern for a very small number of people who are unusually sensitive to air pollution.
101-150	Unhealthy for Sensitive Groups	Members of sensitive groups may experience health effects. The general public is not likely to be affected.
151-200	Unhealthy	Everyone may begin to experience health effects; members of sensitive groups may experience more serious health effects
201-300	Very Unhealthy	Health warnings of emergency conditions. The entire population is more likely to be affected.
300+	Hazardous	Health alert: everyone may experience more serious health effects

Per semplicità abbiamo simulato l'aria esterna simulando la variazione di concentrazione di Pm10 in base ai range e ai valori riportati nella seguente tabella (fonti www.epa.gov) :

AQI Outdoor	Air Level	Range Used(mg/m^3)
0-50	Good	0-0.054
51-100	Moderate	0.055-0.154
101-300	Unhealthy	0.155-0.424
300+	Hazardous	0.425-0.640

Attori: sistema OpenHab

Passi base: il sistema periodicamente (ogni minuto/ogni trenta secondi) modifica il parametro di Pm10 della qualità dell'aria outdoor.



Use cases Simulatore di base:

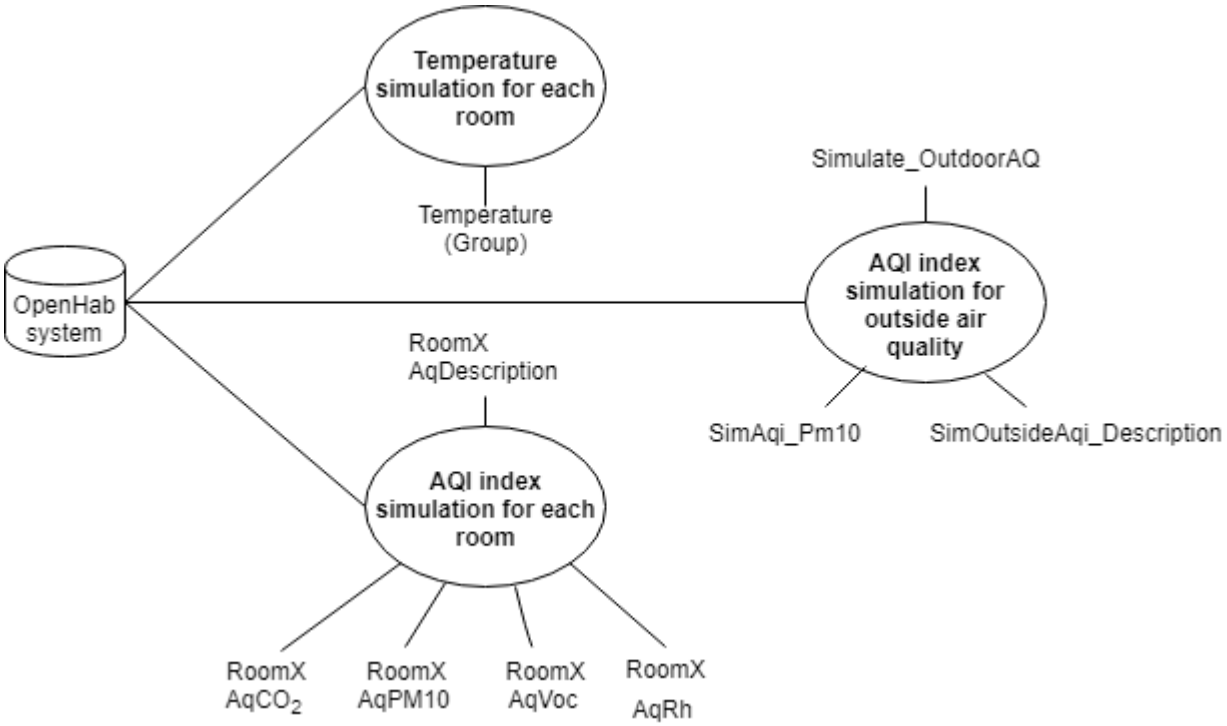


Figura 5: UML riassuntivo degli Use Cases del simulatore di base

Simulazioni speciali

UC4: Simulazione di incendio

Descrizione: alla richiesta dell'utente il sistema deve simulare un incendio in una specifica zona della casa

Passi base:

1. L'utente seleziona la simulazione di incendio in una stanza della casa
2. Il sistema può effettuare due diverse azioni (la scelta tra quale effettuare avviene in modo casuale):
    a. Il sistema attiva il sensore antifumo della stanza selezionata (se presente)
    b. Il sistema imposta una temperatura superiore a 45 °C nella stanza selezionata

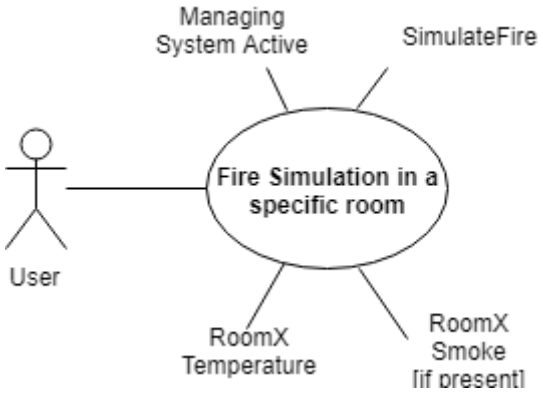


Figura 6: UML use case fire simulation

UC4: Simulazione di qualità dell'aria indoor

Descrizione: alla richiesta dell'utente il sistema deve simulare uno specifico livello di qualità dell'aria in una specifica zona della casa

Passi base:

1. L'utente seleziona la simulazione di uno specifico livello di qualità dell'aria indoor (scelta di uno dei range specificati nell' use case *Simulazione della qualità dell'aria in ogni stanza della casa*)
2. Il sistema simula la qualità dell'aria interna nella stanza specificata sulla base del range imposto dall'utente, in modo randomico vengono scelti i valori dei diversi parametri su cui si basa il controllo di qualità dell'aria indoor

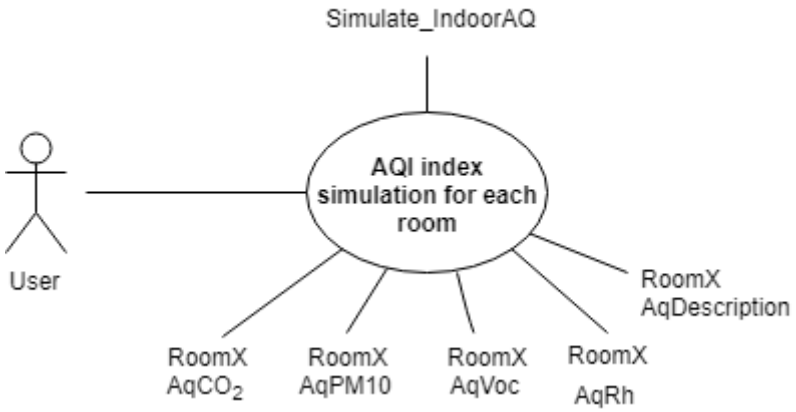


Figura 7: UML use case AQI indoor simulation

UC5: Simulazione di qualità dell'aria outdoor

Descrizione:

Allo scopo di verificare il comportamento in tutte le diverse situazioni di qualità dell'aria esterna che si possono presentare abbiamo dato la possibilità all'utente di simulare la qualità dell'aria esterna attraverso i range utilizzato anche per la simulazione di base dell'aria esterna.

Passi base:

1. L'utente seleziona la simulazione di uno specifico livello di qualità (scelta di uno dei precedenti range) dell'aria outdoor
2. Il sistema simula la qualità dell'aria esterna sulla base del range imposto dall'utente (in modo randomico)

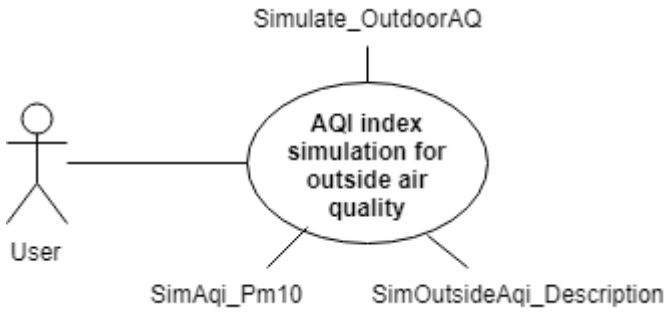


Figura 8: UML use case Outdoor AQI Simulation

UC6: Simulazione di sensore di temperatura guasto

Descrizione: il sistema di controllo avanzato deve gestire anche la possibile situazione in cui un sensore di temperatura sia guasto. L'utente può quindi simulare tale situazione scegliendo quale sensore rendere guasto.

Passi base:

1. L'utente seleziona la simulazione di sensore guasto su uno specifico sensore di temperatura
2. Il sistema simula il sensore guasto, pone ad UNDEF l'output di tale sensore

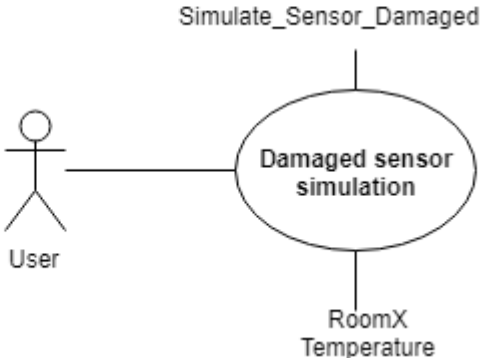


Figura 9: UML use case simulation sensor damaged

Use cases Simulatore Special

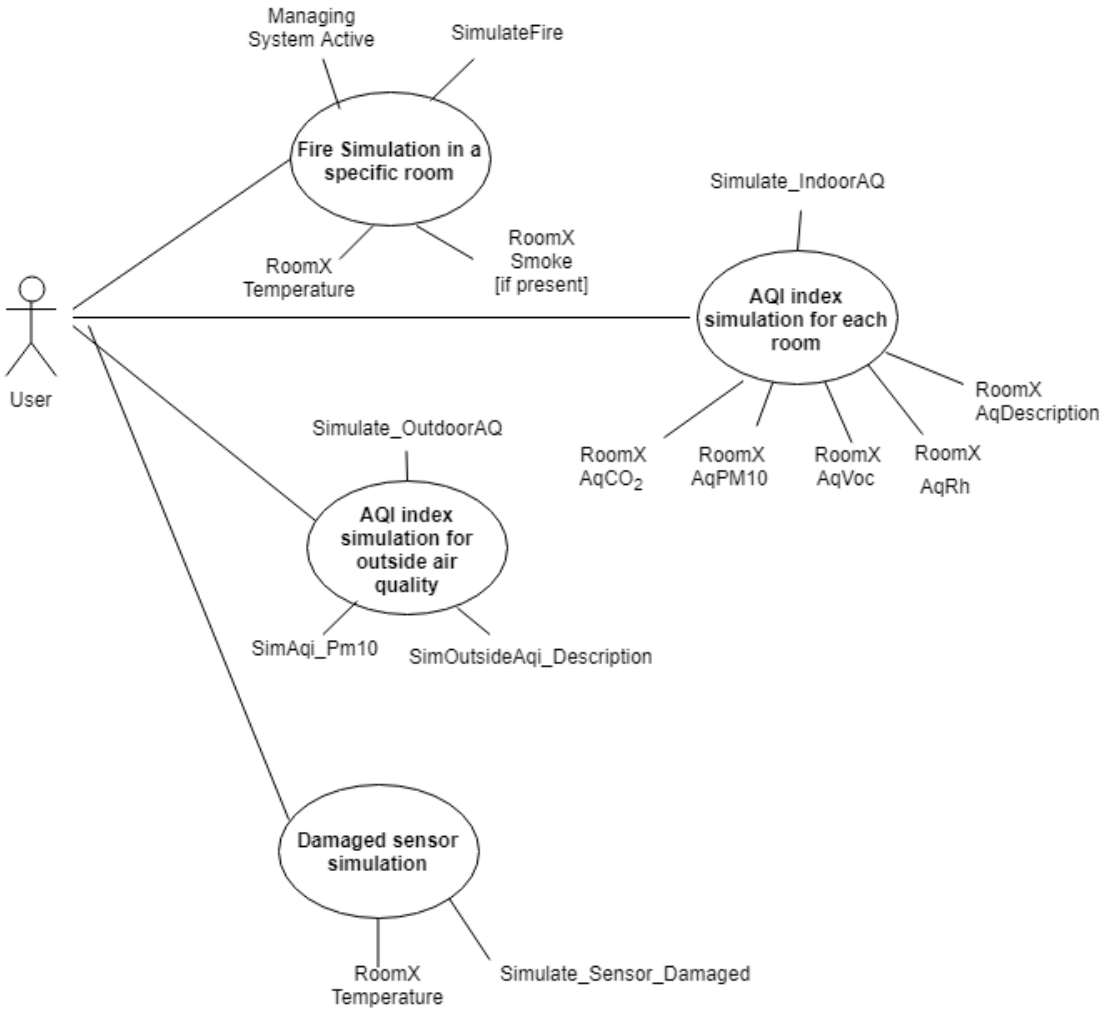


Figura 10: UML riassuntivo degli Use Cases del simulatore special

2.6.2 Managed system Use Cases

Avendo implementato il Managed system suddividendo i componenti per concern anche i relativi casi d'uso del Managed system sono suddivisi in base ad esso.

Heating System

Il sistema di riscaldamento è gestito per mezzo di un termostato per piano.

L'utente può modificare la modalità di riscaldamento che viene modificata per entrambi i termostati a cui possono essere associate soglie diverse di temperatura in base alla scelta dell'utente. Inoltre, è stato implementato un sistema di riscaldamento di base (simulando il comportamento di un generico termostato), esso accende e spegne i radiatori di un intero piano in base alla temperatura media del piano stesso.

UC1: Modifica della modalità di riscaldamento

Descrizione: l'utente può modificare la modalità di riscaldamento per l'intera casa, può scegliere tra quattro modalità: *normal mode, holiday mode, manual mode e stop mode*.

La modalità *normal mode* è quella utilizzata quotidianamente; in tale modalità la giornata è suddivisa in quattro fasce orarie (8.30-13.30,13.30-17.30,17.30-23.30,23.30-8.30), a cui sono associate diverse impostazioni di temperatura; *holiday mode* è la modalità pensata per giornate in cui nessuno si trova in casa, per esempio nel week-end, ad essa è associata una temperatura minima sotto cui non può scendere la temperatura di ogni piano della casa*; manual mode* è invece la modalità usata per far continuare il riscaldamento in modo continuo senza orari prestabiliti, in tal modalità è assegnata una temperatura massima che può essere raggiunta e infine *stop mode* si basa sul blocco completo del riscaldamento.

Attori: utente.

Passi base:

1. L'utente selezione il menu di impostazioni
2. L'utente cambia la modalità di riscaldamento
3. Il sistema cambia il set point dei termostati in base alle impostazioni della modalità selezionata

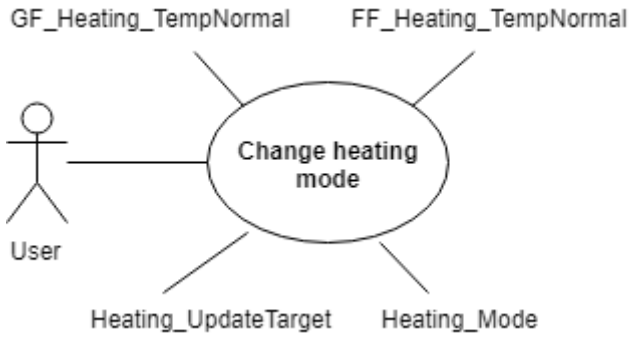


Figura 11: UML use case Change Heating Mode

UC2: Cambio delle impostazioni utente (per ogni piano della casa)

Descrizione: l'utente può modificare le impostazioni di riscaldamento in ogni piano della casa, le impostazioni assegnabili variano in base alla modalità di riscaldamento.

Attori: utente

Passi base:

1. L'utente seleziona il menu di impostazioni
2. L'utente modifica le impostazioni di una modalità
    a. Modalità Normal: l'utente modifica la temperatura target associata ad una specifica fascia di orario
    b. Modalità Manual: l'utente modifica la temperatura massima raggiungibile nelle stanze del piano
    c. Modalità Holiday: l'utente modifica la temperatura minima che deve essere mantenuta nelle stanze del piano
3. Il sistema effettua la modifica richiesta
4. Il sistema controlla la modalità di riscaldamento attuale
    a. Se la modalità attuale è quella che è stata modifica dall'utente: il sistema invia un comando di aggiornamento con il nuovo set point al termostato

Passi alternativi:

4. Se la modalità attuale e quella modificata è *normal mode*:
    b. Il sistema controlla l'attuale fascia oraria:
    Se la modifica è effettuata per l'attuale fascia oraria: il sistema invia un comando di aggiornamento con il nuovo set point al termostato

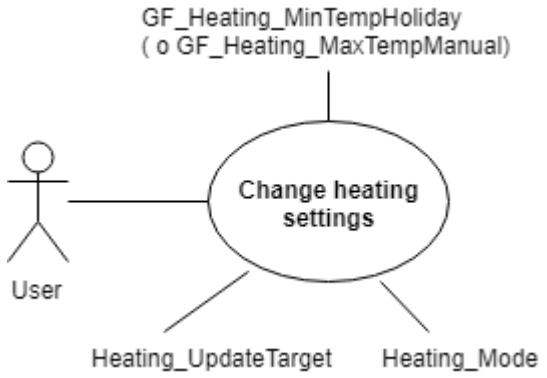


Figura 12: UML use case Change heating settings (when user change Holiday and Manual settings)

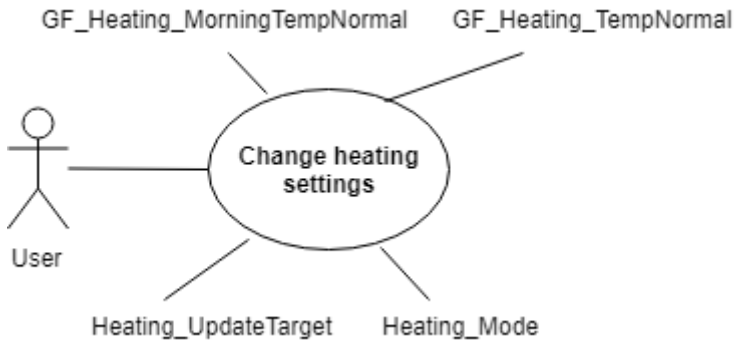


Figura 13: UML use cases Change heating settings (when user change Normal settings)

UC3: Aggiornamento automatico del target in normal mode

Descrizione: quando il sistema è in modalità *normal*, esso deve aggiornare il set point dei termostati ad ogni variazione della fascia oraria

Attori: sistema OpenHab

Precondizioni: l'orario attuale coincide con l'inizio di una fascia oraria

Passi base:

1. Il sistema modifica il set point associato alla modalità *normal*
2. Se la modalità attuale è la modalità *normal*: il sistema invia il nuovo target ai termostati

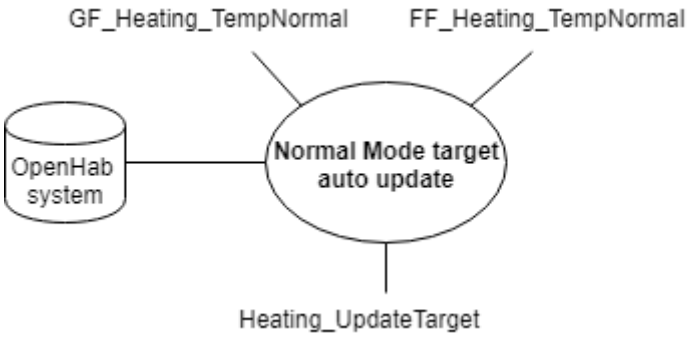


Figura 14: UML use case Normal Mode target auto update

UC4: Simulazione del comportamento di un termostato di base

Descrizione: il sistema periodicamente e alla variazione del target di temperatura di ogni piano accende o spegne tutti i radiatori del piano in base alla temperatura media del piano

Attori: sistema OpenHab

Passi base:

1. Il sistema ogni cinque secondi o ad ogni variazione del setpoint del termostato di un piano controlla la temperatura sul piano; si possono verificare due situazioni:
    a. Se la temperatura media del piano è inferiore del target del termostato del piano vengono attivati tutti i radiatori del piano

b. Se la temperatura media del piano è superiore del target del termostato del piano stesso tutti i radiatori sono spenti

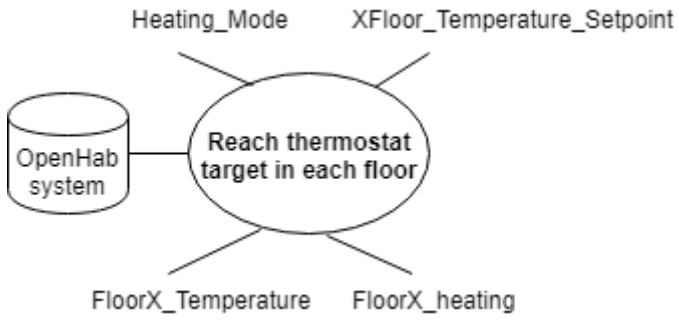


Figura 15: Use cases base thermostat behaviour simulation

Use cases Heating Managed system

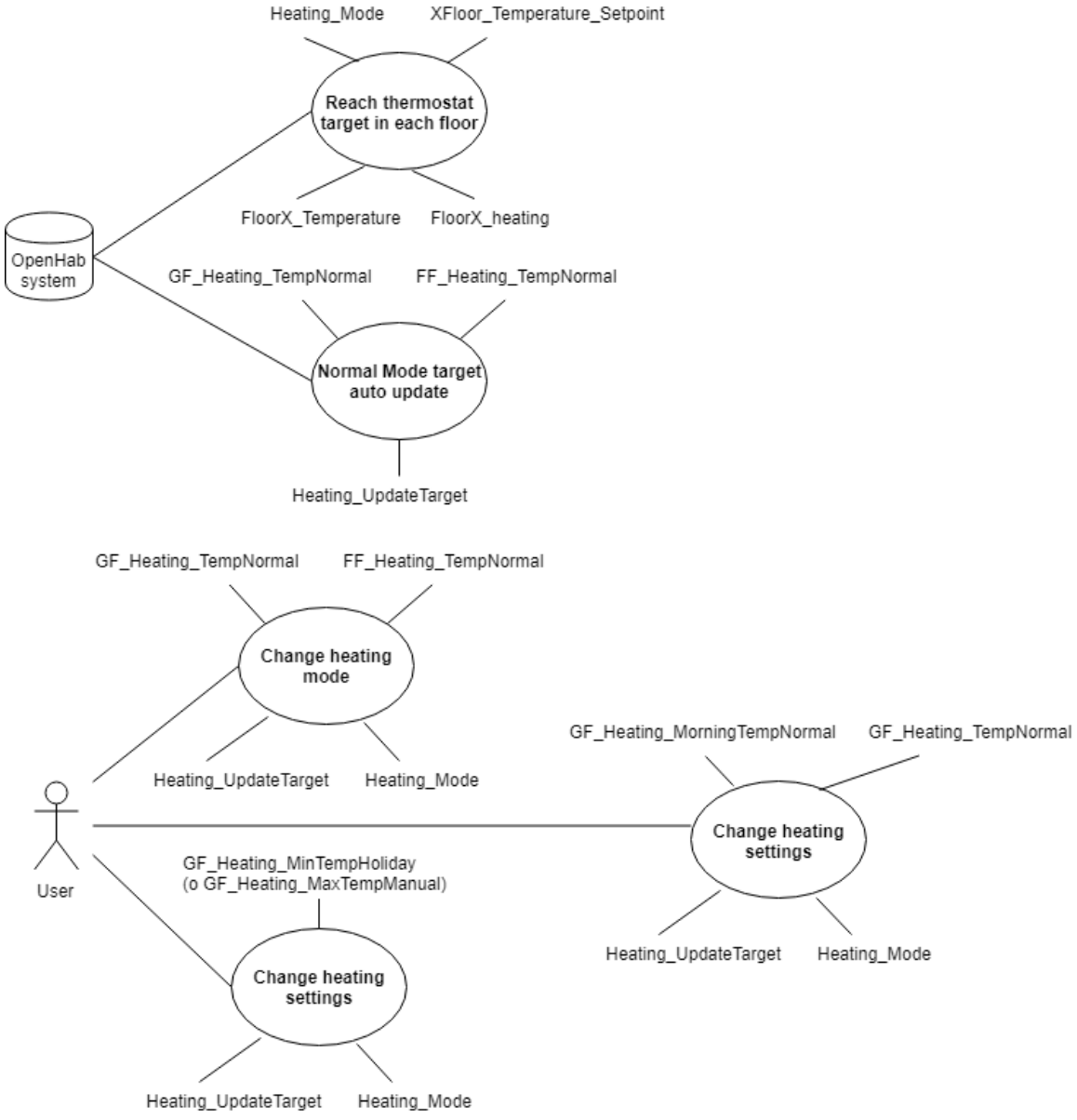


Figura 16: UML riassuntivo degli use cases del heating managed system

Air Quality System

Il sistema di controllo managed permette all'utente di programmare l'apertura o la chiusura di tutte le finestre in determinati orari della giornata, a prescindere dalla qualità dell'aria esterna.

UC1: Programmazione dell'orario di chiusura forzata delle finestre

Descrizione: l'utente può settare un orario in cui mantenere chiuse le finestre, a prescindere dalla qualità dell'aria, sia interna che esterna.

Attore: utente

Passi base: l'utente seleziona l'orario in cui attivare la chiusura forzata delle finestre e l'orario in cui disattivarla.

UC2: Programmazione dell'orario di apertura forzata delle finestre

Descrizione: l'utente può settare un orario in cui mantenere aperte le finestre, a prescindere dalla qualità dell'aria, sia interna che esterna.

Attore: utente

Passi base: l'utente seleziona l'orario in cui attivare l'apertura forzata delle finestre e l'orario in cui disattivarla.

UC3: Chiusura forzata delle finestre

Attore: sistema Openhab

Passi base: il sistema mantiene chiuse tutte le finestre dell'abitazione durante l'orario specificato dall'utente.

UC4: Apertura forzata delle finestre

Attore: sistema Openhab

Passi base: il sistema mantiene aperte tutte le finestre dell'abitazione durante l'orario specificato dall'utente.

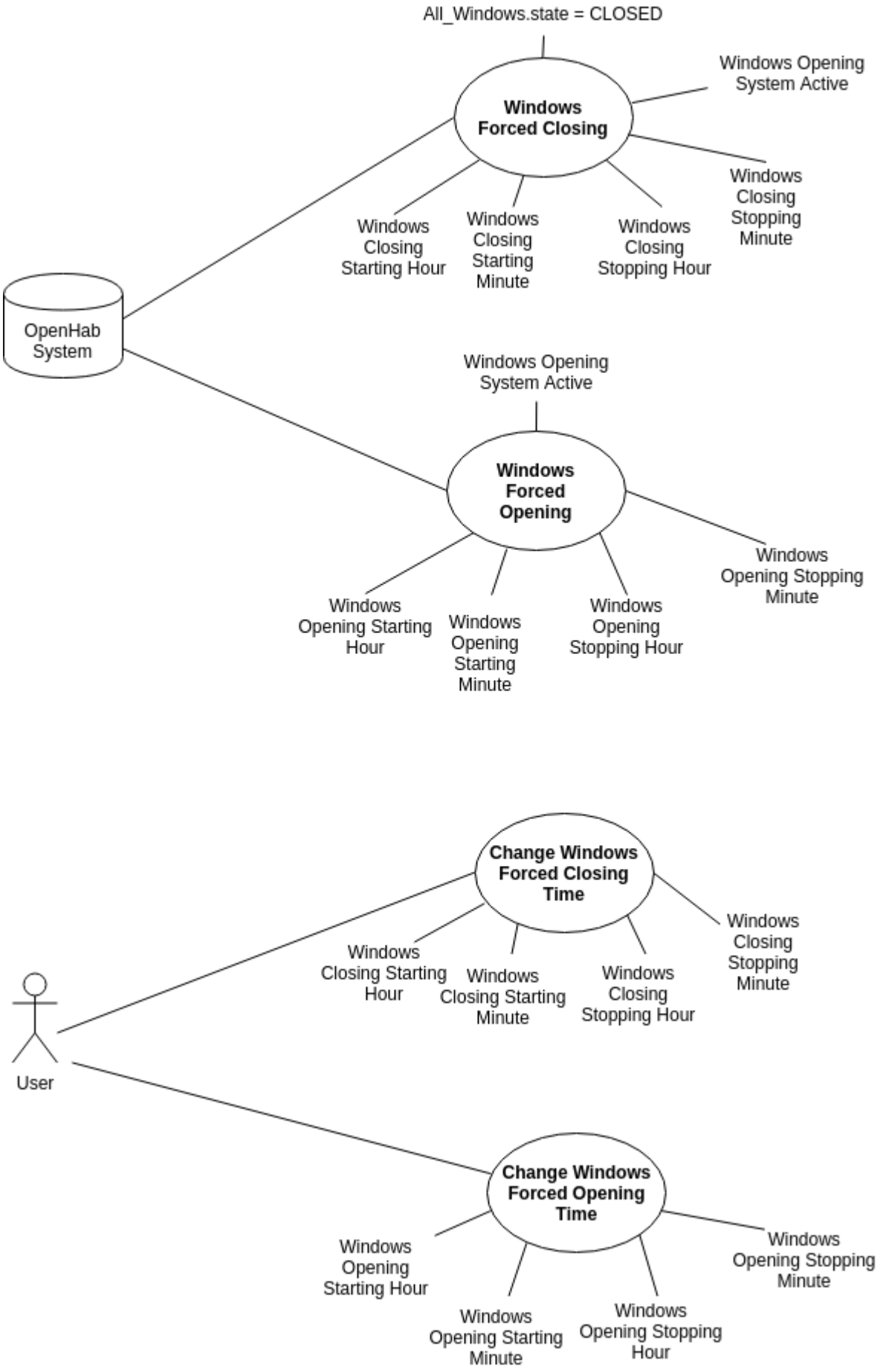


Figura 17: UML riassuntivo degli use cases dell' air quality managed system

Flame Detector System

Nel sistema di controllo di base del sistema antincendio viene verificata la presenza di un incendio solamente in base agli output dei sensori di fumo.

UC5: Controllo di incendio

Attori: sistema OpenHab

Passi base:

1. Quando il sensore di fumo di ogni stanza cambia stato da OFF a ON. Il sistema OpenHab attiva l'allarme di incendio di quella specifica stanza.

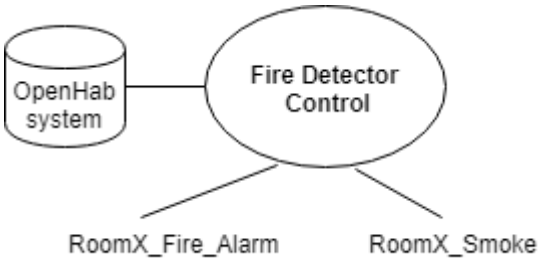


Figura 18: UML use cases Fire detector control

2.6.3 Managing system

L'implementazione del sistema di controllo avanzato è stata anch'essa gestita per concern: comfortable heating, air quality system, fire detection system.
Allo scopo di dare la possibilità all'utente di verificare i diversi comportamenti del sistema quando è presente un sistema Managing, e quando esso è assente, abbiamo aggiunto un Item *Managing_System_Active* che può essere attivato o disattivato dall'utente; esso attiva e disattiva quindi il controllore avanzato.

Managing Heating system

Allo scopo di garantire il massimo comfort e ridurre gli sprechi di energia, sulla base delle temperature impostate dall'utente nei diversi momenti della giornata e in base agli output dei sensori di temperatura vengono attivati e disattivati i radiatori.
Non è stata implementata alcuna regola per la gestione del sistema di riscaldamento nel caso in cui i sensori di temperatura siano guasti in quanto il sistema OpenHab gestisce in automatico tale situazione. Infatti, quando tali sensori hanno come output UNDEF il sistema OpenHab non modifica lo stato del riscaldamento e agisce quindi come se l'output dei sensori fosse ancora pari all'ultima temperatura misurata prima del guasto.

UC1: Raggiungimento del target del termostato in ogni stanza della casa

Descrizione: il sistema deve gestire l'attivazione o spegnimento dei radiatori presenti in ogni stanza della casa allo scopo di mantenere la temperatura dell'attuale set point assegnato ed evitare spreco di energia.

L’attivazione e lo spegnimento sono effettuati per mezzo di un’isteresi di 0.3° allo scopo di evitare accensioni e spegnimenti troppo frequenti dovuti a piccole variazioni di temperatura.

Attori: sistema OpenHab

Precondizioni: il Managing system deve essere attivo

Passi base:

Il sistema controlla periodicamente la temperatura della stanza, si possono verificare due situazioni:

a. Se la temperatura attuale è minore del set point associato al piano della stanza e non ci si trova nella modalità stop mode: il sistema accende il radiatore della stanza
b. Se la temperatura attuale è maggiore del set point associato al piano della stanza: il sistema spegne il radiatore della stanza

Passi alternativi:

Se la modalità di riscaldamento è cambiata in Stop Mode: il sistema spegne tutti i radiatori della casa, indipendentemente dalla temperatura presente

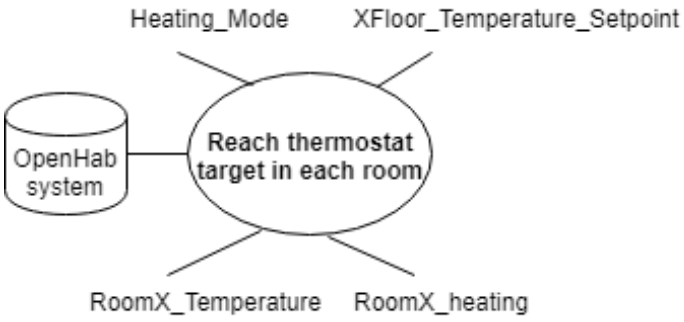


Figura 19: UML use cases Heating Managing system

Managing Air Quality system

Il sistema si occupa di aprire le finestre nel caso in cui l’aria esterna sia migliore di quella interna

UC2: Controllo delle singole stanze basato sulla qualità dell’aria

Descrizione: se la qualità dell’aria esterna è migliore di quella di una stanza, la finestra presente in quest’ultima viene aperta automaticamente dal sistema.

Attori: sistema OpenHab.

Precondizioni: il Managing system deve essere attivo.

Passi base:

1. il sistema controlla la concentrazione di PM10 presente nell’aria all’esterno e all’interno, in ogni singola stanza
2. il sistema confronta la concentrazione tra l’esterno e le singole stanze
3. il sistema apre la finestra di una stanza se la concentrazione di PM10 presente nell’aria all’interno di questa è superiore rispetto a quella contenuta nell’aria esterna.

UC3: Chiusura delle finestre in caso di cattiva qualità dell’aria esterna

Descrizione: se la qualità dell'aria esterna non è buona, vengono automaticamente chiuse tutte le finestre.

Attori:sistema OpenHab.

Precondizioni: il Managing system deve essere attivo.

Passi base: il sistema chiude tutte le finestre nel caso in cui ci sia alta concentrazione di PM10 contenuta nell'aria esterna.

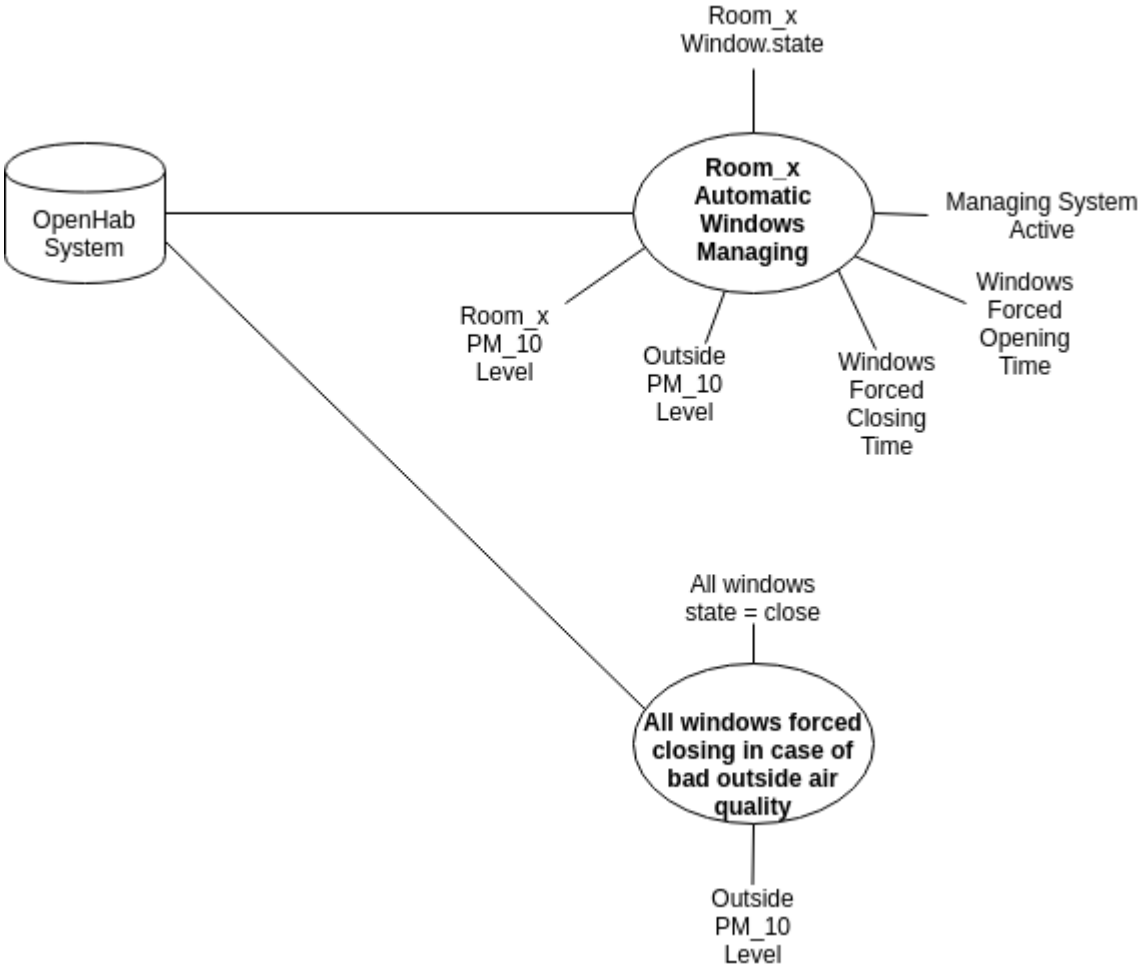


Figura 20: UML use cases Managing System Air Quality System

Managing Fire Detector system

Allo scopo di garantire che il rilevamento di incendio venga effettuato anche quando i sensori di fumo non funzionano o non sono presenti, il controllo di incendio considera anche l'output dei sensori di temperatura.

UC: Controllo di incendio basato sulle temperature

Descrizione: se un sensore di temperatura riporta una misura superiore ai 45°C viene attivata l'allarme di incendio.

Attori: sistema OpenHab

Precondizioni: il Managing system deve essere attivo

Passi base:

1. Il sistema OpenHab controlla ad ogni variazione di temperatura se essa è superiore a 45°C.
    a. Se questo è verificato, viene attivata l'allarme di incendio

UC: Controllo di incendio in caso di sensore guasto

Descrizione: in tale sistema di controllo vengono gestite le situazioni in cui un sensore di temperatura non funzioni, in tal caso il controllo di incendio viene eseguito per mezzo dell'ultima temperatura memorizzata.

Attori: sistema OpenHab

Precondizioni: il Managing system deve essere attivo

Passi base:

1. Il sistema OpenHab, ad ogni variazione di temperatura, memorizza lo stato precedente del sensore interessato alla variazione.
2. Il sistema controlla l'attuale stato del sensore:
    a. Se la temperatura in output è pari ad UNDEF, il sistema controlla la presenza di un incendio sulla base dello stato precedente di tale sensore di temperatura (ultima temperatura disponibile diversa da UNDEF).

Dato che questi due casi d'uso sono molto correlati tra loro, essi sono stati implementati insieme in un'unica OpenHab rule per ogni sensore.

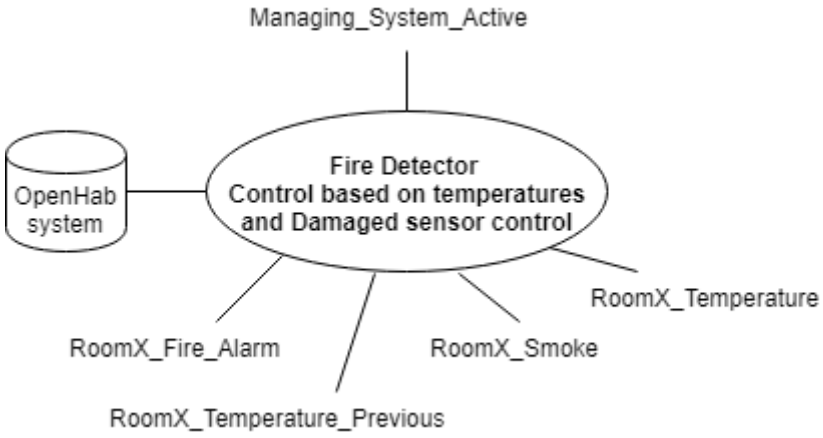
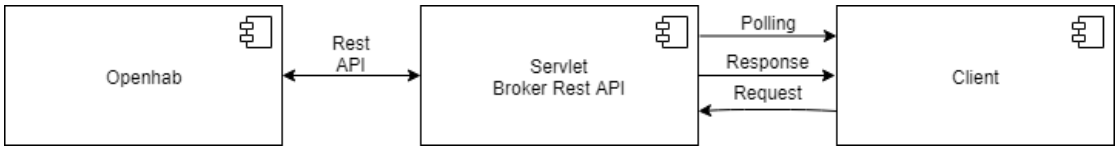


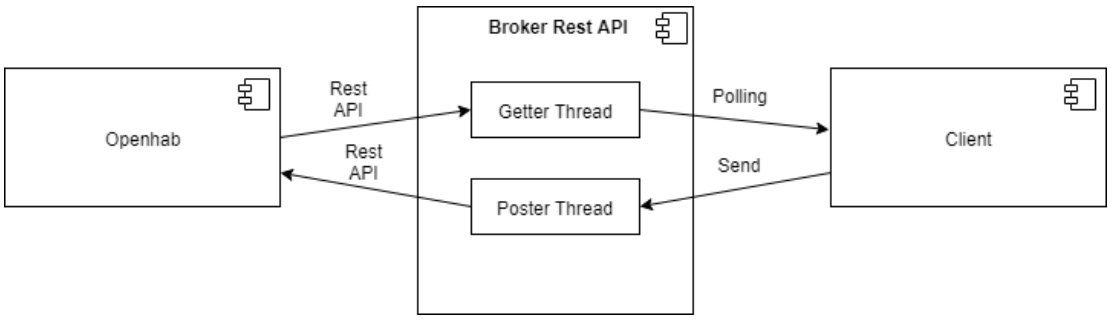
Figura 21: UML use cases Managing System Fire Detection

Iterazione 2

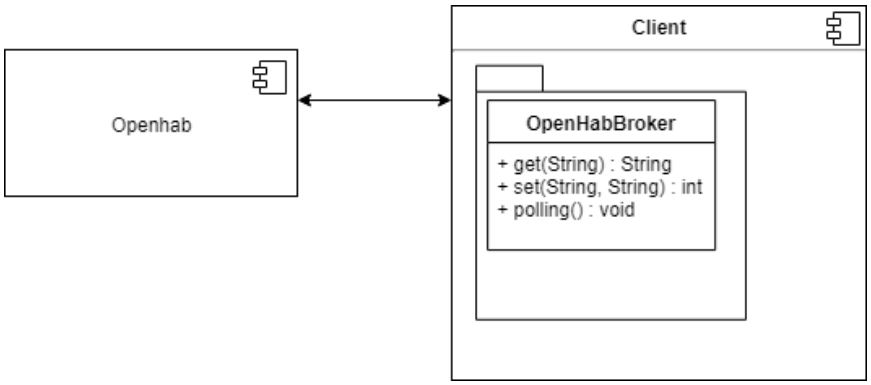
L'iterazione 2 dello sviluppo del progetto Virtual Smart Home è incentrata sull'implementazione della componente OpenHab Broker, essa è stata realizzata in Java e ha lo scopo di consentire la comunicazione tra il sistema OpenHab, in cui è stato realizzato il sistema di controllo (di base ed avanzato), e un sistema di controllo avanzato implementato per mezzo di MAPE loop.
La realizzazione di tale componente Broker può essere effettuata con diverse modalità, in particolar modo abbiamo identificato tre possibili soluzioni adatte a tale scopo.
Alternativa 1: si basa sulla realizzazione di una servlet per esempio per mezzo di Apache Tomcat che permetta di implementare un servizio di polling (richiedere per mezzo di RestAPI lo stato degli Items e trasmetterlo ai clients) e di gestire le richieste da parte di uno o più client (get e post dello stato di uno o più items).



Alternativa 2: si basa sull'utilizzo di socket, che consenta l'utilizzo delle Rest API, nella realizzazione di un thread che svolge il servizio di polling e di un thread per la gestione delle richieste di post del client.



Alternativa 3: si basa sulla realizzazione di un package in cui vengono implementati i metodi get, set e il servizio di polling. Tale package potrà quindi essere utilizzato da una qualsiasi altra applicazione per avere informazioni sugli stati dei diversi items presenti in OpenHab.



Nel seguente progetto è stata implementata la terza alternativa, come infatti si può vedere nel Deployment Diagram dell'iterazione precedente la componente OpenHabBroker può essere utilizzata dal controllore avanzata per semplice invocazione di metodo.
In tale iterazione non abbiamo modificato il Deployment Diagram che è rimasto esattamente uguale a quello dell'iterazione 1.

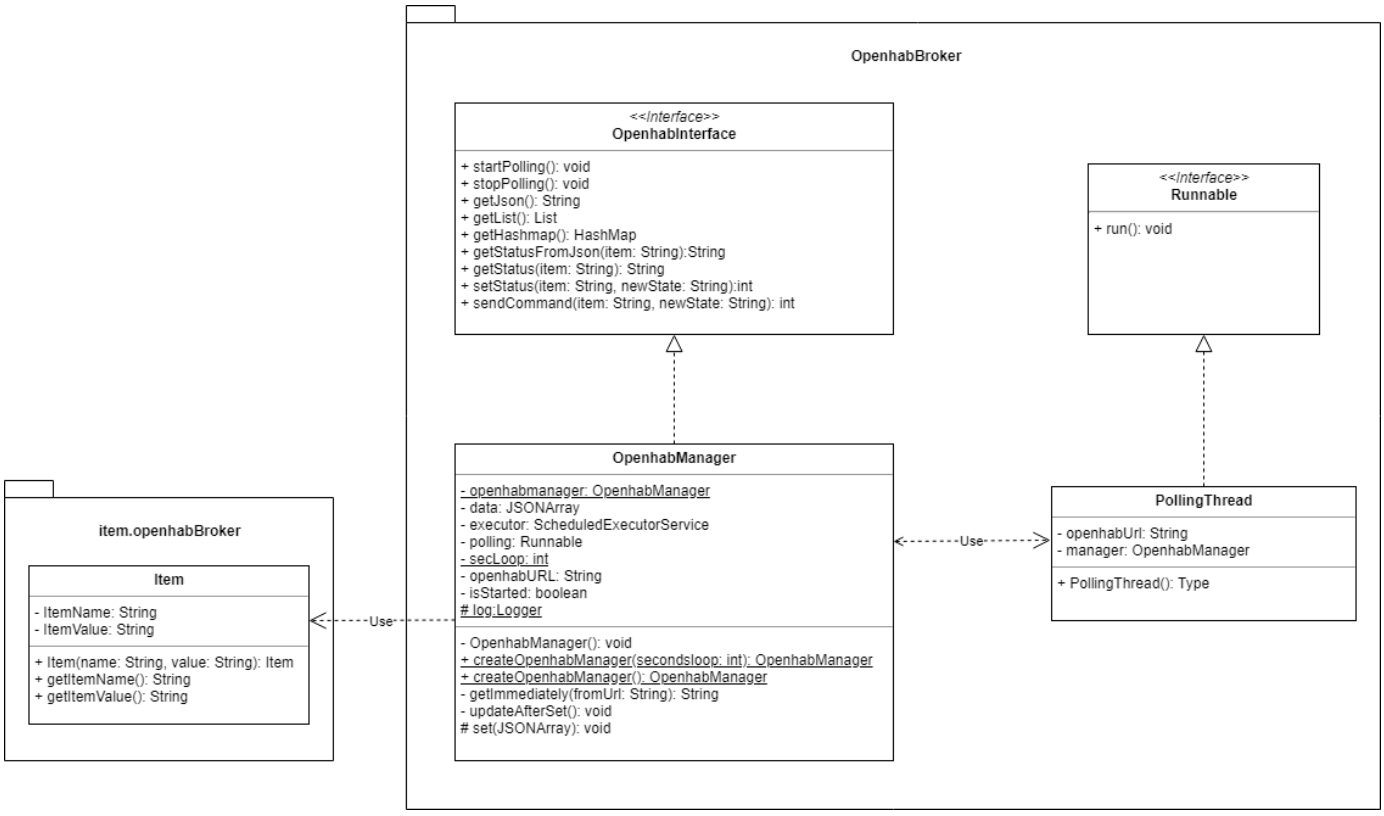
3.1 Requisiti Funzionali

I requisiti funzionali della componente OpenHabBroker sono:

- Servizio di Polling: la componente Java deve periodicamente richiedere tramite RestAPI lo stato di ogni item del sistema OpenHab e memorizzarlo in un'apposita struttura dati. Tale servizio può essere attivato e disattivato dalle applicazioni che utilizzano tale package;
- Metodo Get: la componente deve comprendere metodi che permettono di accedere alle informazioni raccolte per mezzo del servizio di polling, devono essere presenti dei metodi che consentano di ottenere gli stati in diversi "formati";
- Metodo GetStatus: OpenHabBroker deve prevedere anche un metodo che effettui direttamente il get da OpenHab di uno specifico Item, invece che accedere alle informazioni presenti nella struttura dati;
- Metodi setStatus/sendCommand: la componente consente la trasmissione di comandi e l'aggiornamento degli stati degli Items per mezzo di RestAPI

3.2 Class Diagram

La componente realizzata in Java è costituita dalle seguenti classi:



3.2.1 Descrizione metodi

La classe attraverso la quale è stata implementata la componente OpenhabBroker è presente nell'omonimo package ed è la classe OpenhabManager.

Tale classe implementa l'interfaccia *OpenhabInterface* che fornisce i metodi richiesti dai requisiti funzionali. I metodi implementati dell'interfaccia sono:

- startPolling: questo metodo inizia l'esecuzione del servizio di polling. Esso crea un oggetto PollingThread, se non è già presente un altro in esecuzione;
- stopPolling: questo metodo interrompe l'esecuzione del servizio di polling;
- getJson: ritorna l'intero file json, comprendente tutti gli items e i relativi stati, aggiornato per mezzo del servizio di polling (se il servizio di polling non è attivo ritorna un log WARN);
- getList: ritorna una lista di oggetti *Item* a partire dal contenuto del file Json determinato tramite il servizio di polling (se il servizio di polling non è attivo ritorna un log WARN);
- getHashmap: ritorna una HashMap, avente come chiave il nome dell'item e come valore lo stato di quest'ultimo; tale HashMap è costruita a partire dal file Json aggiornato tramite il polling (se il servizio di polling non è attivo ritorna un log WARN);
- getStatusFromJson: tale metodo ritorna lo stato di uno specifico item, argomento del metodo, salvato nel JsonArray data aggiornato tramite il servizio di polling (se il servizio di polling non è attivo ritorna un log WARN);
- getStatus: tale metodo consente di ottenere lo stato di un item effettuando una chiamata rest direttamente ad OpenHab;
- setStatus: con tale metodo è possibile modificare lo stato di un item, esso è utilizzabile per quegli items a cui non è possibile impartire dei comandi (per esempio per gli items di tipo Contact). In tale metodo viene eseguita una PUT rest call ad Openhab, modifica in tal modo lo stato dell'item specificato in argomento e invoca un aggiornamento del JsonArray data tramite il metodo *updateAfterSet*;
- sendCommand: con tale metodo è possibile eseguire una POST rest call su Openhab, permette quindi di inviare un comando ad uno specifico item di Openhab. Il metodo termina aggiornando il JsonArray data invocando il metodo updateAfterSet. Tale metodo non può essere utilizzato per gli items i quali non prevedono la ricezione di comandi per esempio per gli items di tipo Contact;

Oltre ai metodi dell'interfaccia implementata la classe OpenHabManager è costituita anche dai seguenti:

- OpenhabManager: esso è il costruttore privato della classe OpenhabManager, infatti abbiamo implementato la classe con il pattern Singleton. Dato che l'utilizzo all'interno di un'applicazione esterna di tale classe fa sempre riferimento allo stesso sistema OpenHab, allo scopo di mantenere consistenza nei dati trasmessi e nel servizio offerto è possibile creare solo un'istanza di tale classe. In tale metodo viene inizializzata la struttura dati in cui vengono memorizzati tutti gli items e i relativi stati ottenuti da openHab (il JsonArray) e viene richiesto all'utente tramite console l'url del sistema OpenHab a cui è necessario accedere (viene inizializzata la variabile *openhabURL*);

- createOpenhabManager(secondsloop: int): tale metodo statico può essere invocato per creare un'istanza della classe OpenhabManager, essendo essa Singleton il metodo creerà un oggetto OpenhabManager solo se non è già presente, inizializzando la variabile openhabmanager e la variabile secLoop con l'argomento del metodo, quest'ultima rappresenta ogni quanti secondi deve essere eseguito il servizio di polling quanto attivato, in caso contrario ritorna solamente il riferimento a tale oggetto;
- createOpenhabManager: tale metodo svolge le stesse funzionalità del precedente, l'unica differenza riguarda il fatto che se invocato, quando non è stato ancora creato un oggetto OpenhabManager, crea un'istanza della classe OpenhabManager ed inizializza la variabile secLoop, che gestisce la periodicità del servizio di polling di default ad un secondo;
- getImmediately: tale metodo privato è utilizzato dai metodi *getStatus* e *updateAfterSet* per effettuare un GET rest call al sistema Openhab senza dover attendere l'aggiornamento del JsonArray data tramite il polling, esso prende in argomento l'url a cui effettuare la chiamata, in particolar modo quando invocato dal metodo *updateAfterSet* l'url sarà del tipo http://localhost:8080/rest/items; mentre quando invocato dal metodo *getStatus* sarà pari a http://localhost:8080/rest/items/itemname.
- updateAfterSet: tale metodo privato è invocato dai metodi setStatus e sendCommand allo scopo di aggiornare la struttura dati JsonArray data, in cui vengono memorizzati dal servizio di polling tutti gli stati degli items, in seguito all'imposizione di un comando o di un cambiamento di stato in OpenHab, in tal modo la struttura dati potrà essere utilizzata per ottenere informazioni in merito alle variazioni che il comando impartito ha provocato;
- set: tale metodo protected è utilizzato dalla classe PollingThread, che si occupa dell'esecuzione del servizio di polling, allo scopo di aggiornare il JsonArray data ad ogni ciclo.

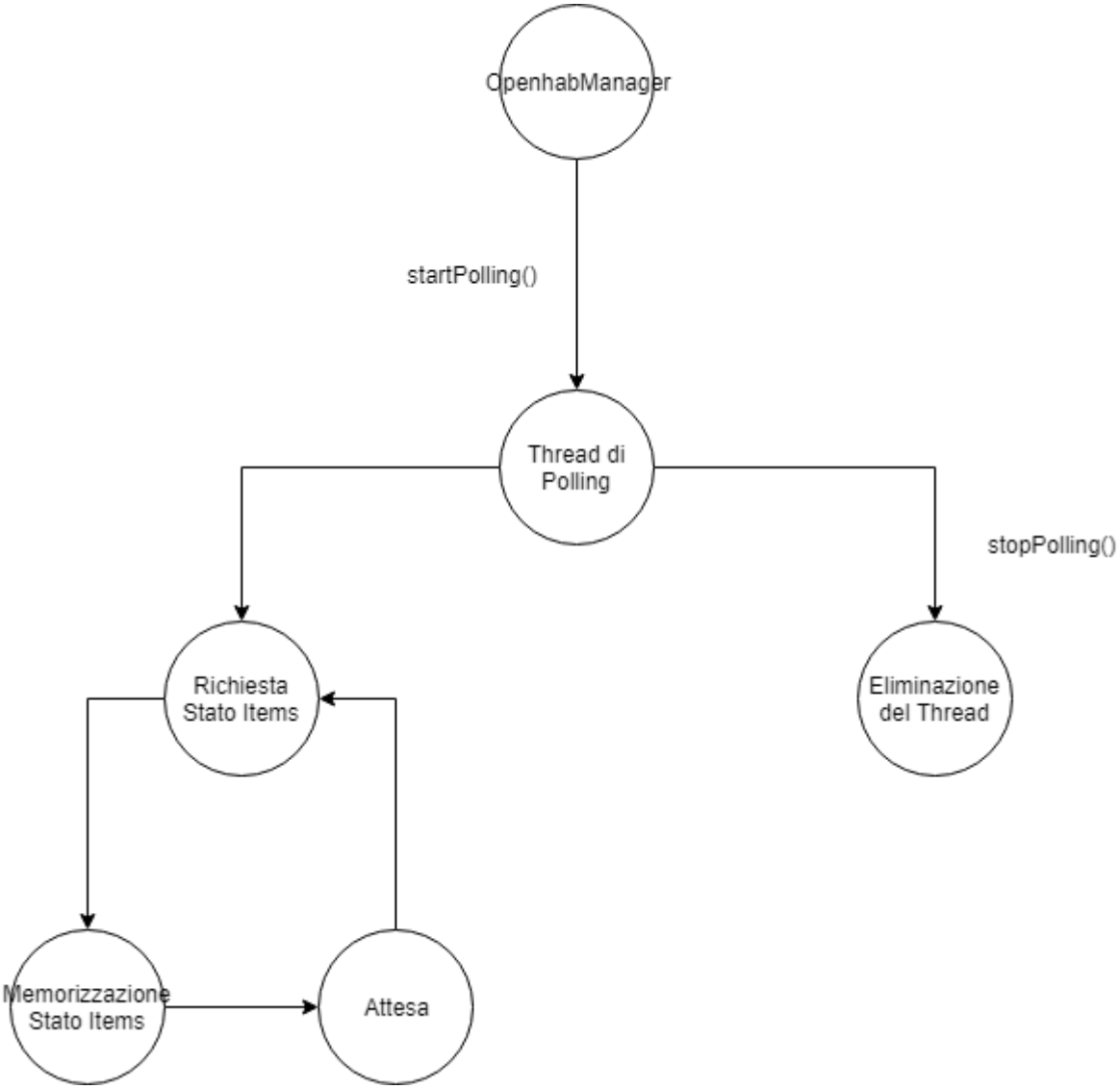
Nel package OpenhabBroker oltre alla classe appena descritta è presente la classe PollingThread, essa è utilizzata dalla classe precedente per l'esecuzione del servizio di polling.
La classe PollingThread implementa l'interfaccia Runnable, essa realizza un Thread il cui metodo run si occupa di eseguire una GET rest call su tutti gli items di OpenHab e di memorizzare il file Json, ottenuto in output, nella struttura dati dell'oggetto OpenhabManager.
Tale classe è strettamente collegata con la classe OpenhabManager, infatti quest'ultima crea un oggetto PollingThread (variabile polling) che viene eseguito secondo uno specifico scheduler (cioè in base alla frequenza richiesta dall'utente per mezzo dei secondsloops indicati) attraverso la variabile *executor* di tipo SchedulerExecutorService. La classe PollingThead rappresenta quindi il thread che implementa l'azione del polling ed è "comandato" tramite lo SchedulerExecutorService della classe OpenhabManager; in tal modo il JsonArray data di tale classe viene aggiornato periodicamente.

La classe OpenhabManager comprende, come detto in precedenza, un metodo getList, il quale fornisce una lista di oggetti Item. Tali oggetti sono istanze della classe Item, definita nel package item.openhabBroker.

Ogni istanza Item è caratterizzata da un nome (itemName) e da un valore/stato (itemValue). Inoltre comprende due metodi getItemValue(), che ritorna il valore di uno specifico oggetto item, e setItemValuer() con cui viene modificato il valore di uno specifico oggetto item.

3.3 Macchina a stati del polling

Il polling implementato ha il seguente funzionamento:



3.4 Testing

Il testing del seguente package è stato effettuato per mezzo di un'analisi statica, utilizzando il tool Stan4J, e di un'analisi dinamica con Junit.

3.4.1 Analisi statica

L'analisi statica, come detto in precedenza, è stata effettuata per mezzo del tool Stan4J.
In particolar modo sono state analizzate le seguenti view: composition, couplings, pollution e distance. Tale analisi è stata effettuata principalmente sul package openhabBroker in quanto è quello che implementa il servizio di broker richiesto; il package item.openhabBroker contiene infatti solo la struttura dati Item e quindi l'analisi di esso non è rilevante per il progetto, mentre il package test.openhabBroker include un semplice main di prova del servizio di broker stesso.

Composition View

La composition view del nostro progetto mostra che esso è costituito da un package principale openhabBroker, che utilizza il contenuto del package item.openhabBroker, in cui è implementata la classe Item. Il package test.openhabBroker che a sua volta utilizza il package principale openhabBroker comprende solamente un main di prova del progetto stesso.

Figura 22: Composition View del progetto

Analizzando la composizione del package openhabBroker e delle classi da cui è formato abbiamo ottenuto i seguenti risultati:

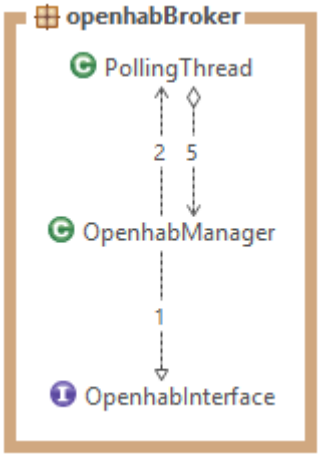
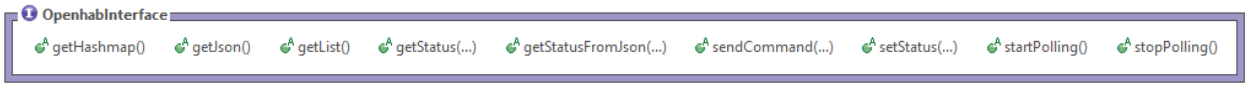
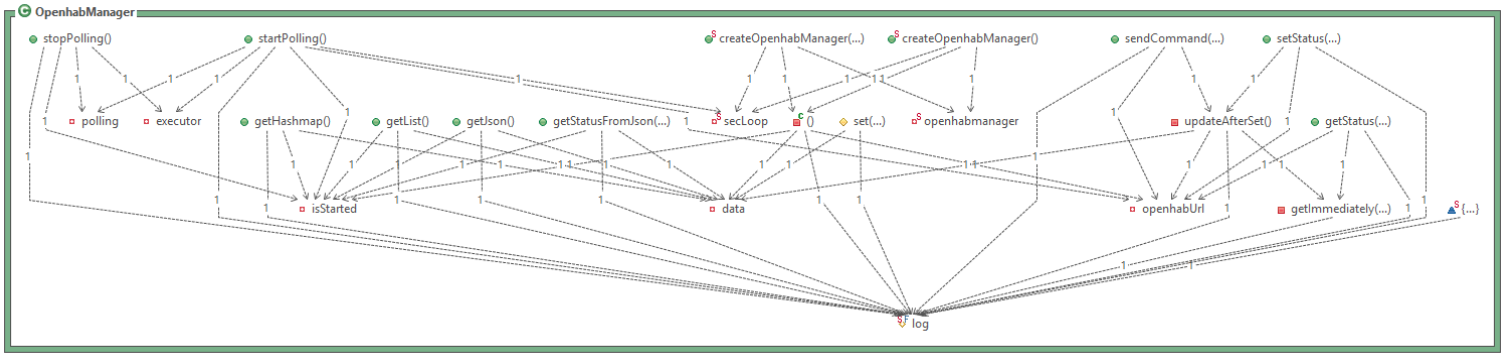


Figura 23: Composition view package openhabBroker

La composition view del package openhabBroker mette subito in evidenza la dipendenza ciclica tra la classe PollingThread e OpenhabManager, questa è presente in quanto l'oggetto PollingThread ha il solo scopo di fornire un servizio all'oggetto OpenhabManager.

Analizzando le composition views di ogni classe si ottengono i seguenti schemi:





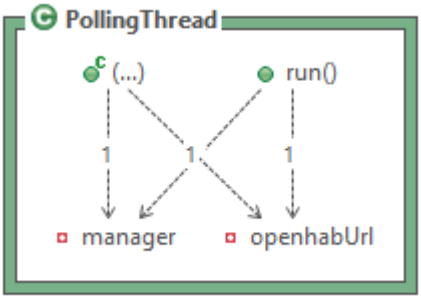


Figura 24: OpenhabInterface, OpenhabManager e PollingThead composition view

Come si poteva già immaginare la classe OpenhabManager, che gestisce il servizio di broker, è quella avente composizione più intricata. Tale situazione dipende dal fatto che quasi tutti i metodi fanno riferimento alle variabili principali: la struttura dati (JsonArray data), la variabile isStarted (mostra se il servizio di polling è attivo) e le variabili di log, con cui vengono scritti i log su file.

Coupling View

Per mezzo della coupling view è invece possibile analizzare le dipendenze di ogni elemento in entrata e in uscita, essa mostra quindi le volte che ogni classe ne richiama un'altra.

Anche da questa analisi è stato possibile notare la forte dipendenza tra la classe PollingThread e la classe principale OpenhabManager.

I risultati ottenuti per mezzo della coupling view sulla classe OpenhabManager mostrano come essa sia richiamata 5 volte dalla classe PollingThread che a sua volta viene chiamata 2 volte, inoltre la classe OpenhabManager richiama l'interfaccia OpenhabInterface, che implementa, e il package item.openhabBroker contenente la classe Item.

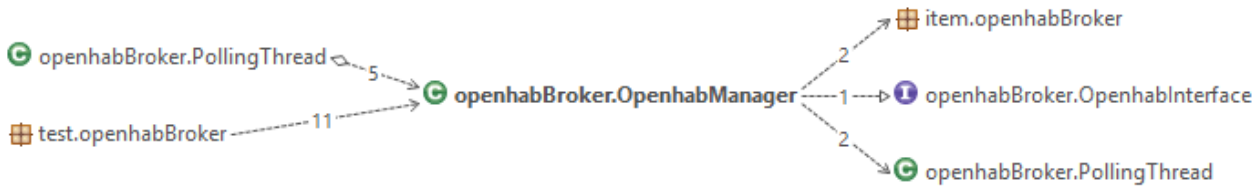


Figura 25: Coupling View della classe OpenhabManager

Figura 26: Coupling View della classe PollingThread

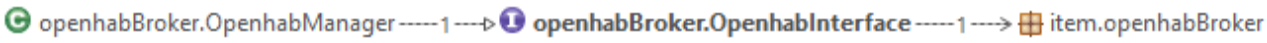


Figura 27: Coupling View della classe OpenhabInterface

Pollution

Le violazioni del progetto riguardano principalmente la distanza e la DIT (Depth of Inheritance Tree), tuttavia esse non dipendono dal package principale cardine del progetto (openhabBroker), quanto piuttosto dal package item.openhabBroker che implementa una semplice struttura dati. Infatti, analizzando la Pollution del solo package openhabBroker non viene evidenziata alcuna violazione.

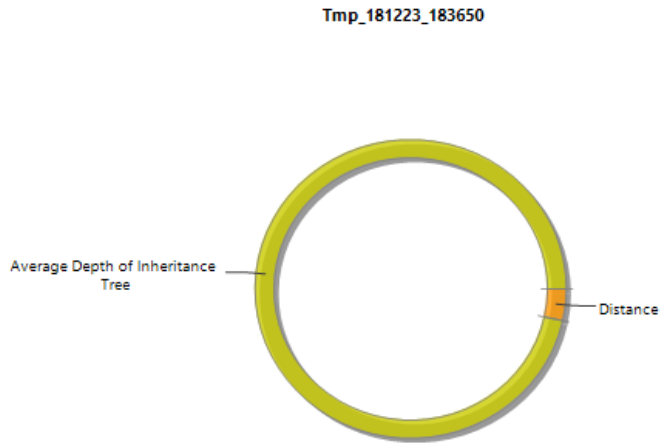


Figura 28: Pollution View

Analizzando inoltre la metrica DIT del progetto si può notare come il nostro progetto non si trovi in una pessima posizione, in quanto è abbastanza vicino alla parte verde del grafico:

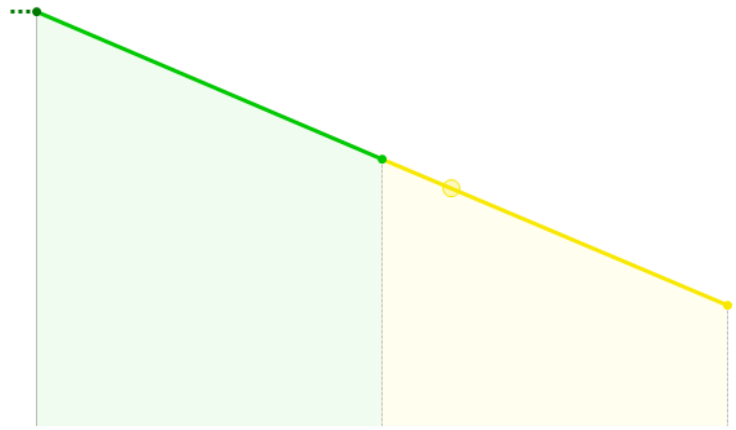


Figura 29: DIT graphic

Distance

Anche l'analisi della distanza mostra come il nostro progetto sia "equilibrato", esso infatti è molto vicino alla retta ideale di Instabilità e Astrazione.

Questo indica che il progetto è parzialmente estendibile, dato che è parzialmente astratto, ed inoltre tali estensioni non sono soggette a instabilità.

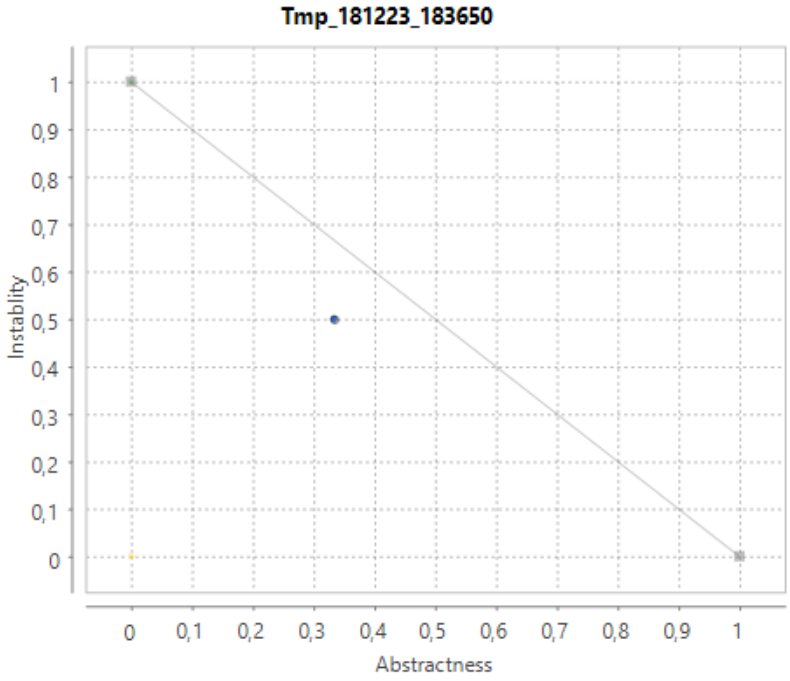


Figura 30: Distance Graphic

Abbiamo inoltre analizzato le diverse metriche, che il tool Stan4j offre per effettuare un'analisi statica completa, per il package openhabBroker, da questa analisi è stato quindi possibile appurare che il progetto realizzato non presenta delle particolari violazioni degli standard di qualità del software.

Category/Metric	Value
Count	
Units	3
Classes / Class	0
Methods / Class	9
Fields / Class	3.33
ELOC	315
ELOC / Unit	105
Complexity	
CC	2.07
Fat	3
ACD - Unit	66.67%
Robert C. Martin	
D	-0.17
A	0.33
I	0.5
Ca	1
Ce	1
Chidamber & Kemerer	
WMC	18.67
DIT	0.67
NOC	0
CBO	1
RFC	10.33
LCOM	0

Figura 31: Metriche del package openhabBroker

3.4.2 Analisi Dinamica

L'analisi dinamica è stata svolta per mezzo del tool Junit4. Con questo sono stati analizzati i vari metodi della classe OpenhabManager al fine di controllarne il corretto funzionamento e, nel caso sia previsto dal metodo, che restituiscano ciò che ci aspettiamo.

Per prima cosa abbiamo testato che la creazione dell'oggetto avvenga nel modo corretto e verificato che la classe sia veramente singleton.
Per far questo abbiamo introdotto una variabile di test all'interno del costruttore, incrementata ogni volta che il costruttore viene richiamato, e controllato che essa rimanga settata a 1 successivamente alla creazione di un nuovo oggetto di classe OpenhabManager (dato che la classe è singleton il costruttore viene richiamato solo una volta).
Questo test è stato eseguito anche per la variante che prevede il passaggio di un intero (rappresentante i secondi di polling).

Successivamente siamo passati a testare i metodi che si occupano di recuperare le informazioni (lo stato degli oggetti) da openhab. Questi sono i metodi:

- getJson()
- getList()
- getStatus(String itemName)
- getStatusFromJson(String itemName)
- getHashmap()

Per testare il metodo getJson() abbiamo dovuto per prima cosa far partire il polling, in quanto è questo che si occupa di aggiornare il file json che il metodo restituisce. In questo modo abbiamo anche testato il corretto funzionamento di quest'ultimo. Abbiamo quindi controllato che la stringa restituita dal metodo non risultasse nulla e l'abbiamo quindi confrontata con un'altra stringa, ottenuta interrogando direttamente openhab, con il fine di dimostrare che le due fossero uguali.

Il metodo getList() è stato testato controllando che all'interno della lista restituita siano presenti gli oggetti e che questi si trovino nello stato corretto.

Per il metodo getStatus(String itemName), che restituisce lo stato di un oggetto prendendolo direttamente dal file json, abbiamo creato una stringa, a partire dal file json restituito dal metodo getJson() testato in precedenza, contenente oggetti e stati e ne abbiamo eseguito il parse. Abbiamo quindi controllato la lista così creata fino a trovare l'oggetto desiderato e abbiamo controllato che lo stato fosse uguale a quello restituito dal metodo getStatus.

Il metodo getStatus(String itemName) è stato inoltre utilizzato per testare i metodi getStatusFromJson(String itemName) e getHashmap(). Il primo controllando che lo stato restituito da entrambi i metodi fosse il medesimo, il secondo controllando che lo stato di vari oggetti all'interno della hashmap fosse corretto.

Siamo poi passati a testare i metodi che si occupano di aggiornare lo stato degli item all'interno di openhab. Questi sono:

- setStatus(String item, String newState)
- sendCommand(String item, String newState)

Per questi la strategia di test utilizzata è stata la medesima, ovvero settare il valore di un item e controllare che lo stato dello stesso fosse effettivamente cambiato.
Abbiamo inoltre controllato che, provando a utilizzare il comando sbagliato per settare il valore di un item, venisse sollevata l'eccezione correttamente.

In fine abbiamo verificato che il comando stopPolling() arrestasse nella maniera corretta il thread di polling verificando che, dopo un certo periodo di tempo, non vi fossero cambiamenti nelle variabili restituite dei metodi precedenti.

Tutti i casi di test eseguiti si sono conclusi con successo.