ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

TESI DI LAUREA

in

Fondamenti di informatica T2

Scenari applicativi per Smart Home su piattaforma Home Manager: verso il livello intelligence

CANDIDATO:
Erika Gardini

RELATORE:
Chiar.mo Prof. Enrico Denti

CORRELATORE:
Dott. Ing. Roberta Calegari

Anno Accademico 2015/2016
Sessione II

Indice

Introduzione

La domotica[1], ovvero l'applicazione della tecnologia all'interno della casa, ha lo scopo di migliorare la qualità della vita delle persone, delegando varie mansioni a "dispositivi intelligenti" – ad esempio, tramite sensori di luminosità esterna persiane "intelligenti" potrebbero in autonomia aprirsi al sorgere del sole.

L'aumento dell'interesse nei confronti della domotica ha portato allo sviluppo dei primi prototipi capaci di ricevere comandi, acquisire informazioni di natura eterogenea da sensori, prendere decisioni ed effettuare azioni sull'abitazione. Fra questi prototipi rientra Home Manager.

Home Manager[3] è una piattaforma sperimentale per il controllo di una casa intelligente, è sviluppato come un sistema a multi-agent ed è implementato sull'infrastruttura di coordinazione TuCSoN[5].

Appoggiandosi su tale piattaforma, che offre già oggi una serie di agenti e servizi prototipali, è possibile sperimentare svariati scenari innovativi di interesse.

Il sistema Home Manager è stato recentemente realizzato sull'architettura Butlers[7], che vede l'abitazione come entità che conosce le preferenze e le abitudini degli abitanti e le sfrutta per prendere decisioni autonome e/o per anticipare le loro esigenze.

In quest'ottica, quindi, risultano particolarmente rilevanti gli scenari le cui operazioni variano autonomamente in base alle informazioni ed allo stato del sistema al momento della loro invocazione, allo scopo di soddisfare il più possibile le esigenze dell'abitante della casa.

Gli obiettivi che si vogliono raggiungere con questa tesi sono i seguenti:

- identificare una serie di scenari interessanti, sviluppabili su Home Manager, non solo basandosi sul suo stato attuale, ma considerando componenti ancora in fase di sviluppo o non ancora sviluppati;
- progettare ed implementare alcune delle funzionalità previste dagli scenari, quelle maggiormente significative;
- collaudare le funzionalità progettate ed implementate al punto precedente mediante la realizzazione di agenti simulati e fisici.

Gli obiettivi sopra descritti saranno raggiunti per gradi. La tesi sarà quindi strutturata come segue:

- nel primo capitolo verrà introdotto il concetto di sistema domotico. Particolare attenzione verrà posta sugli elementi che compongono un sistema domotico, sulle proprietà che un sistema domotico deve avere al fine di essere ben realizzato e sugli scenari in un sistema domotico;

- nel secondo capitolo ci si concentrerà su Home Manager, inteso come piattaforma prototipale per Smart Home. In particolare si porrà l'attenzione sullo stato attuale del sistema, sull'infrastruttura TuCSoN su cui si basa il sistema stesso e sulla tecnologia Raspberry Pi, che ne consente il deployment anche su dispositivi a basso costo;
- nel terzo capitolo verranno individuati ed analizzati in modo approfondito alcuni possibili scenari applicabili ad Home Manager;
- nel quarto capitolo si focalizzerà l'attenzione sulle funzionalità particolarmente significative degli scenari proposti, quelle riguardanti la gestione dei consumi e l'allineamento dei dispositivi, e ci si occuperà dell'analisi dei requisiti e del problema, nonché della relativa progettazione;
- nel quinto capitolo si tratterà l'implementazione delle funzionalità analizzate e progettate nel capitolo 4, sia nel sistema Home Manager che su tecnologia Raspberry Pi, e si collauderà quanto è stato implementato;
- infine, nel sesto capitolo si presenteranno i possibili sviluppi futuri.

Capitolo 1 – Verso Smart Home

1.1 La domotica nelle abitazioni

Il principale obiettivo per un sistema domotico[1] è quello di rendere l'abitazione a misura e a servizio dell'uomo, al fine di migliorarne il quotidiano rendendo più semplici le azioni di tutti i giorni.

Un altro obiettivo per un sistema domotico[1] è, inoltre, quello di aumentare il risparmio energetico ed economico dell'abitazione.
Infatti, in un'abitazione, sono gli elettrodomestici bianchi (lavatrici, frigoriferi, forni, …) a costituire la maggior parte dei consumi di energia elettrica.
Il sistema domotico potrebbe controllare l'attivazione degli elettrodomestici in modo che essa si verifichi, quando possibile, nelle fasce orarie in cui l'energia elettrica è meno costosa.
In un'abitazione dotata di sistema domotico, inoltre, ciascun elettrodomestico potrebbe essere dotato di intelligenza propria. In tal caso potrebbe scambiarsi informazioni con gli altri elettrodomestici dell'abitazione e potrebbe essere in grado di capire se può accendersi o meno senza causare nessun sovraccarico energetico, quindi evitando scatti del salvavita.
Infine, un sistema domotico potrebbe essere in grado di segnalare malfunzionamenti dell'abitazione, come ad esempio perdite, così da poter intervenire immediatamente, da impedire consumi troppo elevati di acqua o gas e da mettere in sicurezza le persone all'interno dell'abitazione.

Le funzioni domotiche[2], in sostanza, sono automatismi quotidiani che semplificano la vita dell'utente, liberandolo da procedure, menù o dalla ricerca affannosa di telecomandi e controlli vari. Infatti, grazie al sistema domotico nell'abitazione, l'utente potrà inviare comandi al sistema ed a tutti i dispositivi dell'abitazione attraverso, per esempio, uno smartphone o un tablet, utilizzando le tecnologie wireless.

1.2 Sistema di controllo dell'abitazione

Alcuni sistemi domotici potrebbero essere dotati di un sistema di controllo[1].
Il sistema di controllo dell'abitazione, se presente, rappresenta una sorta di "intelligenza locale" attraverso la quale è possibile monitorare e controllare tutte le funzioni esistenti (edificio, impianti, utenza, climatizzazione, …) in tempo reale, considerando tutte le interazioni ed ottimizzando le prestazioni complessive secondo criteri prefissati o perfezionabili nel tempo.

Se il sistema di controllo è presente ad esso viene assegnato il compito di controllare che tutto il sistema funzioni correttamente, indicare all'utente

situazioni anomale se presenti e consentire l'inserimento di comandi per l'attivazione di funzioni.

Affinché sia possibile inserire comandi, il sistema di controllo è generalmente dotato di una interfaccia. Quest'ultima può essere comodamente mostrata in uno schermo posizionato all'interno dell'abitazione, magari touch screen.

1.3 Infrastruttura di comunicazione in un sistema domotico

Un elemento di fondamentale importanza per l'esistenza di un sistema domotico è la presenza di una rete di comunicazione[1].
E' tramite la rete di comunicazione che tutti i dispositivi dell'abitazione possono dialogare fra di loro, è tramite la rete di comunicazione che il sistema può intervenire in modo completo e molto agevole in caso di malfunzionamenti, inviando comandi automatici al/ai dispositivo/i causa del guasto, ed è tramite la rete di comunicazione che i comandi inseriti dall'utente possono essere inviati al dispositivo o ai dispositivi in questione.
La possibilità per i vari dispositivi domestici di comunicare consente di aggiungere nel sistema quante più funzionalità si vogliono, in modo estremamente personalizzato.

I nuovi sistemi domotici, oltre che ad offrire una rete di comunicazione locale, offrono soluzioni d'avanguardia per la gestione ed il controllo degli impianti di casa anche a distanza, in remoto, mediante l'utilizzo di software per pc, tablet o smartphone.

1.4 Proprietà di un sistema domotico ben progettato

In fase di realizzazione di un sistema domotico potrebbe essere utile conoscere una serie di proprietà che, se rispettate, potrebbero consentire la riuscita del sistema domotico stesso.

Un sistema domotico dovrebbe essere realizzato dopo una attenta valutazione delle reali necessità dell'utente. In particolare, bisognerebbe prestare attenzione a quelli che sono i "bisogni latenti".
Inoltre, un sistema domotico dovrebbe aumentare l'autonomia della persona e consentire l'accesso ad un mondo sempre più ampio di servizi.

Un sistema domotico dovrebbe essere dotato di interfacce facilmente comprensibili e gestibili da qualsiasi tipo di utente.
Entrando maggiormente nel dettaglio, le interfacce per l'utilizzo del sistema dovrebbero essere intuitive per l'utente finale e pensate per la maggior facilità di utilizzo: userfriendly.
Una buona interfaccia dovrebbe, quindi, possedere i seguenti requisiti:

- versatilità d'uso, ossia dovrebbe essere adattabile ad utenti diversi

ed a situazioni diverse;
- navigabilità, ossia dovrebbe permettere all'utente di muoversi all'interno dei vari menù in modo rapido e agevole;
- riconoscibilità immediata, ossia dovrebbe adottare simboli chiaramente esplicativi della funzionalità svolta.

Un sistema domotico dovrebbe, inoltre, basarsi su due parole chiavi: integrazione e interoperabilità.
Un sistema domotico che si basa sulla parola chiave integrazione è sempre aperto all'aggiunta di componenti mancanti, così che possa essere migliorabile, ma l'aggiunta di nuovi componenti non comporta la modifica dei componenti già esistenti.
Un sistema così realizzato si definisce "open/close": è un sistema aperto per le estensioni, ma chiuso per le modifiche.
Un sistema domotico che si basa sulla parola chiave interoperabilità deve funzionare con altri prodotti o sistemi, esistenti o in fase di sviluppo, senza alcuna restrizione per l'accesso o le implementazioni.

I sistemi domotici dovrebbero, infine, presentare caratteristiche di affidabilità e modularità.
Un sistema domotico si definisce affidabile se le funzionalità offerte corrispondono alle reali necessità dell'utente (correttezza) e se, in caso di guasto, non produce danni fisici o economici (robustezza).
Un sistema si definisce modulare se è formato da singoli elementi separabili o separati.

1.5 Gli scenari in una Smart Home

Lo scenario[1] è una particolare condizione dei dispositivi controllati dalla domotica che è stata memorizzata nel sistema.
Per scenario si intende, quindi, una serie di operazioni programmate per essere attivate da un solo comando.
Ogni singolo scenario deve essere configurato secondo le necessità dell'utente.

Ciascun utente può quindi modificare il sistema affinché si comporti in una certa maniera quando si verifica un certo evento.

Mediante gli scenari è possibile attivare/disattivare o modificare lo stato di più dispositivi del sistema; il tutto avviene automaticamente indicando l'attivazione dello scenario stesso.

Esempi di scenario sono i seguenti:
- scenario notte: tutte le tapparelle abbassate, l'allarme inserito, le porte chiuse e magari il sistema di climatizzazione attivato come stabilito dall'utente;

- scenario film: la televisione si accende, l’impianto surround si accende, le luci della stanza si spengono.

Richiamare uno scenario non è altro che riprodurre la scena precedentemente memorizzata. Uno scenario può essere richiamato con la pressione di un pulsante, con il click su un'icona, a tempo mediante una programmazione ciclica oraria, settimanale o mensile, oppure al verificarsi di una certa condizione data dallo stato di altri impianti del sistema.

Capitolo 2 – Smart Home in Home Manager

2.1 Home Manager

Home Manager[3] è un prototipo di sistema di controllo per una Smart Home. Progettato in origine secondo una metodologia di progettazione agent-oriented (SODA[4]), è implementato al di sopra dell'infrastruttura TuCSoN[5].

SODA (Societies in Open and Distributed Agent spaces)[4] è una metodologia di analisi e progettazione di un sistema complesso agent-based. Essa non si occupa della definizione degli agenti in termini di comportamento richiesto e ruolo nel sistema multi-agente, ma si concentra su come deve essere realizzata l'infrastruttura di un sistema multi-agent.

Home Manager è stato realizzato su piattaforma Java. E' poi stato portato su piattaforma Raspberry ed è stato parzialmente integrato con Win10-IoT; in questo modo è stato possibile svincolarsi dalla presenza di pc stand alone ed è stato possibile sfruttare la possibilità di connettere sensori.

Recentemente si è scelto di realizzare il sistema Home Manager secondo un'architettura Butlers[7], che mira ad ottenere un sistema in grado di anticipare i bisogni dell'utente e di agire in modo autonomo in base alle proprie conoscenze.

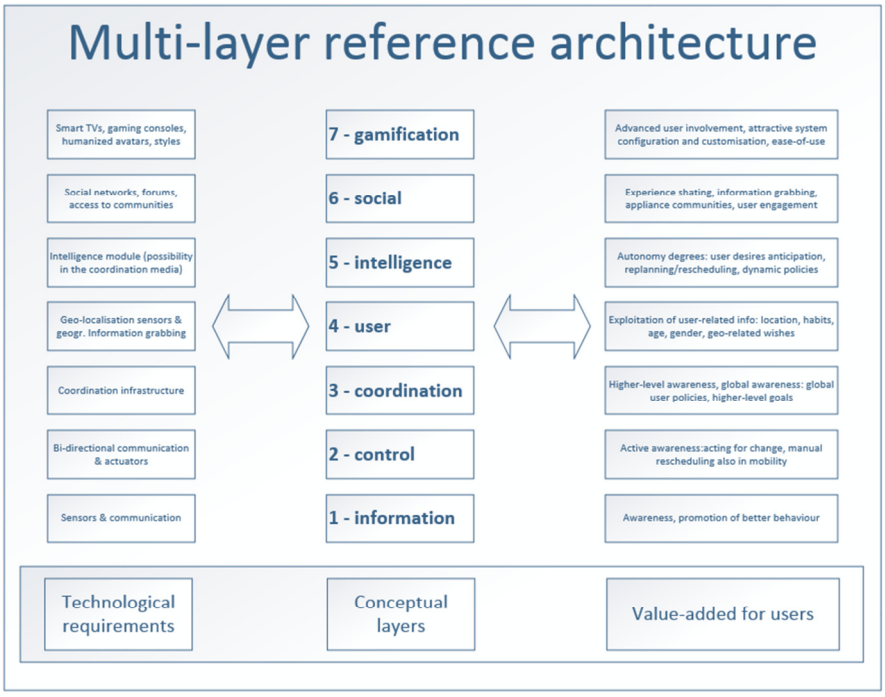


Figura 1: requisiti tecnologici (sinistra), livelli concettuali (centro), servizi offerti all'utente (destra)

L'architettura Butlers si articola nei seguenti livelli:

- information: consente di ottenere informazioni fisiche riguardanti il sistema;
- control: aggiunge una forma di controllo remoto così da assegnare una sorta di automazione all'abitazione;
- coordination: consente la comunicazione e la coordinazione dei componenti del sistema;
- user: aggiunge tutte le informazioni che riguardano l'utente e le utilizza per stabilire come comportarsi;
- intelligence: aggiunge la capacità di anticipare i desideri e le decisioni dell'utente, grazie all'analisi di tutte le informazioni e le risorse del sistema;
- social: permette al Butler di accedere alle informazioni sui social network, aumentando così le sue prestazioni;
- gamification: unito al livello social rende umanizzato il sistema.

2.1.1 Il prototipo attuale

Home Manager, allo stato attuale, rappresenta una sorta di virtualizzazione dell'abitazione e più in generale, tramite la geo-localizzazione supportata dall'app su smartphone, dell'ambiente cittadino in cui l'utente vive.
L'abitazione simulata è composta da quattro stanze: ingresso, sala da pranzo e cucina, bagno e camera.

In ciascuna delle stanze dell'abitazione si suppone la presenza di sensori che consentono la localizzazione e l'identificazione dei soggetti che abitano la casa o che ne sono visitatori.
Ciascun utente dell'abitazione può, mediante appositi terminali, identificarsi esplicitamente; questa operazione, se effettuata, consente al sistema di assecondare le preferenze dell'utente identificato.
Inoltre, ciascuna persona nell'abitazione ha assegnato un ruolo: amministratore, utente oppure visitatore. In base al ruolo assegnato si ha la possibilità di compiere tutte oppure un sottoinsieme di operazioni nel sistema. In particolare: gli amministratori hanno il pieno controllo, quindi possono specificare il comportamento del sistema ed i privilegi di utenti e visitatori, gli utenti possono esprimere le proprie preferenze e possono inviare comandi al sistema, quindi non possono modificare il comportamento del sistema o assegnare privilegi, i visitatori non hanno nessun privilegio, ma solo un'assistenza di base.

In ciascuna delle stanze dell'abitazione sono presenti dei dispositivi. In particolare, allo stato attuale, sono presenti una lavatrice nel bagno, una

televisione nella camera ed in cucina, un frigorifero in cucina, un forno in cucina ed uno stereo in cucina.
Infine, sono presenti attuatori per la regolazione dell'impianto luminoso e la climatizzazione.

Inoltre, nel sistema Home Manager attuale, è stata assegnata una prima forma di intelligenza ai dispositivi forno e frigorifero presenti nella cucina dell'abitazione.
Ciascuno dei due dispositivi è affiancato da un agente, che lo gestisce e gli permette di relazionarsi con il sistema come membro di una società di agenti. Nel caso specifico, il forno è affiancato da un agente chiamato OvenAgent, mentre il frigorifero è affiancato da un agente chiamato FridgeAgent. Gli agenti sono a loro volta collegati ciascuno ad un'interfaccia utente, che consente l'esecuzione delle operazioni e mostra i risultati delle stesse.

Allo stato attuale, il forno:

- contente l'inserimento, la modifica e la memorizzazione di ricette;
- invia una richiesta di controllo degli ingredienti al frigorifero, nel caso in cui un utente richiede la cottura di una ricetta, ed attende la risposta;
- cuoce la ricetta se il frigorifero ha inviato una risposta affermativa al controllo al punto precedente, altrimenti richiede all'utente se vuole generare una lista della spesa;
- invia una richiesta di creazione della lista della spesa al frigorifero se gli ingredienti di una ricetta non sono presenti e se l'utente lo ha richiesto.

Il frigorifero, invece, allo stato attuale offre le seguenti funzionalità:

- consente l'inserimento, la modifica e la cancellazione di ingredienti;
- verifica la presenza degli ingredienti di una ricetta quando richiesto dal forno;
- invia al forno l'esito dell'operazione di verifica al punto precedente, che consente o meno di proseguire con la cottura;
- crea le liste della spesa, quando richiesto dal forno, e le inoltra all'agente ShopperAgent che si occupa di gestirle.

Quanto appena descritto rappresenta una prima forma di comunicazione fra agenti, nonché un primo scenario: lo scenario SmartKitchen. E' infatti evidente come l'inserimento del comando cottura da parte dell'utente scateni una serie di altre operazioni compiute in modo automatico dal sistema.

Il sistema Home Manager si sta quindi sempre più muovendo verso il livello quattro dell'architettura Butler, quello di intelligence, che è dotato ora di una prima capacità decisionale autonoma.

Infine, è stato di recente aggiunto al sistema il Device Manager Agent. Questo agente si occupa dell'auto-detect dei dispositivi collegati al sistema. In altre parole, il Device Manager Agent si occupa, dinamicamente, di mostrare nell'interfaccia del sistema Home Manager i dispositivi connessi al sistema, in modo tale che l'utente possa interagire con loro. Allo stato iniziale si suppone che il sistema sia dotato di almeno una versione simulata per ogni tipologia di dispositivo gestibile da Home Manager. Ogni qual volta arriva un nuovo dispositivo fisico, esso richiede un nome al Device Manager Agent, in modo tale che da lì in avanti possa essere riconosciuto all'interno del sistema.
Il Device Manager Agent, dopo aver assegnato un nome al dispositivo fisico, si occupa di generare un simulato corrispondente in modo tale che, in caso di distacchi dal sistema dello stesso fisico, rimanga sempre presente il simulato. Si occupa, inoltre, di aggiornare la lista dispositivi mostrata nell'interfaccia grafica.

In fase di realizzazione del DeviceManagerAgent è stata aggiunta al sistema una classe astratta per la comunicazione, chiamata AgentCommunicationLanguage. Questa classe astratta contiene al suo interno una serie di costanti con i rispettivi metodi getters e consente di non cablare nel codice, e quindi negli agenti, il formato delle tuple di comunicazione, in modo tale che possa essere facilmente modificato.

2.2 L'infrastruttura ad agenti TuCSoN

TuCSoN[5] (Tuple Centres Spread over the Network) è l'infrastruttura di coordinazione ad agenti utilizzata in Home Manager.
La comunicazione e la coordinazione tra agenti avviene attraverso i centri di tuple, ovvero spazi di informazioni reattivi, condivisi e distribuiti sui nodi.
E' nei centri di tuple che gli agenti depositano, prelevano e leggono informazioni sotto forma di tuple, ossia collezioni ordinate di porzioni di informazioni eterogenee.

L'infrastruttura TuCSoN[6] si basa su tre entità principali:

- agenti: sono le entità del sistema distribuito da coordinare;
- centri di tuple: sono il mezzo attraverso il quale avviene la comunicazione;
- nodi: sono l'astrazione topologica di base; è sui nodi che si trovano i centri di tuple.

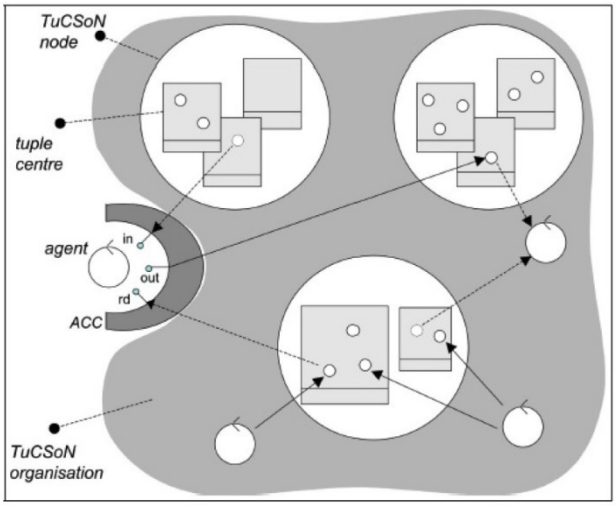


Figura 2: sistema Home Manager in base all'infrastruttura TuCSoN

I protagonisti dell'interazione sono in TuCSoN gli agenti ed i centri di tuple.
I primi hanno un ruolo attivo, in quanto sfruttano i centri di tuple per la comunicazione depositando, prelevando e leggendo in/da essi tuple mediante le primitive di lettura e scrittura.
I secondi, invece, hanno un ruolo reattivo: forniscono agli agenti uno spazio condiviso per la comunicazione e possono essere anche programmati dinamicamente affinché reagiscano a determinate operazioni sulle tuple che contengono.
Il modello topologico adottato da TuCSoN è di tipo client-server, dove i clienti sono gli agenti ed i server sono i centri di tuple.

TuCSoN, oltre che essere un middleware Java-based, è anche Prolog-based. Quest'ultimo aspetto gli da la possibilità di appoggiarsi a tuProlog, interprete Prolog leggero e Java-based, per compiere operazioni di vario genere, fra le quali rientrano la definizione delle tuple e dei loro template.

I centri di tuple possono essere aggregati in organizzazioni. A ciascun agente viene assegnata un'interfaccia, ACC (Agent Coordination Context), che gli consente di effettuare le operazioni sui centri di tuple che appartengono ad una specifica organizzazione. Ciascun agente avrà, quindi, un ACC diverso per ciascuna organizzazione.
In questo modo è possibile fornire a ciascun agente il mezzo per la comunicazione con i centri di tuple, ma al tempo stesso è possibile limitare la comunicazione di un agente solo alle organizzazioni di cui possiede l'interfaccia.

Ogni entità del sistema TuCSoN deve essere definita univocamente affinché sia possibile indirizzarla in ogni momento.
Ogni nodo è definito univocamente dalla coppia <indirizzoIP, porta>, dove la porta di default è 20504.

Ogni centro di tuple è definito univocamente dalla tripla <indirizzoIP, porta, nome>, dove i primi due elementi identificano il nodo in cui si trova il centro di tuple, mentre il nome non ha particolari caratteristiche da rispettare se non l'univocità all'interno del nodo.
Ogni agente è definito univocamente da un nome generico arbitrario e da uno universally unique identifier (UUID) assegnatogli al momento dell'ingresso nel sistema TuCSoN.

2.2.1 La primitive per la comunicazione

Affinché un agente possa comunicare e sincronizzarsi, deve effettuare delle operazioni di scrittura o lettura di tuple verso un centro di tuple[6].

Le operazioni possono essere suddivise, in base alla fase in cui avvengono, in:

- *invotation*: richiesta inviata da un agente ad un centro di tuple, contiene tutte le informazioni necessarie al centro di tuple per effettuare l'operazione;
- *completion*: contiene il risultato dell'operazione ed informazioni relative all'esecuzione.

Le primitive di comunicazione definite dal linguaggio di coordinazione TuCSoN sono:

- *out*, scrive una tupla specifica in un centro di tuple;
- *in*, elimina una tupla specifica da un centro di tuple;
- *rd*, legge una tupla specifica in un centro di tuple;
- *no*, cerca una tupla specifica in un centro di tuple e restituisce esito positivo se non la trova.

Le tre primitive sopra indicate sono sincrone: l'agente si blocca in attesa della tupla specificata.
Oltre alle tre primitive mostrate, esistono le seguenti:

- *inp*, elimina una tupla specifica da un centro di tuple;
- *rdp*, legge una tupla specifica in un centro di tuple;
- *nop*, cerca una tupla specifica in un centro di tuple e restituisce esito positivo se non la trova.

Le tre primitive sopra indicate sono asincrone: l'agente non si blocca in attesa della tupla specificata e, nel caso in cui non sia possibile effettuare l'operazione richiesta per mancanza della tupla, viene restituito esito negativo.
Infine, esistono le seguenti primitive:

- *get*, restituisce una lista di tutte le tuple presenti nel centro di tuple;
- *set*, sovrascrive le tuple del centro di tuple con quelle passate come argomento;

- *out_all*, scrive una serie di tuple specifiche in un centro di tuple;
- *rd_all*, legge tutte le tuple in un centro di tuple che hanno le caratteristiche della tupla specificata;
- *in_all*, elimina tutte le tuple da un centro di tuple che hanno le caratteristiche della tupla specificata.

2.2.2 Le principali API

Le principali Java API messe a disposizione dalla libreria TuCSoN[6] sono le seguenti:

- *TucsonAgentId*: espone i metodi che consentono di ottenere l'identificativo di un agente, di accedere ai suoi campi e di ottenere un ACC;
- *TucsonMetaACC*: espone i metodi per ottenere agenti TuCSoN con un ACC;
- *TucsonTupleCentreId*: espone i metodi che consentono di ottenere l'identificato di un centro di tuple, accedere ai suoi campi, invocare le operazioni sull'ACC;
- *ITucsonOperation*: espone i metodi che consentono di accedere ai risultati di una operazione. Fra i più importanti metodi, si elencano:
  - `isResultSuccess():Boolean`, per verificare il successo di un'operazione;
  - `getLogicTupleResult():LogicTuple`, per ottenere la tupla logica risultato di un'operazione;
  - `getLogicTupleListResult():List<LogicTuple>`, per ottenere una lista di tuple logiche, risultato di un'operazione;
- *TucsonAgent*: classe base astratta che mette a disposizione metodi per la creazione di agenti, crea automaticamente un *TucsonAgentId* ed ottiene un *EnhancedACC*. Fra i più importanti metodi, si elencano:
  - `main():void`, è un metodo da riscrivere nella classe che estende la classe astratta e consente di inserire la logica dell'agente;
  - `getContext():EnhancedACC`, è il metodo che consente di ottenere l'ACC;
  - `go():void`, è il metodo che esegue il `main()`;
- *LogicTuple*: espone i metodi per la manipolazione e l'utilizzo di tuple logiche;
- *TupleArgument*: espone i metodi per accedere agli argomenti delle tuple logiche.

2.3 La tecnologia Raspberry Pi

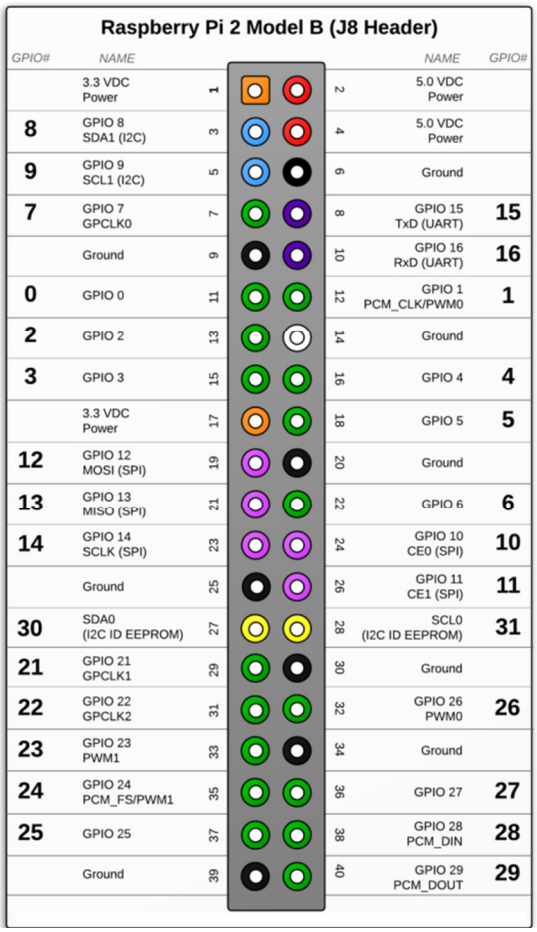
Il Raspberry Pi[8] è un piccolo personal-computer realizzato e commercializzato dalla Raspberry Foundation. Viene fornito con una versione del sistema operativo Linux: Raspbian.

Figura 3: Raspberry Pi

Il Raspberry Pi è una piccola scheda elettronica sulla quale sono montati diversi connettori: porta micro-USB per l'alimentazione, porte USB, uscita video HDMI, jack audio, porta di rete Ethernet, slot per SD card.
L'SD card inserita nel Raspberry Pi rappresenta quello che nei normali dispositivi è il disco fisso.
Il "cuore" del Raspberry Pi è un chip (in realtà sono due chip montati uno sopra l'altro) che racchiude la RAM ed il processore.
In particolare, il processore è un Broadcom BCM2835, comunemente chiamato "System On Chip" (SOC). Si tratta di un unico componente che integra al suo interno il processore vero e proprio (basato sull'architettura ARM) ed una serie di unità periferiche, normalmente esterne.

Ciò che differenzia il Raspberry Pi da un normale computer è la presenza di un connettore GPIO. Il connettore GPIO è composto da 40 pin che consentono di collegare al Raspberry Pi una serie di dispositivi esterni, che permettono di acquisire informazioni sull'ambiente circostante tramite i sensori e di interagire con oggetti reali tramite gli attuatori. Oltre che a collegare dispositivi, è possibile controllare direttamente i pin di input e di output.
A seconda delle versioni di Raspberry Pi, le funzioni dei pin cambiano[10].
In fase di progettazione di questa tesi verrà utilizzato un Raspberry Pi 2, model B.

Le funzioni dei pin per il Raspberry Pi indicato sono le seguenti:



GPIO#	NAME			NAME	GPIO#
	3.3 VDC Power	1	2	5.0 VDC Power	
8	GPIO 8 SDA1 (I2C)	3	4	5.0 VDC Power	
9	GPIO 9 SCL1 (I2C)	5	6	Ground	
7	GPIO 7 GPCLK0	7	8	GPIO 15 TxD (UART)	15
	Ground	9	10	GPIO 16 RxD (UART)	16
0	GPIO 0	11	12	GPIO 1 PCM_CLK/PWM0	1
2	GPIO 2	13	14	Ground	
3	GPIO 3	15	16	GPIO 4	4
	3.3 VDC Power	17	18	GPIO 5	5
12	GPIO 12 MOSI (SPI)	19	20	Ground	
13	GPIO 13 MISO (SPI)	21	22	GPIO 6	6
14	GPIO 14 SCLK (SPI)	23	24	GPIO 10 CE0 (SPI)	10
	Ground	25	26	GPIO 11 CE1 (SPI)	11
30	SDA0 (I2C ID EEPROM)	27	28	SCL0 (I2C ID EEPROM)	31
21	GPIO 21 GPCLK1	29	30	Ground	
22	GPIO 22 GPCLK2	31	32	GPIO 26 PWM0	26
23	GPIO 23 PWM1	33	34	Ground	
24	GPIO 24 PCM_FS/PWM1	35	36	GPIO 27	27
25	GPIO 25	37	38	GPIO 28 PCM_DIN	28
	Ground	39	40	GPIO 29 PCM_DOUT	29

Figura 4: la funzione dei pin per Raspberry Pi 2 Model B

Il connettore GPIO ha dei pin di alimentazione, a 3,3V e a 5V, con i rispettivi pin di massa (GND). Inoltre, ci sono dei pin GPIO, ossia dei pin di ingresso/uscita digitale.
Tutti gli altri pin possono essere utilizzati per far comunicare il Raspberry con altri tipi di dispositivi, ma possono essere usati anche come normali pin GPIO.
Dispositivi e sensori possono essere collegati direttamente ai pin mediante cavi jumper femminafemmina, oppure si possono usare delle breadboard. In quest'ultimo caso è necessario riportare i segnali del connettore sulla basetta, usando cavi jumper femminamaschio oppure mediante un adattatore.
E' inoltre possibile collegare al Raspberry Pi sensori che catturano valori in formato analogico grazie ad un dispositivo chiamato breakout board, che funge da convertitore analogico/digitale.

2.3.1 Libreria Pi4J

La libreria Pi4J[9] offre una serie di API per Java. Essa consente di gestire dispositivi di input/output collegati al Raspberry Pi in modo facile ed immediato, senza doversi così preoccupare degli aspetti di basso livello.

Le principali API messe a disposizione sono le seguenti:

- *GpioFactory*, è una factory sulla quale è possibile invocare il metodo `getInstance()` per ottenere una istanza di *GpioController*;
- *GpioController*, espone una serie di metodi per la creazione di pin digitali. Di particolar importanza è il metodo `provisionDigitalOutputPin(Pin pin, String name, PinState defaultState):GpioPinDigitalOutput`, con il quale è possibile creare un sensore di output;
- *GpioPinDigitalOutput*, espone una serie di metodi per modificare lo stato del sensore di ouput.

2.3.2 La tecnologia Raspberry Pi in Home Manager

Affinché sia possibile utilizzare la tecnologia Raspberry Pi per la realizzazione di dispositivi fisici, è necessario che nel Raspberry Pi (che si suppone già dotato di sistema operativo) sia installata una versione aggiornata di Java. In questo modo sarà possibile eseguire TuCSoN e far funzionare l'applicativo Home Manager.

Capitolo 3 – Scenari applicativi in Home Manager

Dopo aver analizzato il concetto di scenari in una abitazione dotata di sistema domotico e dopo aver analizzato nel dettaglio il sistema Home Manager e l'infrastruttura di comunicazione su cui si appoggia (TuCSoN), è possibile procede con l'analisi dei possibili scenari in Home Manager.

3.1 La nuova idea di cucina

Un possibile scenario in ottica Home Manager, pensando anche ai dispositivi già presenti nell'abitazione e già in parte sviluppati, potrebbe essere quello di pensare ad una cucina dotata di intelligenza, ossia una cucina che "sappia", in seguito ad un comando inserito dall'utente ed in base alle preferenze dell'utente, prendere delle decisioni in modo automatico.

Una delle funzionalità che la cucina di Home Manager già offre, come esposto nel capitolo 2, è quella di consentire l'inserimento, la modifica e la cancellazione di ricette. Questa operazione potrebbe facilmente avvenire attraverso uno schermo touch screen inserito all'interno della cucina stessa.

A partire da questa funzionalità si potrebbe pensare ad una serie di altre funzionalità da poter aggiungere al sistema.

3.1.1 Le richieste di preparazione di una ricetta

Una prima funzionalità potrebbe essere chiamata "*richiesta preparazione ricetta*" e viene attivata direttamente dall'utente che, mediante l'interfaccia utente, richiede la preparazione di una delle ricette memorizzate dal sistema o di una ricetta inserita, ma non ancora memorizzata.
La funzionalità indicata comporta il susseguirsi di una serie di operazioni automatiche. Nel caso specifico, sarebbe richiesto di scatenare un'operazione di verifica della presenza degli ingredienti nel frigorifero al fine di stabilire se la ricetta richiesta possa essere o meno correttamente realizzata. Inoltre, nel caso in cui non fosse possibile realizzare correttamente la ricetta, verrebbe scatenato automaticamente un processo di valutazione delle alternative: in base agli ingredienti presenti nel frigorifero, il sistema dovrebbe essere in grado, se possibile, di proporre all'utente una ricetta equivalente a quella richiesta, ma con gli ingredienti riproporzionati sulla base delle quantità di cui dispone. Una ricetta non potrebbe, ovviamente, essere preparata se uno degli ingredienti non fosse presente nel frigorifero o se il sistema non fosse riuscito a calcolare un'alternativa.

Infine, il sistema dovrebbe mostrare all'utente l'esito dell'operazione e dovrebbe richiedere all'utente, sempre attraverso l'interfaccia grafica e solo nel caso in cui sia possibile, se vuole proseguire con la preparazione della ricetta o meno.

3.1.2 La preparazione di una ricetta

Quando l'utente richiede al sistema, mediante l'interfaccia grafica, di proseguire con la preparazione della ricetta, viene attivata la funzionalità "preparazione ricetta". Seppur non possa essere attivata direttamente, ma solo come conseguenza della funzionalità descritta sopra, la sua attivazione avviene sempre a seguito di una richiesta dell'utente.

La funzionalità appena descritta dovrebbe comportare l'accensione del forno alla temperatura della ricetta da preparare e l'accensione del timer, automaticamente impostato in base al tempo di cottura della ricetta. Inoltre, il frigorifero dovrebbe verificare che la preparazione della ricetta avvenga correttamente: dovrebbe, quindi, controllare che le quantità di ingredienti per la ricetta siano prelevate correttamente dal suo interno e dovrebbe segnalare all'utente errori nella preparazione o dimenticanze.

Terminata la preparazione della ricetta l'utente dovrebbe spostare il preparato nel forno.

3.1.3 La cottura della ricetta preparata

Quando l'utente sposta il preparato della ricetta nel forno, il forno rileva la presenza di un composto al suo interno e viene automaticamente avviata la funzionalità "*cottura ricetta*". La funzionalità dovrebbe comportare lo scatto del timer e lo spegnimento del forno allo scadere del tempo di cottura.

3.1.4 I "contenitori di ingredienti"

Nella funzionalità "*preparazione ricetta*" è il frigorifero che si occupa di verificare la corretta preparazione della ricetta, ma nel frigorifero non sono contenuti tutti gli ingredienti della cucina. Ciò che effettivamente si riesce a verificare è che tutti gli ingredienti contenuti nel frigorifero vengano prelevati correttamente, lasciando incontrollati gli altri.

La prima modifica da effettuare alla cucina al fine di superare il limite descritto dovrebbe consistere nell'aggiungere altri dispositivi.
Per prima cosa sarebbe necessario considerare altri "contenitori di ingredienti", come per esempio una dispensa. Quest'ultima potrebbe contenere tutti gli ingredienti che non vengono mantenuti in frigorifero

oppure le confezioni ancora non iniziate di ingredienti che si trovano in frigorifero.

Se venissero aggiunti altri "contenitori di ingredienti", sarebbe necessario modificare la funzionalità "*richiesta preparazione ricetta*", che non consisterebbe più nel verificare se il frigorifero contiene gli ingredienti della ricetta, ma dovrebbe svolgere delle operazioni per verificare se gli ingredienti della ricetta sono contenuti in cucina, o nel frigorifero, o nella dispensa o negli altri possibili "contenitori di ingredienti".
Sarebbe in questo caso necessaria una nuova entità, che dovrebbe occuparsi di interrogare ciascun "contenitore di ingredienti" e valutare il suo contenuto.
Non solo, un problema da non sottovalutare sarebbe la politica con cui la nuova entità dovrebbe interrogare i vari "contenitori di ingredienti". E' proprio a questo punto che intervengono gli interessi dell'utente, il quale dovrebbe aver indicato al sistema la modalità con cui desidera che gli ingredienti vengano rimossi dalla cucina.
La modalità di default potrebbe essere quella di partire dagli alimenti in frigorifero, le cui confezioni sono già state aperte e che vanno quindi consumati prima di quelli nella dispensa. Ciascun utente potrebbe, però, modificare tale modalità di rimozione in base alle proprie esigenze: un utente potrebbe specificare una modalità di rimozione in base alla categoria di ricette, e così via.

Affinché l'operazione di controllo della corretta preparazione della ricetta nella funzionalità "*preparazione ricetta*" sia effettivamente realizzabile, sarebbe, inoltre, necessario modificare la quantizzazione degli ingredienti effettuata dai "contenitori di ingredienti". Nella versione attuale di Home Manager, le quantità di ingredienti vengono considerate in unità. Sarebbe necessario, però, quantificare gli ingredienti a peso oppure, ancor meglio, dividere gli ingredienti in due categorie, una da quantificare a peso (farina, latte, …) ed una da quantificare in unità (uova, …).

3.1.5 Il preparatore di ricette

Per quanto riguarda l'entità che dovrebbe occuparsi di interrogare i contenitori di ingredienti si potrebbe pensare ad un preparatore.
L'utente quindi dovrebbe chiedere al preparatore la preparazione di una ricetta, mediante un apposito display, e questo dovrebbe comportare l'avvio della funzionalità "*richiesta preparazione ricetta*".
Affinché quanto detto sia possibile sarebbe necessario che fosse il preparatore a memorizzare le ricette e sarebbe altresì necessario che fosse il preparatore a consentire l'inserimento e l'eliminazione delle ricette.

Inoltre, sarebbe necessario che fosse il preparatore ad occuparsi della creazione di una lista della spesa sulla base degli ingredienti mancanti per la preparazione della ricetta.

L'aggiunta del preparatore comporterebbe una divisione delle responsabilità più "equa", andando a togliere alcuni degli incarichi assegnati al forno ed al frigorifero.
Infatti, nella versione attuale di Home Manager, il forno è sovraccarico di responsabilità e rappresenta l'autorità nell'interazione forno/frigorifero: è il forno che stabilisce quando cuocere una ricetta, è il forno che stabilisce quando il frigorifero deve controllare la presenza degli ingredienti, è il forno che stabilisce quando il frigorifero deve rimuovere gli ingredienti ed è il forno che funge da intermediario nella comunicazione utente/frigorifero.
Al frigorifero, nella versione attuale di Home Manager, è assegnata la responsabilità di redigere la lista della spesa a seguito di una segnalazione da parte del forno, che a sua volta ha interagito con l'utente. E' evidente non solo che la responsabilità assegnata non gli appartiene, ma anche che il protocollo di comunicazione è molto complesso e non chiuso alle modifiche.

3.2 Il gestore dei consumi

Un altro possibile scenario in ambiente Home Manager potrebbe riguardare l'intera abitazione, dal punto di vista del controllo del consumo energetico, con lo scopo di raggiungere un livello maggiore di risparmio energetico. Si potrebbe quindi pensare ad un'entità che si occupi di gestire i consumi dell'abitazione: il gestore dei consumi.

3.2.1 Le funzionalità del gestore dei consumi

Il gestore dei consumi dovrebbe essere sempre attivo nel sistema ed in ascolto delle richieste da parte degli altri dispositivi.
In particolare, il gestore dei consumi dovrebbe occuparsi di mantenere le informazioni sui dispositivi, ossia di sapere quali dispositivi fisici e simulati sono presenti nel sistema, lo stato in cui si trovano e le loro caratteristiche.
Inoltre, il gestore dei consumi dovrebbe essere in attesa di richieste di spegnimento o accensione da parte di un dispositivo, in modo tale da poter aggiornare le informazioni di stato di cui è a conoscenza.

Volendo aggiungere una maggiore intelligenza al gestore di consumi, quest'ultimo potrebbe occuparsi, all'arrivo di ogni richiesta di accensione, di controllare se il consumo del dispositivo che vuole accendersi comporterebbe lo scatto del salvavita; in tal caso dovrebbe impedire

l'accensione del dispositivo, altrimenti dovrebbe consentire al dispositivo di procedere con l'accensione e dovrebbe provvedere ad aggiornare le informazioni di stato del dispositivo in questione.
Nel caso in cui il dispositivo voglia spegnersi, invece, il gestore dei consumi non dovrebbe occuparsi di nessun controllo, ma dovrebbe semplicemente provvedere ad aggiornare lo stato del dispositivo in questione.

3.2.2 La gestione dei consumi negli scenari applicativi

Aggiungendo un gestore dei consumi con le caratteristiche sopra descritte all'abitazione, sarebbe necessario aggiungere delle operazioni automatiche alle funzionalità descritte nel paragrafo precedente.
In particolare, quando viene attivata la funzionalità "*richiesta preparazione ricetta*", il preparatore dovrebbe entrare in azione e dovrebbe in primo luogo verificare la possibilità di accendersi; dovrebbe quindi inviare una richiesta al gestore dei consumi ed attendere una sua risposta, prima di procedere con le operazioni descritte nel paragrafo precedente. Terminate le operazione da svolgere, il preparatore dovrebbe poi spegnersi e comunicare il suo cambiamento di stato al gestore dei consumi.
Allo stesso modo, quando viene attivata la funzionalità "*preparazione ricetta*", il preparatore ed il forno dovrebbero richiedere l'accensione al gestore dei consumi prima di poter procedere. Inoltre, terminate le funzioni da svolgere, sia il preparatore che il forno dovrebbero comunicare il loro spegnimento al gestore dei consumi, in modo tale che aggiorni le sue informazioni.

Infine, frigorifero, dispensa e tutti gli altri possibili "contenitori di ingredienti" dovrebbero essere accesi per poter essere presi in considerazione nelle funzionalità "*richiesta preparazione ricetta*" e "*preparazione ricetta*". Pertanto, la prima operazione che i "contenitori di ingredienti" dovrebbero svolgere, dopo la registrazione al sistema, dovrebbe essere proprio quella di richiedere l'accensione al gestore dei consumi.
Lo spegnimento di frigorifero, dispensa e altri contenitori di ingredienti dovrebbe avvenire soltanto in casi di malfunzionamenti; quando questo accade, il "contenitore di ingredienti" in questione non dovrebbe più essere considerato fino a che non si riaccende.

3.2.3 Gli ordini di spegnimento

Volendo aggiungere ancora più intelligenza, si potrebbe pensare ad un gestore di consumi che sia in grado, a seguito di una richiesta di

accensione che comporterebbe lo scatto del salvavita, di calcolare delle soluzioni alternative prima di impedire al dispositivo di accendersi.
Il gestore dei consumi, dovrebbe, in altre parole, analizzare se è possibile spegnere altri dispositivi in azione e consentire al dispositivo che ha effettuato la richiesta di accensione di accendersi.
In questo caso non si parlerebbe più di richieste di spegnimento, ma si parlerebbe di ordini di spegnimento, che il gestore dei consumi dovrebbe inviare ad uno o più dispositivi attivi nel sistema.
La politica di spegnimento dei dispositivi adottata dal gestore dei consumi dovrebbe essere stabilita dall'utente in base alle proprie esigenze. Per esempio, l'utente potrebbe stabilire che il gestore dei consumi spenga la lavatrice quando il forno vuole accendersi ma il consumo complessivo supera quello di soglia dell'abitazione.
Le politiche di spegnimento potrebbero essere di vario tipo e potrebbero essere differenziate in base alla stagione dell'anno.

3.2.4 Il gestore dei consumi ed il risparmio energetico

Il gestore dei consumi potrebbe essere in grado, al momento della richiesta di accensione da parte di un dispositivo, di calcolare se esistono momenti della giornata in cui l'accensione dello stesso dispositivo comporterebbe un consumo energetico inferiore. In tal caso, prima di procede all'accensione, dovrebbe essere mostrata nell'interfaccia del sistema, un messaggio che informi l'utente della possibilità di rimandare l'operazione in un altro momento.
Per esempio, a seguito di una richiesta di accensione della lavatrice, il gestore dei consumi potrebbe valutare l'esistenza di fasce orarie più agevoli, che comportano l'accensione della lavatrice con un costo dell'energia minore, e proporle all'utente. Quest'ultimo dovrebbe decidere se rimandare l'accensione nelle fasce orarie consigliate, o se procedere all'accensione.
Affinché il gestore dei consumi possa effettuare i suoi calcoli, dovrebbe essere a conoscenza di tutte le caratteristiche dei contratti di energia elettrica, gas, ecc…
La capacità del gestore dei consumi di stabilire fasce orarie più convenienti per l'accensione degli elettrodomestici potrebbe comportare un rilevante risparmio energetico complessivo.

Capitolo 4 – Il gestore dei consumi e l'allineamento dei dispositivi: analisi e progettazione

4.1 Obiettivi

Una volta analizzati i possibili scenari applicativi sviluppabili in Home Manager, rimane da raggiungere l'obiettivo di progettare ed implementare alcune funzionalità degli scenari, quelle di maggior rilevanza.

Ai fini della tesi, si è scelto di progettare ed implementare il gestore dei consumi, realizzando così le operazioni di accensione e spegnimento comuni in tutte le funzionalità dello nuovo scenario cucina indicato al paragrafo precedente. La progettazione e l'implementazione della cucina, invece, verrà progettata ed implementata in altre tesi.

4.2 Analisi del problema

Affinché il gestore dei consumi sia ben progettato, occorre far in modo che si integri con il sistema esistente.
Nell'attuale versione del sistema, come già analizzato nel capitolo 2, esiste un agente che si occupa dell'auto-detect, ossia di registrare i dispositivi fisici al sistema e di assegnare a ciascuno di essi un dispositivo simulato corrispondente.
Un aspetto di particolar importanza ai fini della realizzazione di un corretto gestore dei consumi, ed anche quello più delicato, sarà quello di mantenere l'allineamento dello stato dei dispositivi simulati con i corrispondenti fisici.
In sostanza, sarà necessario che la richiesta di accensione di un dispositivo fisico, nel caso in cui venga accettata, sia propagata anche al dispositivo simulato corrispondente, che dovrà accendersi, e viceversa. Lo stesso vale per lo spegnimento di uno dei due dispositivi.

4.3 Scelte progettuali

4.3.1 Chi si occupa del controllo dei consumi

Home Manager si basa su una metodologia di progettazione multi-agente pertanto il gestore di consumi sarà realizzato come agente indipendente, che si aggiunge al sistema. Il nome che gli sarà assegnato è UsageManagerAgent e si occuperà, per il momento, delle gestione delle richieste di accensione e spegnimento, nonché della gestione delle problematiche di allineamento.

4.3.2 Dove avviene la comunicazione

Home Manager si appoggia sull'infrastruttura TuCSoN. Pertanto, lo UsageManagerAgent potrà comunicare con i dispositivi del sistema utilizzando un centro di tuple apposito, chiamato `usage_manager_tc`.

4.3.3 Le conseguenze per gli agenti del sistema

Affinché ciascun agente possa comunicare con lo UsageManagerAgent è necessario che ciascuno di essi sia a conoscenza del centro di tuple `usage_manager_tc`.
Ciascun agente rappresentante un dispositivo del sistema dovrà utilizzare il centro di tuple `usage_manager_tc` per inviare allo UsageManagerAgent le richieste di accensione o spegnimento; inoltre, dovrà essere in ascolto sul centro di tuple `usage_manager_tc` di richieste di allineamento dello stato, che rappresenteranno degli ordini da parte dello UsageManagerAgent di accensione o spegnimento.

4.4 Progettazione

4.4.1 Come avviene il controllo dei consumi: il protocollo di comunicazione

Una volta analizzate le entità in gioco, è necessario stabilire un protocollo di comunicazione.

In primo luogo è necessario che ciascun dispositivo, al momento della connessione al sistema, informi lo UsageManagerAgent della sua presenza. Dovrà pertanto scrivere una tupla nel centro di tuple `usage_manager_tc` del tipo:

```
presence_information(name_device,info_device(state, watt),type)
```

Lo stato può essere "on" oppure "off", mentre il type può essere "s" oppure "p" a seconda che si tratti di un dispositivo simulato o fisico.

I dispositivi simulati scrivono la propria `presence_information` al momento dell'attivazione e, una volta attivati, non possono mai disconnettersi dal sistema. Il loro consumo in watt è lo stesso del corrispondente dispositivo fisico.
I dispositivi fisici scrivono la loro `presence_information` quando si connettono al sistema e la rimuovo quando si disconnettono dal sistema.

Infine, per quanto riguarda i dispositivi simulati presenti al momento dell'avvio del sistema (a cui non corrispondo fisici e di cui si è parlato nel capitolo 2), si suppone che la loro `presence_information` sia già presente nel centro di tuple, che non possa essere rimossa e che il consumo in watt di ciascuno di essi sia nullo.

Anche i dispositivi fisici (e simulati corrispondenti) che non consumano corrente devono segnalare allo UsageManagerAgent la loro presenza mediante una tupla `presence_information`, indicando però un valore pari a zero nel sottocampo watt.

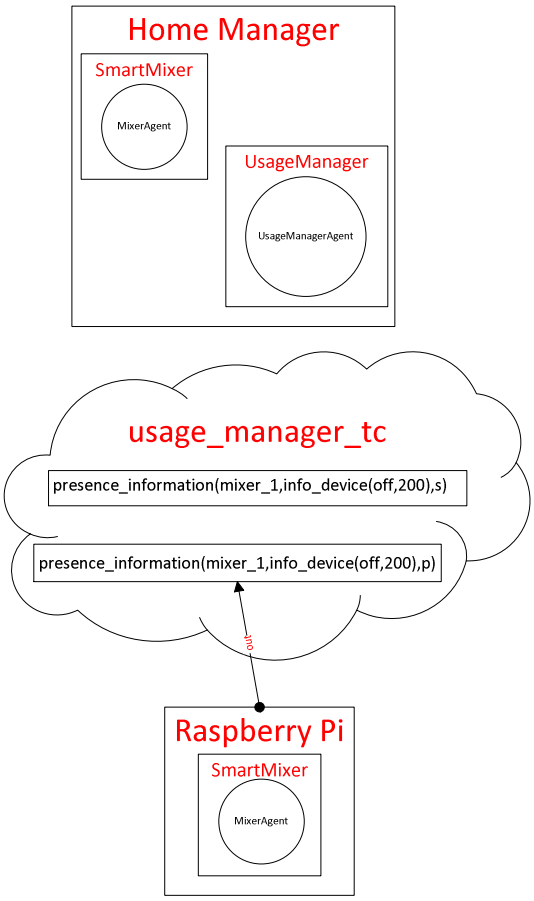


Figura 5: segnalazione di presenza da parte di un dispositivo fisico

Quando un dispositivo simulato viene creato, viene scritta la sua `presence_information` nel centro di tuple ed il suo stato è necessariamente "off".
Lo stesso vale per lo stato di un dispositivo fisico nel momento in cui si connette al sistema e scrive la `presence_information`. In quest'ultimo caso, però, lo UsageManagerAgent dovrà occuparsi di controllare lo stato del corrispondente simulato che, a seguito della creazione, potrebbe essersi acceso e quindi avere stato "on". In tal caso,

lo UsageManagerAgent dovrà modificare lo stato del dispositivo fisico appena connesso in “on”, in modo tale che sia allineato. Dovrà, inoltre, ordinare al dispositivo fisico di accendersi mediante una tupla il cui formato verrà riportato in seguito (vedi “ordine di accensione”).

Quando un dispositivo vuole accendersi, deve scrivere nel centro di tuple `usage_manager_tc` una tupla del tipo

state_change(“request”, name_device,“on”, type)

Lo UsageManagerAgent, riceve la richiesta in questione e, in base alle `presence_information` dei dispositivi con stato on, controlla se il consumo attuale sommato a quello del dispositivo che vuole accendersi è inferiore o maggiore a quello di soglia dell’abitazione (in questa operazione dovrà fare attenzione a non considerare il consumo di un dispositivo fisico se ha già considerato quello del simulato corrispondente e viceversa). Una volta effettuato il controllo, lo UsageManagerAgent deve inviare una risposta al dispositivo che ha effettuato la richiesta; scrive quindi nel centro di tuple `usage_manager_tc` una tupla del tipo

state_change(“response”,name_device,state,type)

Lo stato indicato sarà “on” nel caso in cui il dispositivo può accendersi, “off” altrimenti.
A questo punto, solo nel caso di risposta affermativa, lo UsageManagerAgent deve verificare se esiste nel centro di tuple una tupla `presence_information` di un dispositivo con lo stesso `name_device` di quello che è appena stato acceso, ma con tipo differente. Nel caso in cui esista, deve inviare al dispositivo in questione un ordine di accensione affinché possa allinearsi; scrive quindi nel centro di tuple `usage_manager_tc` una tupla del tipo:

state_change(“order”,name_device,“on”,type)

Infine, solo nel caso in cui la risposta all’accensione sia affermativa, deve modificare lo stato nella `presence_information` in “on”. Il cambiamento dello stato avverrà in una o in due tuple, a seconda che siano presenti entrambi (dispositivo fisico e simulato corrispondente) o uno solo dei dispositivi.

Anche i dispositivi che non consumano corrente devono inviare allo UsageManagerAgent delle richieste di accensione. In questo modo viene mantenuta uniformità nel protocollo.
Tali richieste di accensione verranno, ovviamente, sempre accettate senza la necessità di effettuare alcun controllo.

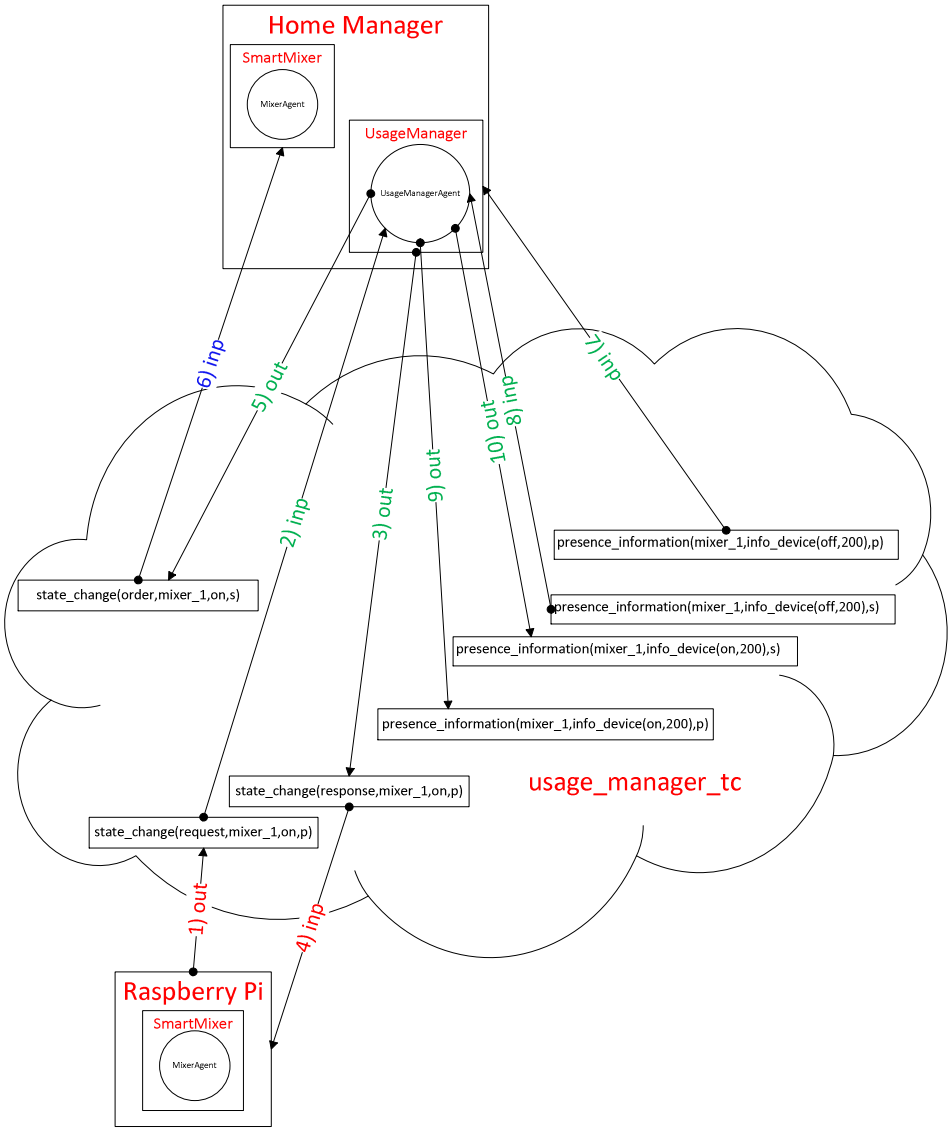


Figura 6: richiesta di accensione da parte di un dispositivo fisico

Allo stesso modo, quando un dispositivo vuole spegnersi, invia una richiesta allo UsageManagerAgent scrivendo nel centro di tuple `usage_manager_tc` una tupla del tipo

```
state_change("request",name_device,"off",type)
```

In questo caso lo UsageManagerAgent non deve effettuare alcun controllo e non deve inviare alcuna tupla di risposta al dispositivo in questione.
Nel caso in cui esista una tupla `presence_information` di un dispositivo con lo stesso `name_device` di quello che è appena stato spento, ma con tipo differente, deve provvedere all'allineamento del suo stato, scrivendo nel centro di tuple `usage_manager_tc` una tupla del tipo:

```
state_change("order",name_device,"off",type)
```

Infine, lo UsageManagerAgent deve modificare lo stato nella `presence_information` in "off". Il cambiamento dello stato avverrà, anche in questo caso, in una o in due tuple, a seconda che siano presenti entrambi (dispositivo fisico e simulato corrispondente) o uno solo dei dispositivi.

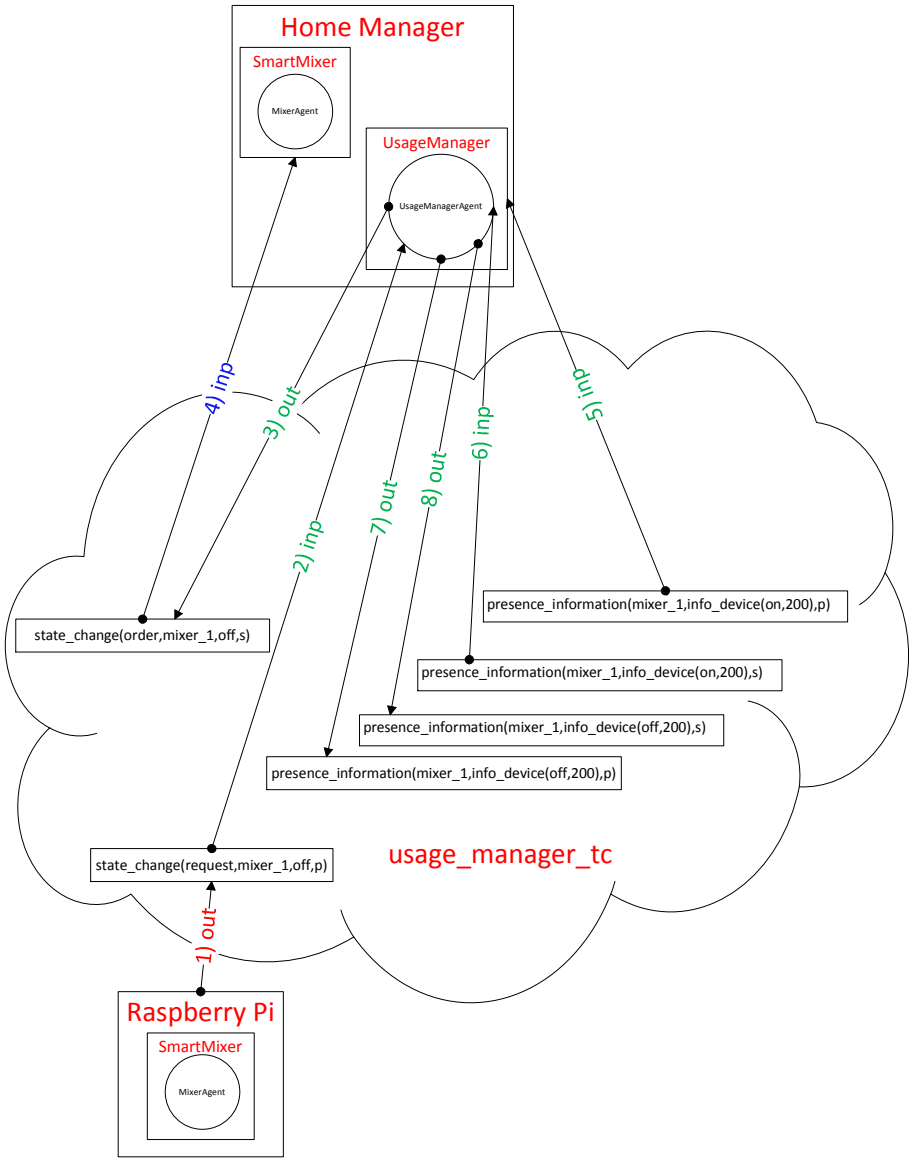


Figura 7: richiesta di spegnimento da parte di un dispositivo fisico

L'entità UsageManagerAgent è quindi sempre in ascolto nel centro di tuple `usage_manager_tc`, in attesa di leggere una tupla `presence_information` o una tupla `state_change`. Non appena ne trova una, effettua le operazioni sopra descritte.

Capitolo 5 – Il gestore dei consumi e l'allineamento dei dispositivi: implementazione e collaudo

5.1 Scelte implementative

Una volta effettuate tutte le scelte progettuali e stabilito un protocollo di comunicazione, è possibile procedere con l'implementazione del gestore dei consumi.

Per prima cosa si è scelto di creare un nuovo package nel progetto Home Manager, che dovrà contenere tutte le future classi per il controllo dei consumi. Il motivo per cui si è scelto di creare un package dedicato alla gestione dei consumi è quello di mantenere una buona organizzazione delle classi.

Il nome scelto per il package è *it.unibo.homemanager.usages*.

Il package conterrà lo UsageManagerAgent, che è una classe java.

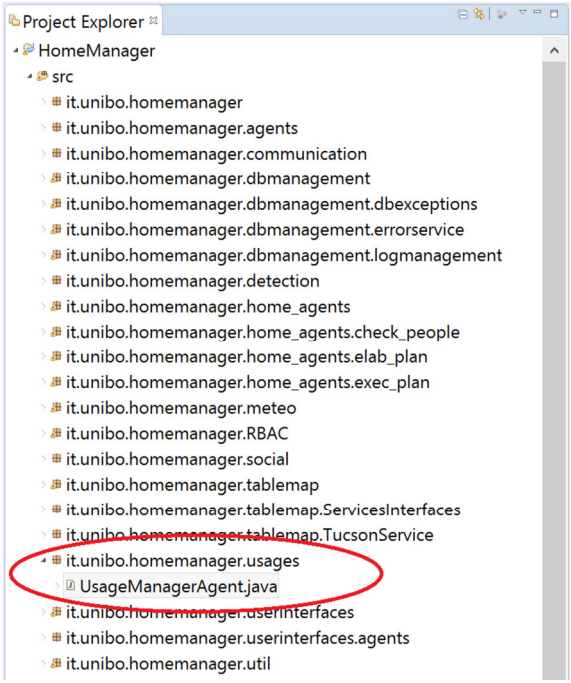


Figura 8: package e agente per il controllo dei consumi

La classe UsageManagerAgent.java sarà realizzata come singleton in modo tale che nel sistema sia presente sempre una sola istanza della stessa.
Affinché la classe UsageManagerAgent.java sia un singleton è necessario che il costruttore sia privato ed inoltre deve esporre un metodo statico `getInstance()` che sia un punto di accesso globale all'istanza.

Inoltre, lo UsageManagerAgent potrà essere considerato come un demone: le sue funzionalità sono sempre attive nel sistema e messe a disposizione dell'utente, ma quest'ultimo non può controllarle.

Per le caratteristiche appena descritte, lo UsageManagerAgent sarà attivato mediante il metodo *go()* nella classe Main.java, ossia all'avvio del sistema, e si disattiverà soltanto allo spegnimento del sistema:

```
// init usageManager
UsageManagerAgent usageManagerAgent = UsageManagerAgent.getInstance();
usageManagerAgent.go();
```

Prerequisito per l'attivazione dello UsageManagerAgent è la creazione del centro di tuple *usage_manager_tc* e la sua aggiunta alla lista di centri di tuple del sistema.

```
TucsonTupleCentreId usageManager_tc = new TucsonTupleCentreId(
        "usage_manager_tc", "localhost", "20504");

            tupleCenters.add(19, usageManager_tc);
```

Le seguenti linee di codice sono inserite nella classe Main.java del progetto, nella quale vengono definiti tutti i centri di tuple del sistema, prima dell'attivazione dello UsageManagerAgent.

Per quanto riguarda la realizzazione del protocollo di comunicazione, si utilizzerà la classe astratta AbstractCommunicationLanguage, citata al capitolo2, nella quale verranno inserite le costanti per la comunicazione ed i rispettivi metodi getters:

```
private static final String PRESENCE_INFORMATION = "presence_information";
private static final String STATE_CHANGE = "state_change";
private static final String REQUEST = "request";
private static final String RESPONSE = "response";
private static final String ORDER = "order";


            public static String getPresenceInformation() {
                return (PRESENCE_INFORMATION);
            }
            public static String getStateChange() {
                return (STATE_CHANGE);
            }
            public static String getRequest() {
                return (REQUEST);
            }
            public static String getResponse() {
                return (RESPONSE);
            }
            public static String getOrder() {
                return (ORDER);
            }
```

5.2 UsageManagerAgent

Prima di procedere alla descrizione di come le funzionalità dello UsageManagerAgent verranno realizzate è necessario porre l'attenzione sul metodo *main()* dello UsageManagerAgent, invocato al momento dell'attivazione dell'agente stesso.

```
@Override
protected void main() {
    setAcc(getContext());
    try {
        init();
        while ( true ) {
                List<LogicTuple> presenceInformationRequest = presenceInformationRequest();
                if ( presenceInformationRequest.size() != 0 )
                    managePresenceInformationRequest(presenceInformationRequest);

                LogicTuple stateChangeRequest = stateChangeRequest();
                if ( stateChangeRequest != null )
                    manageStateChangeRequest(stateChangeRequest);

                Thread.sleep(1000);
            }
    }catch (Exception e) { }
}
```

Il gestore dei consumi, come descritto nel capitolo 4, deve essere sempre in attesa di tuple del tipo `presence_information` o di tuple del tipo `state_change`.

E' proprio questa attesa infinita (*while(true)*) che lo caratterizza come demone. Infatti, il metodo *main()* viene invocato automaticamente al momento dell'attivazione del sistema e una volta invocato provoca proprio il comportamento desiderato.

Prima dell'attesa viene invocato il metodo *init()*. In tale metodo lo UsageManagerAgent accede al file di configurazione, *usages.txt*, legge l'informazione sul massimo consumo previsto per l'abitazione e la scrive nel centro di tuple `usage_manager_tc` sotto forma di tupla. Questa informazione è fondamentale affinché lo UsageManagerAgent possa svolgere le sue funzioni.

5.2.1 Gestione delle presence_information ed allineamento

Lo UsageManagerAgent, come si può notare nel metodo *main()* riportato al paragrafo precedente, invoca ciclicamente il metodo

presenceInformationRequest():List<LogicTuple>

Quest'ultimo si occupa di restituire, se presenti, tutte le nuove tuple di tipo `presence_information`, ossia tutte quelle tuple `presence_information` che non sono ancora state gestite dallo stesso UsageManagerAgent.

Se viene trovata una o più tupla `presence_information` non gestita, viene invocato il metodo

managePresenceInformationRequest(List<LogicTuple):void

.

```
private void managePresenceInformationRequest(List<LogicTuple> newPresences) throws Exception {
    int first = 0;
    int third = 2;

    for (LogicTuple presenceInformation : newPresences) {
        String deviceName = presenceInformation.getArg(first).toString();
        String deviceType = presenceInformation.getArg(third).toString();
        if ( deviceType.equals("s") )
            getPresences().put(deviceName + "-" + deviceType, presenceInformation.toString());
    }
    for (LogicTuple presenceInformation : newPresences) {
        String deviceName = presenceInformation.getArg(first).toString();
        String deviceType = presenceInformation.getArg(third).toString();
        if (deviceType.equals("p")) {
            LogicTuple association = getAssociation(deviceName);
            alignState(presenceInformation, association);
            getPresences().put(deviceName + "-" + deviceType, presenceInformation.toString());
        }
    }
}
```

Questo metodo, per prima cosa, si occupa di identificare tutte le tuple di presenza che riguardano dispositivi simulati, in modo che lo UsageManagerAgent possa memorizzarle al suo interno e così che non vengano più considerate da gestire al controllo successivo.

Infine, si occupa di gestire le tuple di presenza riguardanti i dispositivi fisici. Queste richiedono un maggior controllo: come descritto nel capitolo 4, le tuple `presence_information` dei dispositivi fisici vengono aggiunte con uno stato del dispositivo "off", ma il dispositivo simulato corrispondente potrebbe non essere spento. Per prima cosa, quindi, occorre ritrovare tra le tuple `presence_information` memorizzate dallo UsageManagerAgent, quella riguardante il dispositivo corrispondente a quello in questione, ma di tipo simulato. Questa prima operazione viene effettuata mediante il metodo

getAssociation(String):logicTuple

che appunto restituisce la tupla del dispositivo simulato corrispondente al fisico in questione, dato il `name_device`.
Una volta trovata la tupla `presence_information` del dispositivo simulato corrispondente al fisico in questione, viene invocato il metodo

alignState(LogicTuple, LogicTuple):void

Questo metodo si occupa, data la tupla `presence_information` di un dispositivo fisico e quella del dispositivo simulato ad esso corrispondente, di allineare lo stato del dispositivo fisico con quello del dispositivo simulato e di inviare un ordine di accensione/spegnimento al dispositivo fisico, mediante una tupla di tipo `state_change`, nel caso in cui il suo stato sia cambiato, in modo tale che quest'ultimo possa essere a conoscenza dello stato che deve assumere.

5.2.2 Gestione degli state_change ed allineamento

Lo UsageManagerAgent si occupa, inoltre, di invocare ciclicamente il metodo

stateChangeRequest():LogicTuple

Questo metodo restituisce, se presente, la richiesta di cambio stato da parte di un dispositivo del sistema.

Se la richiesta di cambio stato viene rilevata, viene invocato il metodo

manageStateChangeRequest()

```
private void manageStateChangeRequest(LogicTuple stateChangeRequest) throws Exception {
    int second = 1;
    int third = 2;
    int fourth = 3;

    String deviceName = stateChangeRequest.getArg(second).toString();
    String requestType = stateChangeRequest.getArg(third).toString();
    String deviceType = stateChangeRequest.getArg(fourth).toString();

    if ( requestType.equals("on") )
        turnOn(deviceName, deviceType);
    else if ( requestType.equals("off") )
        turnOff(deviceName, deviceType);
}
```

Questo metodo si occupa di distinguere le richieste di accensione dalle richieste di spegnimento, delegando la loro gestione rispettivamente ai seguenti metodi

turnOn(String, String):void

turnOff(String, String):void

5.2.2.1 Richieste di accensione

Le richieste di accensione, come appena indicato, comportano l'invocazione del metodo

turnOn(String, String):void

Questo metodo si occupa, per prima cosa, di leggere dal centro di tuple la `presence_information` del dispositivo che ha appena richiesto l'accensione.
La tupla `presence_information` è necessaria affinché sia possibile ottenere il consumo in watt del dispositivo.

Una volta ottenuto il consumo in watt devono essere effettuate le operazioni di controllo, necessarie per stabilire se l'accensione del dispositivo comporta o meno il superamento del consumo massimo dell'abitazione.

Quindi, prima di poter procedere con la scrittura di una risposta al dispositivo che ha effettuato la richiesta di accensione è necessario invocare il seguente metodo

canTurnOn(String, int):boolean

```
private boolean canTurnOn(String deviceName, int watt) throws Exception {
    int first = 0;
    int second = 1;

    String nameTemplate = "max";
    String tuple = nameTemplate + "(" + "A" + ")";
    LogicTuple template = LogicTuple.parse(tuple);
    ITucsonOperation operation = getAcc().rdp(getUsageManagerTc(),
            template, Long.MAX_VALUE);

    LogicTuple maxTuple = operation.getLogicTupleResult();
    int max = maxTuple.getArg(first).intValue();

    nameTemplate = AgentCommunicationLanguage.getPresenceInformation();
    tuple = nameTemplate + "(A,B,C)";
    template = LogicTuple.parse(tuple);
    operation = getAcc().rdAll(getUsageManagerTc(), template, Long.MAX_VALUE);

    List<LogicTuple> allPresences = operation.getLogicTupleListResult();
    List<String> devices = new ArrayList<String>();

    int currentWatt = 0;
    for (LogicTuple presence : allPresences) {
        String currentDevice = presence.getArg(first).toString();
        if ( !devices.contains(currentDevice) && !currentDevice.equals(deviceName) ) {
            String deviceState = presence.getArg(second).getArg(first).toString();
            int currentDeviceWatt = presence.getArg(second).getArg(second).intValue();

            if ( deviceState.equals("on") )
                currentWatt = currentWatt + currentDeviceWatt;

            devices.add(currentDevice);
        }
    }
    return ( (currentWatt + watt) < max );
}
```

Questo metodo si occupa, per prima cosa, di leggere dal centro di tuple `usage_manager_tc` il consumo massimo possibile nell'abitazione affinché non scatti il salvavita.

Una volta ottenuto tale valore, è necessario calcolare il consumo attuale dell'abitazione. E' necessario quindi leggere tutte le tuple `presence_information` dal centro di tuple `usage_manager_tc` ed effettuare la somma del consumo in watt di tutti i dispositivi che risultano accesi, cioè che hanno stato "on". Occorre prestare attenzione a non sommare due volte il consumo in watt di un dispositivo (se si è considerato il consumo del dispositivo fisico acceso, non bisogna considerare il consumo del corrispondente dispositivo simulato acceso).

A questo punto è possibile verificare se il consumo attuale dell'abitazione sommato a quello del dispositivo che vuole accendersi supera o meno il consumo massimo possibile nell'abitazione, ed è quindi possibile restituire un risultato.

Il risultato sarà "true" nel caso sia possibile accendere il dispositivo, "false" altrimenti.

Una volta ottenuto il risultato dall'operazione di controllo è possibile procedere.
Nel caso in cui il risultato è negativo, è necessario scrivere una tupla `state_change` di tipo "response" con stato "off", che indichi al dispositivo che ha effettuato la richiesta di accensione che non può accendersi.

Nel caso in cui, invece, il risultato è positivo, è necessario informare il dispositivo che può procedere con l'accensione, è necessario inviare un ordine di accensione al dispositivo corrispondente a quello in questione (se presente) ed è, inoltre, necessario modificare la `presence_information` in modo tale che lo stato del dispositivo (ed anche quello del dispositivo corrispondente, se presente) risulti "on".

5.2.2.2 Richieste di spegnimento

Le richieste di spegnimento, invece, comportano l'invocazione del metodo

turnOff(String, String):void

Questo metodo si occupa di aggiornare lo stato nella `presence_information` del dispositivo che richiede lo spegnimento con quello corretto.
Inoltre, aggiorna lo stato anche nella `presence_information` del dispositivo corrispondente a quello che ha richiesto lo spegnimento

(se connesso al sistema) ed invia a tale dispositivo un ordine di spegnimento affinché si possa allineare.

5.3 Un esempio di dispositivo simulato: MixerAgent

Per poter procedere con il collaudo delle funzionalità implementate al paragrafo precedente è necessario realizzare un dispositivo che sia in grado di segnalare al gestore dei consumi la sua presenza, che sia in grado di inviare richieste di accensione e spegnimento e che sia in grado di allinearsi.

La scelta del dispositivo da realizzare è "casuale". Infatti le funzionalità che verranno implementate dovranno essere comuni a tutti i dispositivi del sistema affinché possano accendersi, spegnersi ed allinearsi.

In questa tesi verranno implementate le funzionalità appena elencate in un agente, MixerAgent, che rappresenta il preparatore della cucina citato nel capitolo 3.

La classe java su cui si andrà ad agire è la classe MixerAgent.java, già esistente nel sistema all'interno del package

it.unibo.homemanager.agents

La classe è già stata realizzata in fase di implementazione dell'auto-detect e, come tutti gli altri agenti collegati a dei dispositivi del sistema, prende in ingresso nel costruttore il dispositivo da monitorare, i centri di tuple necessari alla comunicazione ed i pannelli necessari all'aggiornamento della grafica.

Prima di procedere, è necessario modificare quanto è già stato realizzato andando ad aggiungere nel costruttore della classe il centro di tuple per la comunicazione con il gestore dei consumi: `usage_manager_tc`.

A questo punto è possibile proseguire con l'implementazione.

5.3.1 Scrittura della presence_information

Ciascun agente associato ad un dispositivo simulato del sistema, nel caso specifico il MixerAgent, deve come prima cosa in seguito all'avvio, segnalare la propria presenza al gestore dei consumi.
Questa operazione verrà implementata mediante l'invocazione, all'interno del *`main()`* dell'agente, del metodo

`writePresenceInformation():boolean`

Questo metodo comporta la scrittura della tupla `presence_information` nel centro di tuple `usage_manager_tc`.

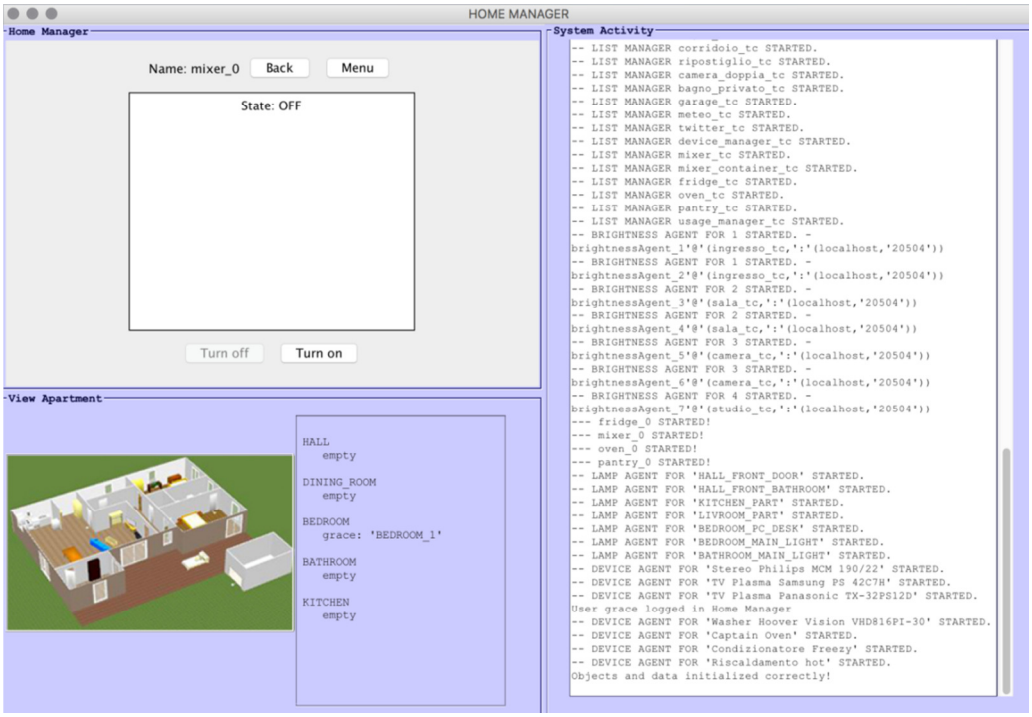
5.3.2 Invio di state_change

Il dispositivo simulato, dopo essere stato avviato ed aver scritto la propria `presence_information` nel centro di tuple, è pronto ad interagire con l'utente. L'interazione può avvenire grazie alla presenza dell'auto-detect, che provvede ad aggiornare opportunamente la grafica mostrando, nella lista dei dispositivi, il dispositivo in questione.

Inoltre, l'interazione può avvenire grazie alla creazione di un pannello, MixerPanel, opportunamente creato.

Il MixerPanel verrà mostrato quando l'utente seleziona un dispositivo mixer dalla lista dei dispositivi presenti nel sistema e, successivamente, seleziona l'opzione "Properties".

Una volta mostrato, il MixerPanel consente di tornare indietro o di tornare al menù, mostra lo stato del dispositivo e consente l'interazione con il mixer selezionato.



Affinché sia possibile il collaudo delle funzionalità dello UsageManagerAgent, è necessario che il MixerPanel sia dotato di due pulsanti che, se premuti consentano rispettivamente l'accensione e lo spegnimento del dispositivo.

L'evento "click" sul pulsante "Turn on" comporta l'invocazione del metodo

turnOn():int

contenuto nella classe MixerAgent, che a sua volta scatena una richiesta di accensione.

```
private void onClickButtonTurnOn(ActionEvent event) {
    turnOn();
}

private void turnOn() {
    try{
        int result = getMixerAgent().turnOn();
        if(result == 0)
            JOptionPane.showMessageDialog(this,
                    "Request completed successfully!");
        else if(result == -3)
            JOptionPane.showMessageDialog(this,
                    "I can't turn on, max consumption reached!");
        else
            JOptionPane.showMessageDialog(this,
                    "Error!");
    }catch(Exception e){
        e.printStackTrace();
    }
}
```

L’evento “click” sul pulsante “Turn off” comporta l’invocazione del metodo

turnOff():int

contenuto nella classe MixerAgent, che a sua volta scatena una richiesta di spegnimento.

```
private void onClickButtonTurnOff(ActionEvent event) {
    turnOff();
}

private void turnOff() {
    try {
        int result = getMixerAgent().turnOff();
        if(result == 0)
            JOptionPane.showMessageDialog(this,
                    "Request completed successfully!");
        else
            JOptionPane.showMessageDialog(this,
                    "Error!");
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Inoltre, affinché l’utente non invii richieste di accensione e spegnimento in modo errato, il MixerPanel “si mette in ascolto” dell’evento *onStateChange()*, lanciato dallo stesso MixerAgent ogni qual volta il proprio stato interno cambia (quando passa da acceso a spento e viceversa)

```
getMixerAgent().addStateListener(new StateChangeListener() {
    @Override
    public void onStateChanged() {
        onStateChangedEvent();
    }
});
onStateChangedEvent();
```

e quando un cambio di stato viene rilevato, i bottoni della grafica si disabilitano secondo la seguente logica:

- quando lo stato risulta ON, il bottone “Turn on” è disabilitato;
- quando lo stato risulta OFF, il bottone “Turn off” è disabilitato.

```
private void onStateChangedEvent() {
    if ( getMixerAgent().getState() )
        getMixerState().setText("State: ON");
    else
        getMixerState().setText("State: OFF");

    getButtonTurnOff().setEnabled( getMixerAgent().getState() );
    getButtonTurnOn().setEnabled( !getMixerAgent().getState() );
}
```

Inoltre, viene mostrato nel pannello il nuovo stato del dispositivo.

5.3.2.1 Richieste di accensione

Come indicato nel paragrafo precedente, le richieste di accensione vengono gestite mediante l'invocazione del metodo

turnOn():int

Il metodo si occupa di scrivere una tupla del tipo `state_change` di richiesta con stato "on" nel centro di tuple `usage_manager_tc`.
Inoltre, si occupa di attendere la risposta da parte dello UsageManagerAgent e, solo nel caso in cui il risultato della risposta sia "on", procede con l'accensione del dispositivo.

L'intero restituito dal metodo riporta il risultato dell'operazione.

5.3.2.2 Richieste di spegnimento

Come precedentemente indicato, le richieste di spegnimento comportano l'invocazione del metodo

turnOff():int

Questo metodo si occupa di scrivere nel centro di tuple `usage_manager_tc` una tupla `state_change` di richiesta con stato "off".
In questo caso non si attende nessuna risposta da parte dello UsageManagerAgent e si procede direttamente con il cambio dello stato.

L'intero restituito rappresenta il risultato dell'operazione.

5.3.3 Allineamento

Per completare il MixerAgent è necessario aggiungere la funzionalità di allineamento.

Affinché questo sia possibile è necessario che il MixerAgent, dopo aver scritto la propria `presence_information`, sia in attesa di possibili ordini di accensione o spegnimento da parte dello UsageManagerAgent.

Il MixerAgent invoca ogni secondo il metodo

alignState()

all'interno del metodo `main()` dell'agente stesso:

```
while (true) {
    alignState();
    Thread.sleep(1000);
}
```

Questo metodo controlla la presenza di tuple `state_change` di tipo "order" e, nel caso in cui ne rilevi una, controlla se l'ordine è di accensione o di spegnimento e provvede ad aggiornare lo stato del dispositivo.

5.4 Un esempio di dispositivo fisico: SmartMixer

Terminata la realizzazione dell'agente MixerAgent, è necessario procedere con la creazione di un dispositivo fisico.

Il dispositivo fisico verrà soprattutto utilizzato per il collaudo delle funzionalità di allineamento.

Il dispositivo fisico che verrà creato sarà, sia per uniformità con la tesi sia per proseguire il lavoro effettuato in altre tesi, lo SmartMixer.

Infatti, per il collaudo delle funzionalità di auto-detect è stato realizzato un dispositivo SmartMixer dotato soltanto delle funzionalità di richiesta di un nome per identificarsi nel sistema.

Per poter aggiungere le funzionalità di nostro interesse allo SmartMixer esistente è necessario modificare il progetto già esistente su Raspberry Pi.

Per prima cosa è necessario creare nel `main()` del progetto il centro di tuple `usage_manager_tc` e modificare il costruttore dell'agente passando il centro di tuple come argomento.

E' inoltre necessario modificare il file Configuration.java nel package *model.configuration* aggiungendo il consumo effettivo del dispositivo fisico.

A questo punto è possibile procedere con l'aggiunta delle funzionalità all'interno della classe MixerAgent.java, presente nel package *model.agent*.
I metodi che contiene la seguente classe sono del tutto identici a quelli già implementati per il dispositivo simulato al paragrafo precedente.

Anche l'interfaccia grafica per l'interazione con l'utente sarà del tutto simile a quella già implementata per il dispositivo simulato.

5.4.1 Rimozione della presence_information

Il MixerAgent presente nel dispositivo fisico si differenzia da quello del dispositivo simulato per la presenza del metodo

removePresenceInformation():boolean

Il metodo viene invocato quando il dispositivo si disconnette dal sistema e comporta la rimozione della tupla `presence_information` dal centro di tuple `usage_manager_tc`.

Nel progetto in questione, la disconnessione di un dispositivo fisico dal sistema avviene quando viene chiuso il pannello grafico che rappresenta il dispositivo fisico stesso.

```
addWindowListener(new WindowListener() {
    public void windowOpened(WindowEvent event) { }

    public void windowIconified(WindowEvent event) { }

    public void windowDeiconified(WindowEvent event) { }

    public void windowDeactivated(WindowEvent event) { }

    public void windowClosing(WindowEvent event) { }

    public void windowClosed(WindowEvent event) {
        try {
            getMixerAgent().removePresenceInformation();
        } catch (Exception e) { }
    }

    public void windowActivated(WindowEvent event) { }
});
```

5.4.2 Aggiunta dei sensori di stato

Per mostrare lo stato del dispositivo fisico, oltre all'apposita area nell'interfaccia grafica dello stesso, si connettono al Raspberry Pi due led, uno giallo ed uno verde.

Il led verde sarà acceso quando il dispositivo è acceso, mentre il led giallo sarà acceso quando il dispositivo è spento.

Per poter inviare ai led i segnali di accensione e spegnimento è necessario aggiungere al progetto la libreria Pi4J descritta nel capitolo 2.

A questo punto è possibile procedere. Per motivi di separazione delle responsabilità viene creata una nuova classe, SensorsManager.java, e viene inserita in un package, *model.sensors*, creato ad hoc per contenere tutte le classi per la gestione dei sensori.

La classe è implementata come singleton ed avrà quindi le stesse caratteristiche dello UsageManagerAgent, descritte all'inizio del capitolo 4.

Nella classe sono definiti i due sensori e sono implementati i metodi di accensione e spegnimento degli stessi.

```
private SensorsManager() {
    GpioController controller = GpioFactory.getInstance();

    greenLed = controller.provisionDigitalOutputPin(
            RaspiPin.GPIO_00, "GreenLed", PinState.LOW);

    yellowLed = controller.provisionDigitalOutputPin(
            RaspiPin.GPIO_01, "YellowLed", PinState.LOW);

    getGreenLed().setShutdownOptions(true,
            PinState.LOW, PinPullResistance.OFF);

    getYellowLed().setShutdownOptions(true,
            PinState.LOW, PinPullResistance.OFF);
}
```

```
public void turnOnGreenLed() {
    getGreenLed().high();
}

public void turnOffGreenLed() {
    getGreenLed().low();
}

public void turnOnYellowLed() {
    getYellowLed().high();
}

public void turnOffYellowLed() {
    getYellowLed().low();
}
```

Questi metodi verranno invocati all'interno dei metodi `turnOn()`, `turnOff()` ed `alignState()` del MixerAgent associato al dispositivo fisico.
In particolare:

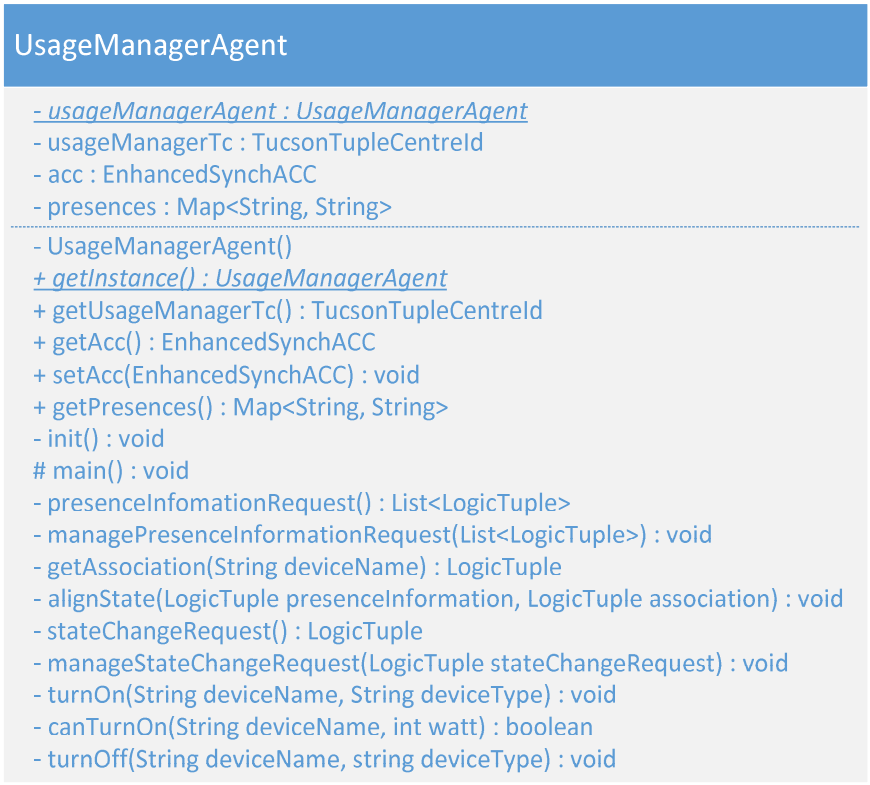
- quando viene invocato il metodo `turnOn()`, se il dispositivo riceve risposta affermativa di accensione da parte dello UsageManagerAgent, lo stato del dispositivo diventa acceso, il led verde viene acceso ed il led giallo viene spento;

- quando viene invocato il metodo `turnOff()`, lo stato del dispositivo diventa spento, il led verde viene spento ed il led giallo viene acceso;
- quando viene rilevato un ordine di accensione nel metodo `alignState()`, il led verde viene acceso ed il led giallo viene spento;
- quando viene rilevato un ordine di spegnimento nel metodo `alignState()`, il led verde viene spento ed il led giallo viene acceso.

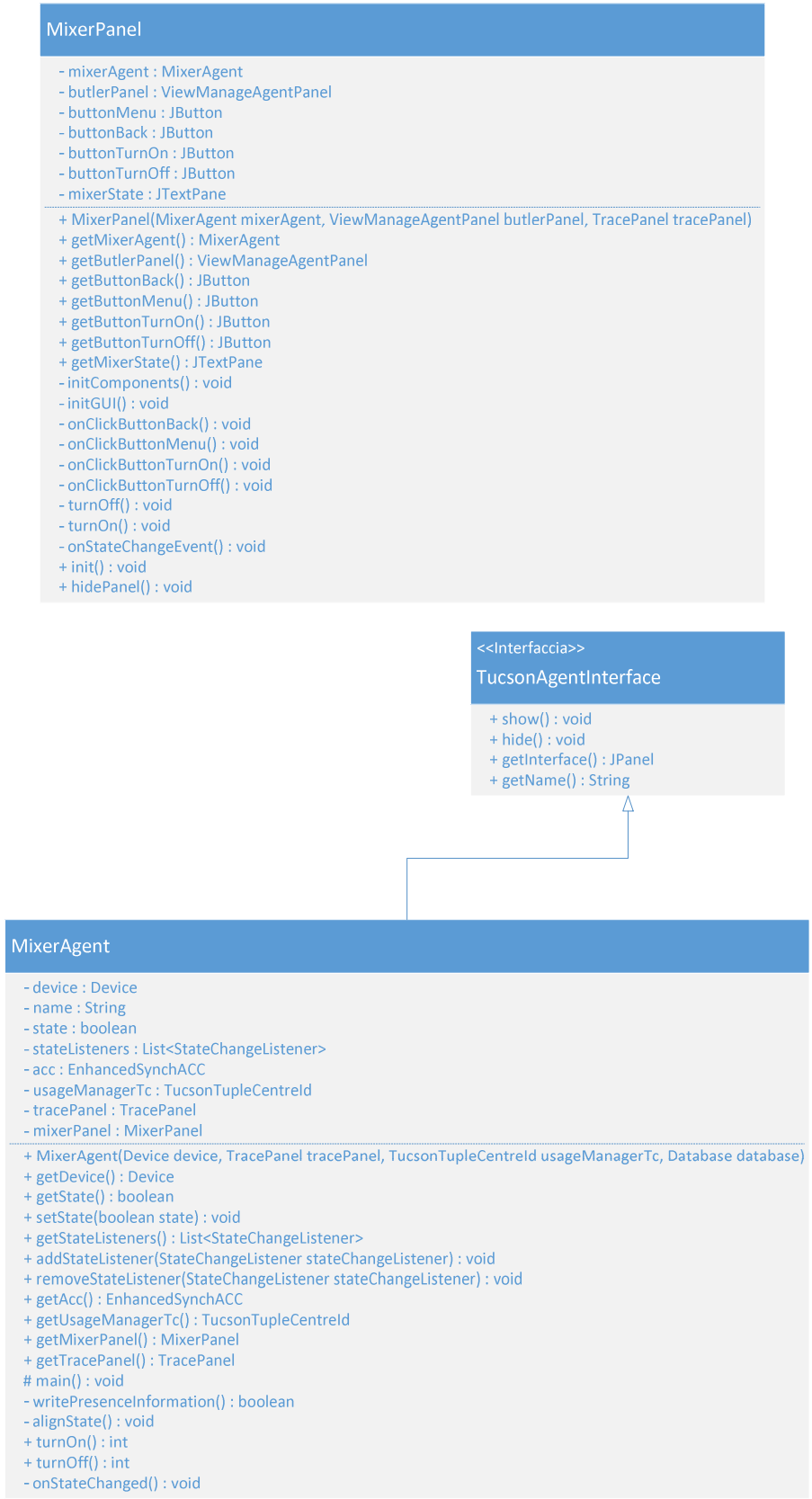
5.5 Diagrammi delle classi

Al fine di riepilogare le classi aggiunte al sistema ed i metodi che espongono, sono di seguito riportati i diagrammi UML.

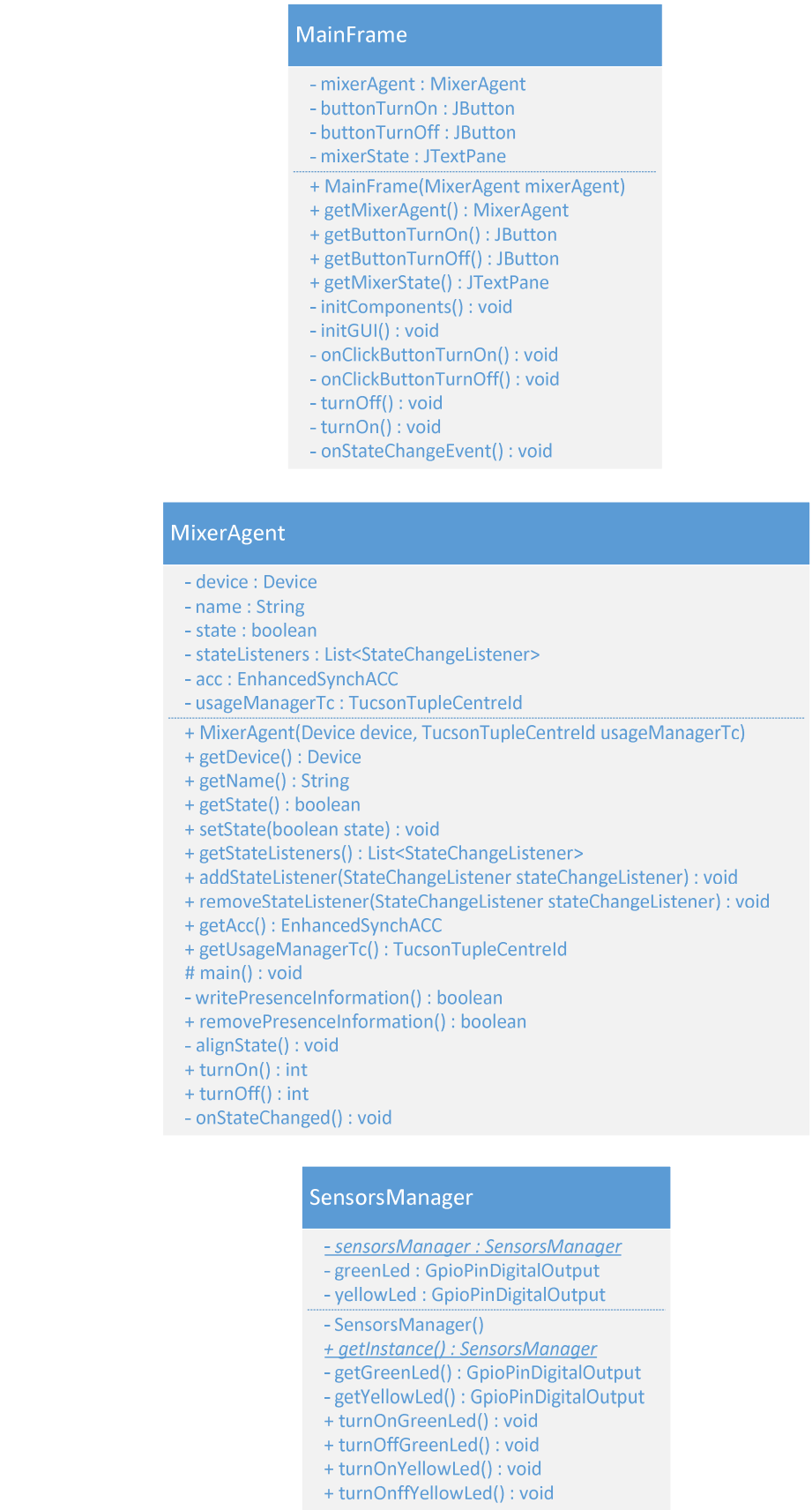
Gestore dei consumi:



Dispositivo simulato:



Dispositivo fisico:

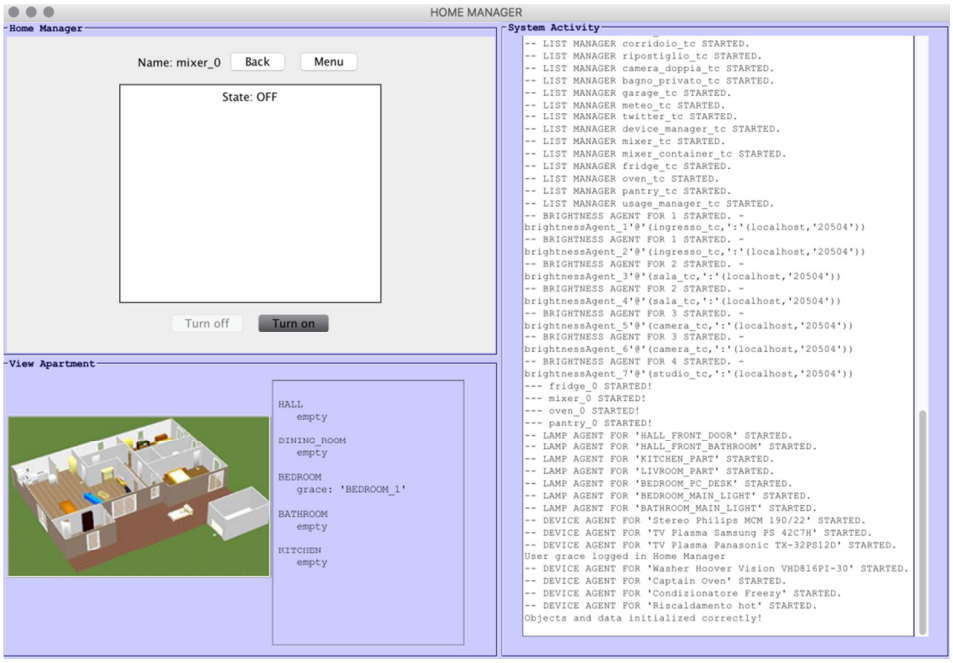


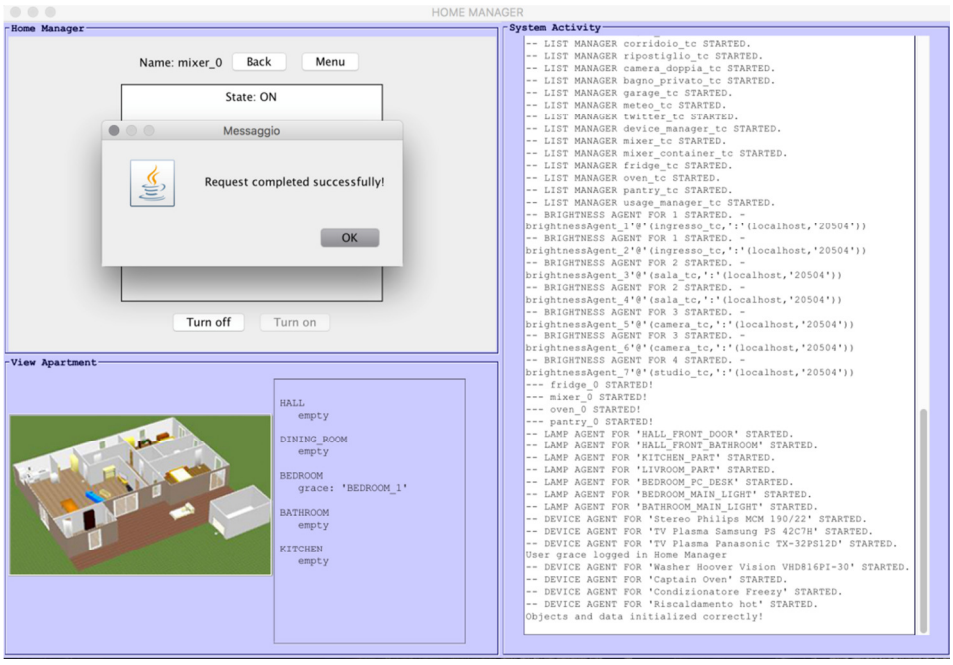
5.6 Collaudo delle funzionalità

Terminata l'implementazione del dispositivo simulato e fisico è possibile procedere con il collaudo delle funzionalità.

Per prima cosa si testano le funzionalità del solo dispositivo simulato.

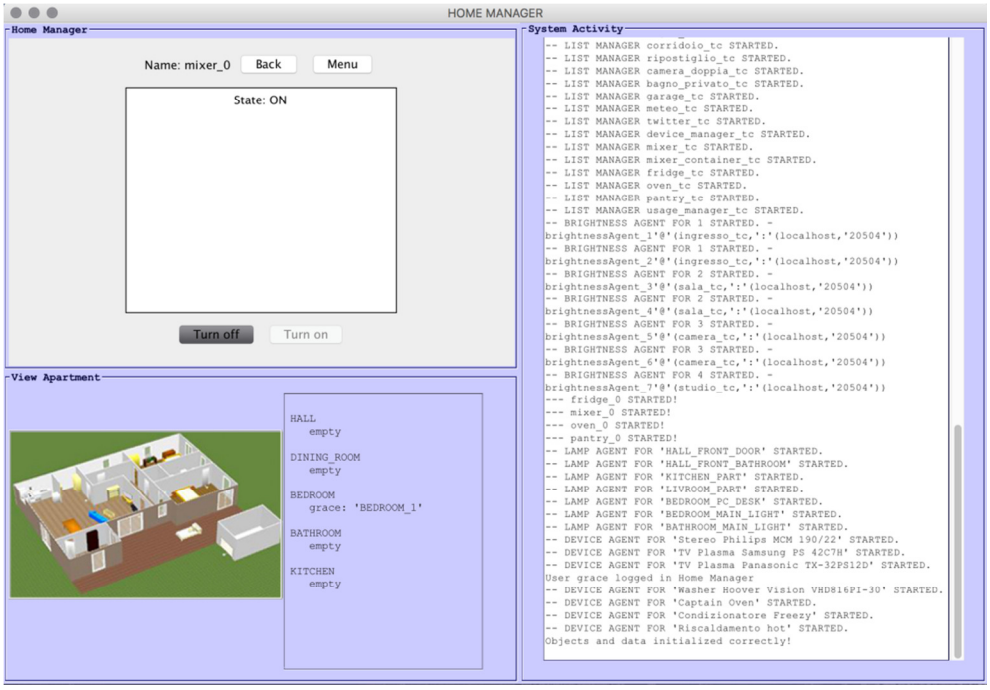
La pressione del pulsante "Turn on" comporta l'accensione del dispositivo, l'attivazione del bottone "Turn off" e la disattivazione del bottone "Turn on":

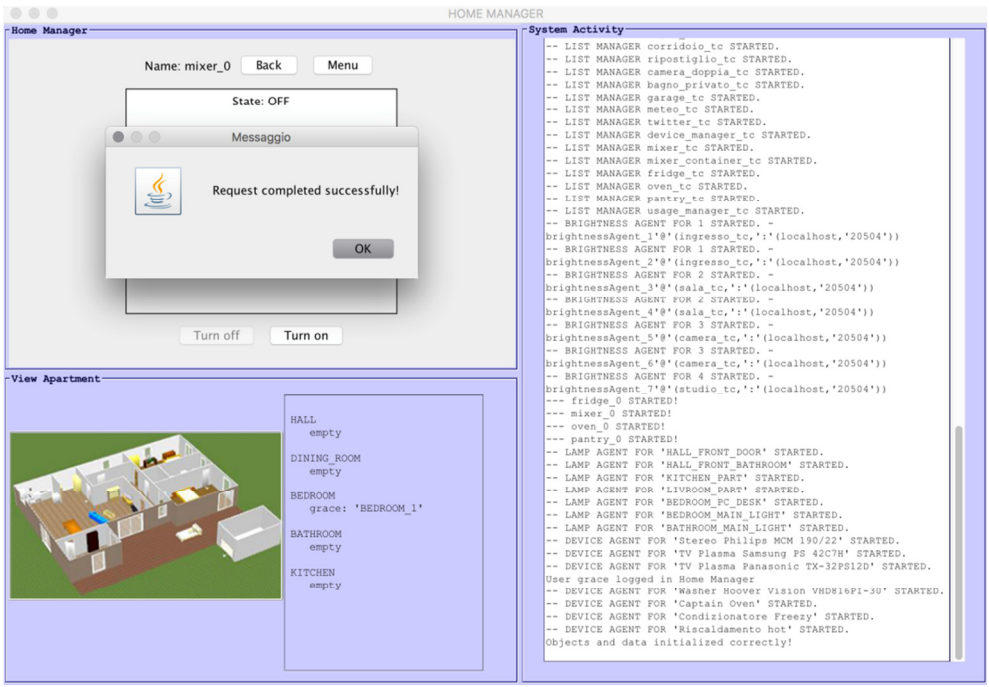




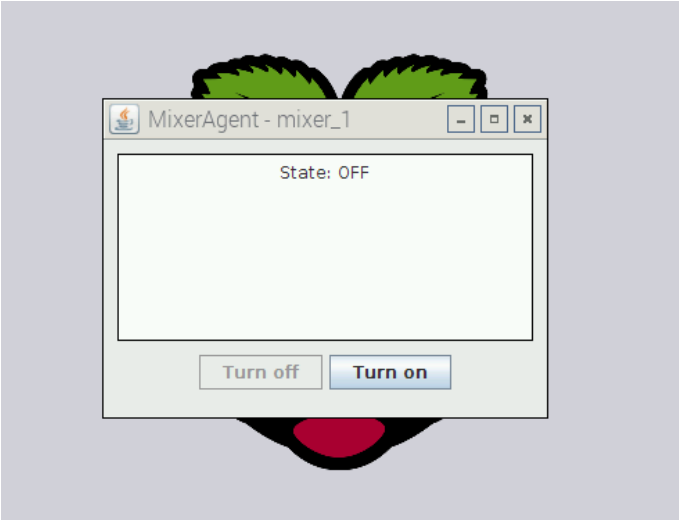
In questa prima fase del collaudo l'accensione è sempre consentita, in quanto il dispositivo che richiede l'accensione è il `mixer_0`, ossia il dispositivo simulato già presente al primo utilizzo del sistema ed il cui consumo è 0 watt.

Viceversa, la pressione del pulsante "Turn off" comporta lo spegnimento del dispositivo, l'attivazione del pulsante "Turn on" e la disattivazione del pulsante "Turn off":





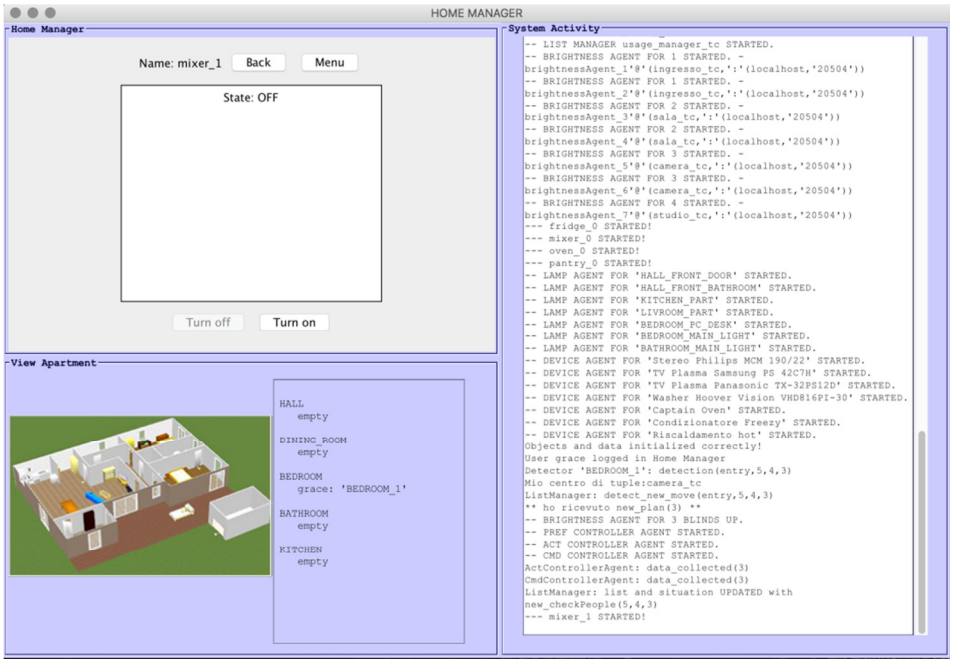
Collaudate le funzionalità del dispositivo simulato è possibile procedere con il collaudo delle funzionalità di allineamento.
Per prima cosa, quindi, è necessario avviare il dispositivo fisico.



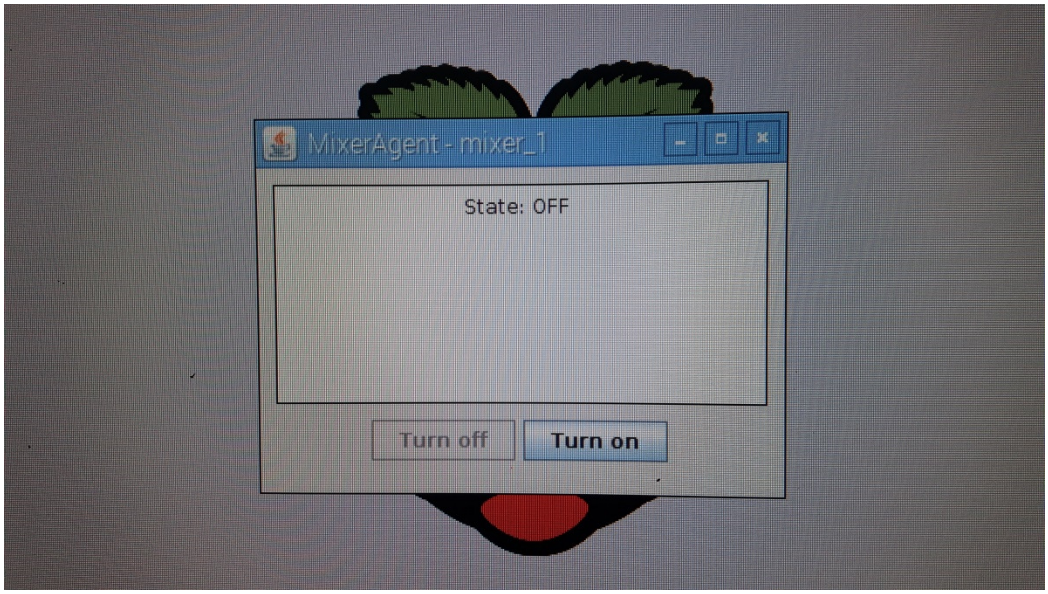
Il suo primo avvio comporta, grazie alla presenza dell'auto-detect, la comparsa di un nuovo dispositivo nella lista dei dispositivi del sistema: `mixer_1`.

Selezionando tale dispositivo e successivamente l'opzione "Properties" verrà mostrato il pannello MixerPanel, come nel caso precedente, che consentirà però in questo caso di interagire con il dispositivo simulato `mixer_1`, associato al fisico avviato nel Raspberry.

Al momento dell’avvio del dispositivo fisico, il dispositivo simulato è spento.

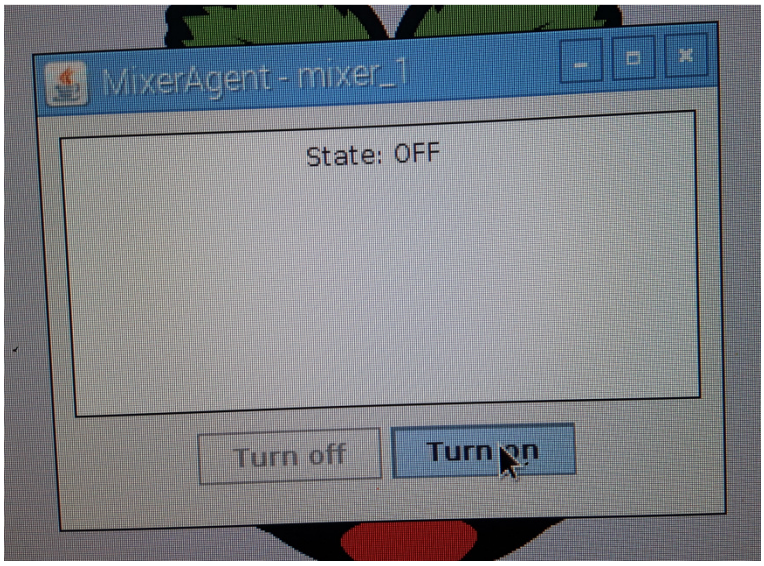


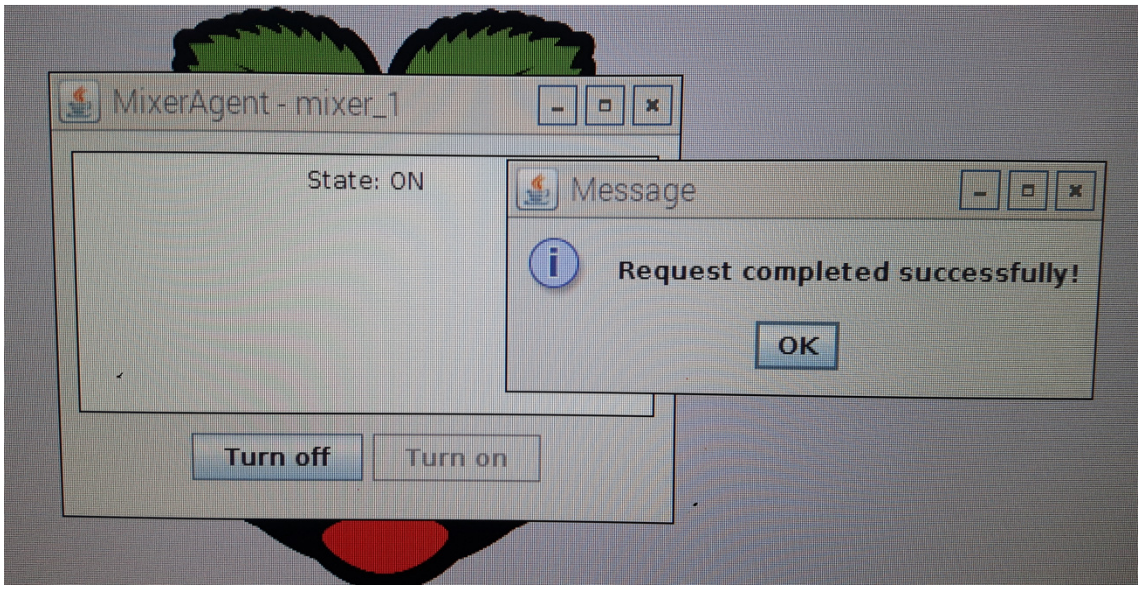
Quindi anche il dispositivo fisico risulterà spento.

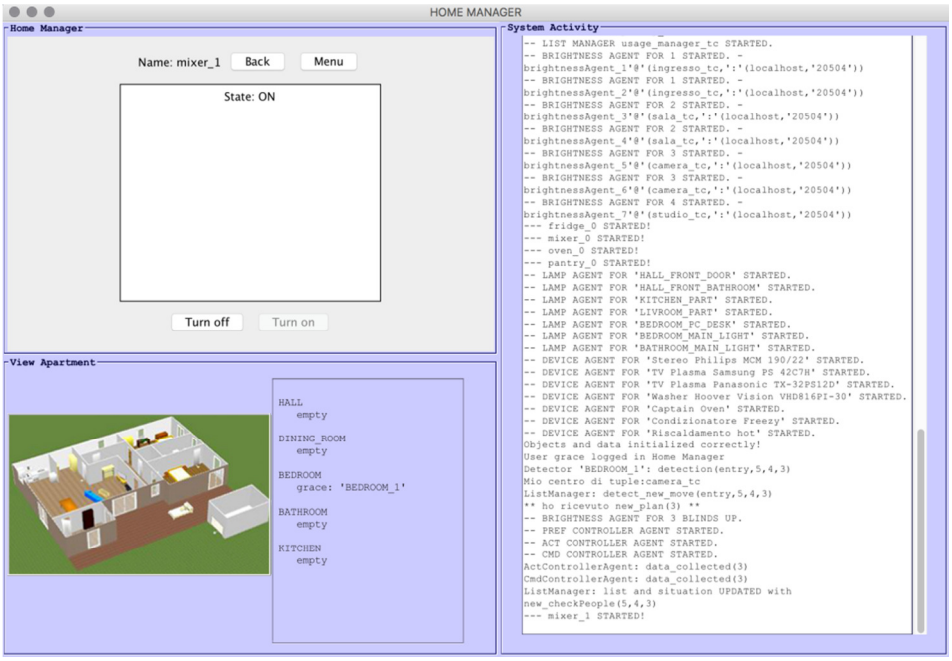


E di conseguenza il led giallo risulterà acceso mentre il led verde risulterà spento.

Premendo il pulsante “Turn on” dell’interfaccia grafica su Raspberry, lo stato del dispositivo fisico passerà ad “on”, nel caso in cui il gestore dei consumi consenta l’accensione, e di conseguenza il led verde verrà acceso, il led giallo verrà spento ed il dispositivo simulato associato cambierà automaticamente stato, accendendosi.

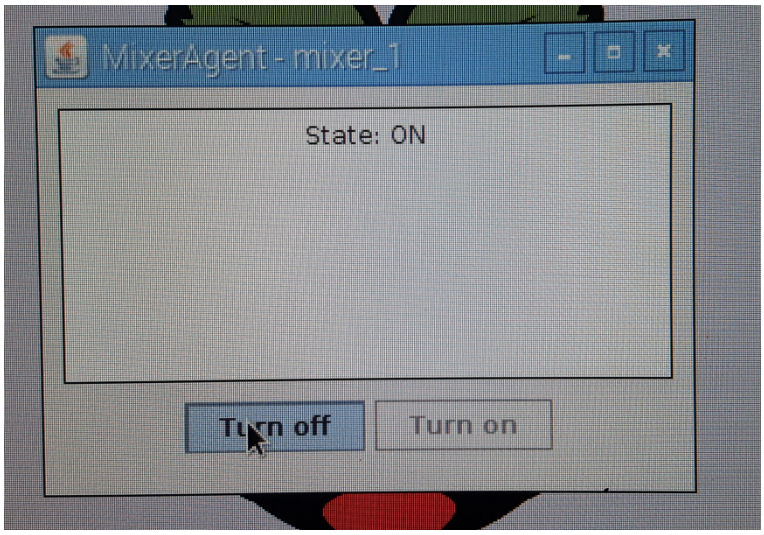


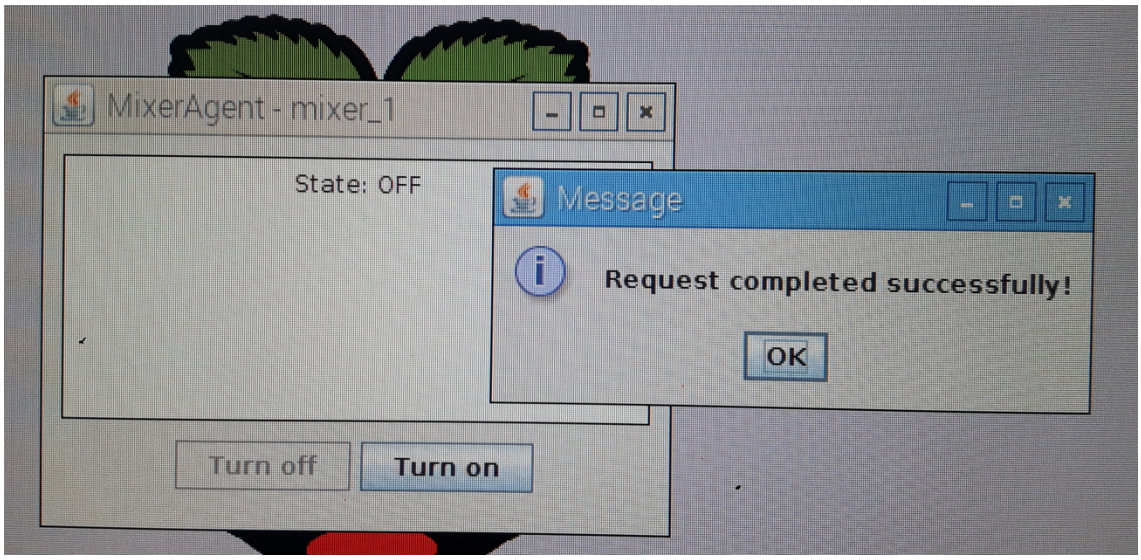

MixerAgent - mixer_1
State: ON
Turn off
Turn on
Message
Request completed successfully!
OK

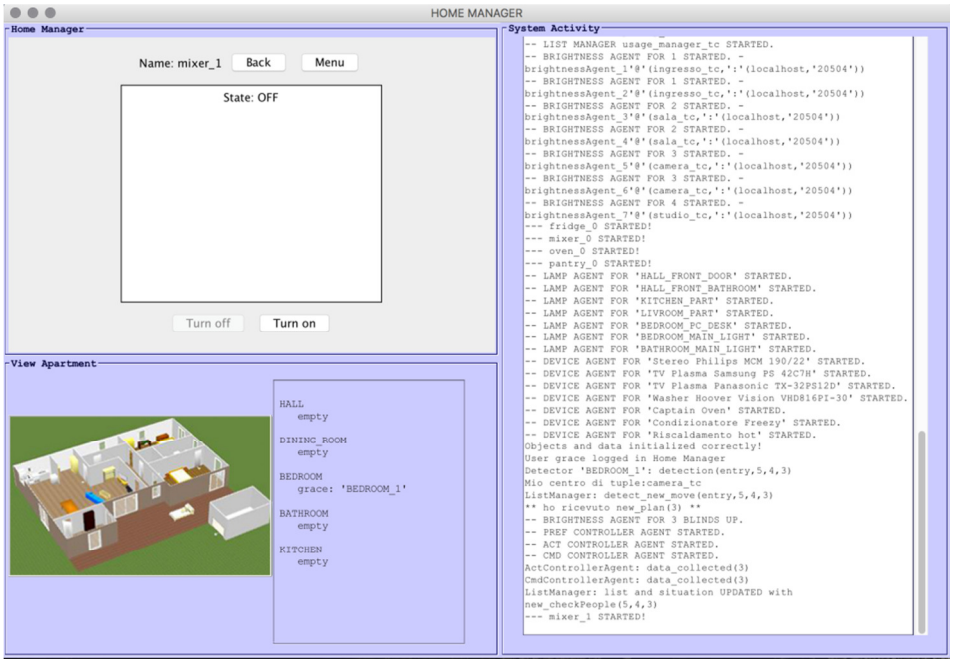


L’accensione del dispositivo, in questa fase di collaudo, verrà sempre consentita in quanto nel sistema non sono presenti altri dispositivi fisici ed il consumo dello SmartMixer è inferiore a quello di soglia dell’abitazione.

Viceversa, se viene premuto il pulsante “Turn off” del dispositivo fisico, il suo stato passerà ad “off”, il led giallo si accenderà, il led verde si spegnerà ed il dispositivo simulato associato cambierà automaticamente stato, spegnendosi.

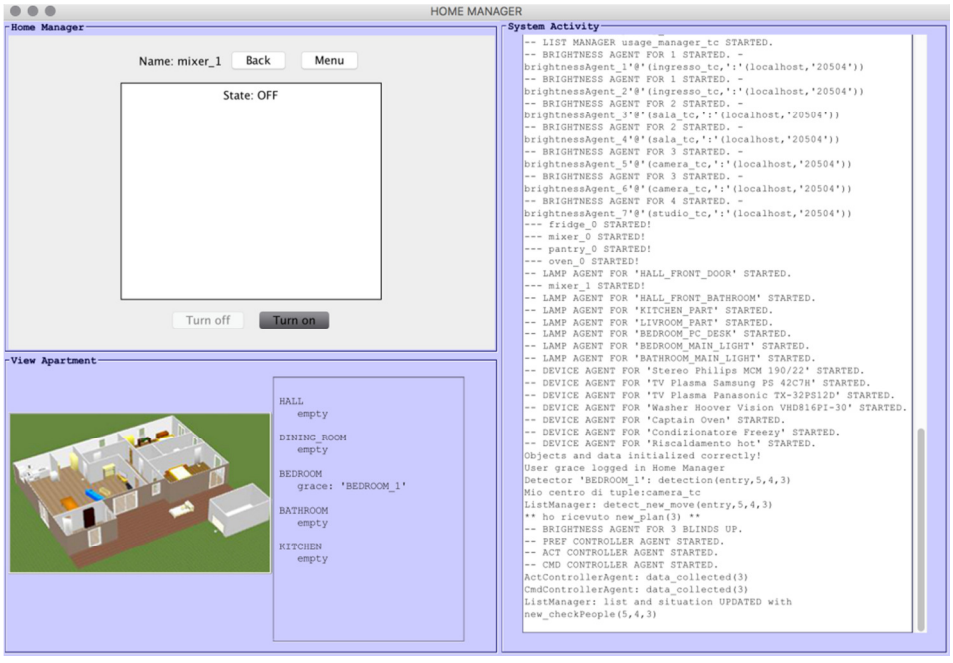


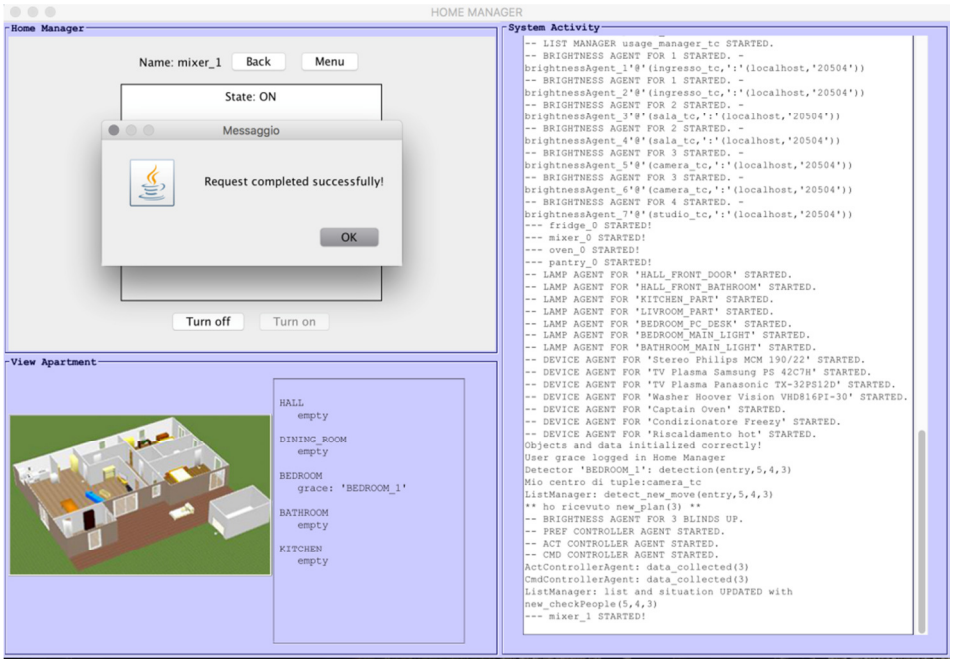

MixerAgent - mixer_1
State: OFF
Turn off
Turn on
Message
Request completed successfully!
OK

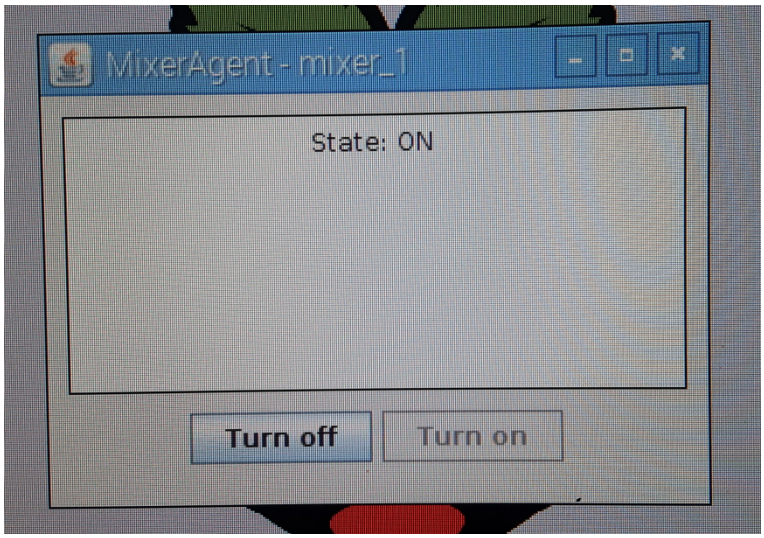


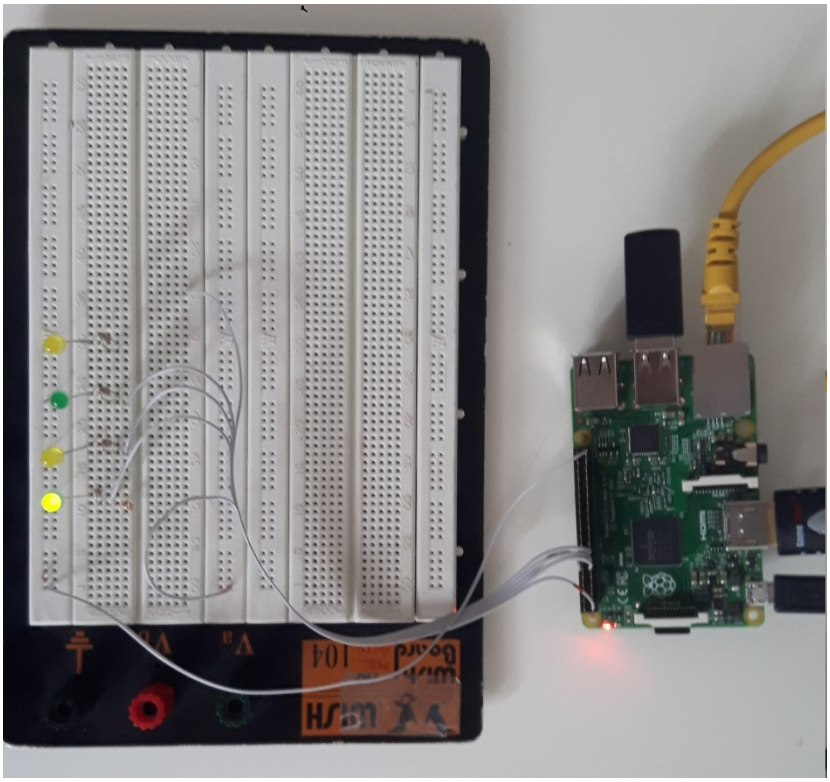
Una volta collaudata la capacità di allinearsi del dispositivo simulato, è necessario collaudare la capacità del dispositivo fisico di allinearsi al dispositivo simulato.

Se nell'interfaccia grafica del dispositivo simulato viene premuto il bottone "Turn on", il dispositivo simulato si accende, solo se il gestore dei consumi restituisce esito positivo, il dispositivo fisico associato si accende automaticamente, il led verde si accende ed il led giallo si spegne.



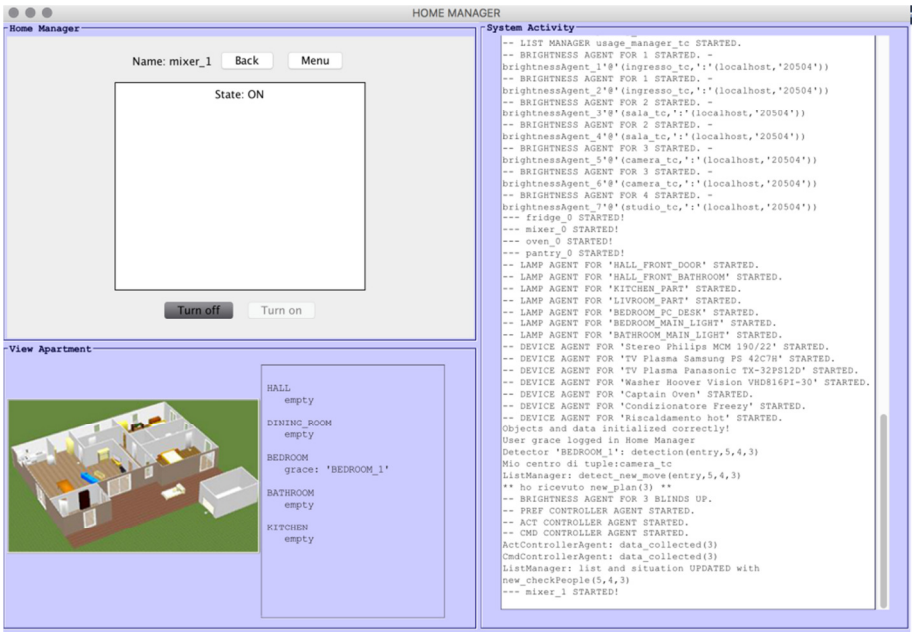

HOME MANAGER
Home Manager
Name: mixer_1
Back
Menu
State: ON
Messaggio
Request completed successfully!
OK
Turn off
Turn on
View Apartment
HALL
empty
DINING_ROOM
empty
BEDROOM
grace: 'BEDROOM_1'
BATHROOM
empty
KITCHEN
empty
System Activity
-- LIST MANAGER usage_manager_tc STARTED.
-- BRIGHTNESS AGENT FOR 1 STARTED. -
brightnessAgent_1'@'(ingresso_tc,':'(localhost,'20504'))
-- BRIGHTNESS AGENT FOR 1 STARTED. -
brightnessAgent_2'@'(ingresso_tc,':'(localhost,'20504'))
-- BRIGHTNESS AGENT FOR 2 STARTED. -
brightnessAgent_3'@'(sala_tc,':'(localhost,'20504'))
-- BRIGHTNESS AGENT FOR 2 STARTED. -
brightnessAgent_4'@'(sala_tc,':'(localhost,'20504'))
-- BRIGHTNESS AGENT FOR 3 STARTED. -
brightnessAgent_5'@'(camera_tc,':'(localhost,'20504'))
-- BRIGHTNESS AGENT FOR 3 STARTED. -
brightnessAgent_6'@'(camera_tc,':'(localhost,'20504'))
-- BRIGHTNESS AGENT FOR 4 STARTED. -
brightnessAgent_7'@'(studio_tc,':'(localhost,'20504'))
--- fridge_0 STARTED!
--- mixer_0 STARTED!
--- oven_0 STARTED!
--- pantry_0 STARTED!
-- LAMP AGENT FOR 'HALL_FRONT_DOOR' STARTED.
-- LAMP AGENT FOR 'HALL_FRONT_BATHROOM' STARTED.
-- LAMP AGENT FOR 'KITCHEN_PART' STARTED.
-- LAMP AGENT FOR 'LIVROOM_PART' STARTED.
-- LAMP AGENT FOR 'BEDROOM_PC_DESK' STARTED.
-- LAMP AGENT FOR 'BEDROOM_MAIN_LIGHT' STARTED.
-- LAMP AGENT FOR 'BATHROOM_MAIN_LIGHT' STARTED.
-- DEVICE AGENT FOR 'Stereo Philips MCM 190/22' STARTED.
-- DEVICE AGENT FOR 'TV Plasma Samsung PS 42C7H' STARTED.
-- DEVICE AGENT FOR 'TV Plasma Panasonic TX-32PS12D' STARTED.
-- DEVICE AGENT FOR 'Washer Hoover Vision VHD816PI-30' STARTED.
-- DEVICE AGENT FOR 'Captain Oven' STARTED.
-- DEVICE AGENT FOR 'Condizionatore Freezy' STARTED.
-- DEVICE AGENT FOR 'Riscaldamento hot' STARTED.
Objects and data initialized correctly!
User grace logged in Home Manager
Detector 'BEDROOM_1': detection(entry,5,4,3)
Mio centro di tuple:camera_tc
ListManager: detect_new_move(entry,5,4,3)
** ho ricevuto new_plan(3) **
-- BRIGHTNESS AGENT FOR 3 BLINDS UP.
-- PREF CONTROLLER AGENT STARTED.
-- ACT CONTROLLER AGENT STARTED.
-- CMD CONTROLLER AGENT STARTED.
ActControllerAgent: data_collected(3)
CmdControllerAgent: data_collected(3)
ListManager: list and situation UPDATED with
new_checkPeople(5,4,3)
--- mixer_1 STARTED!

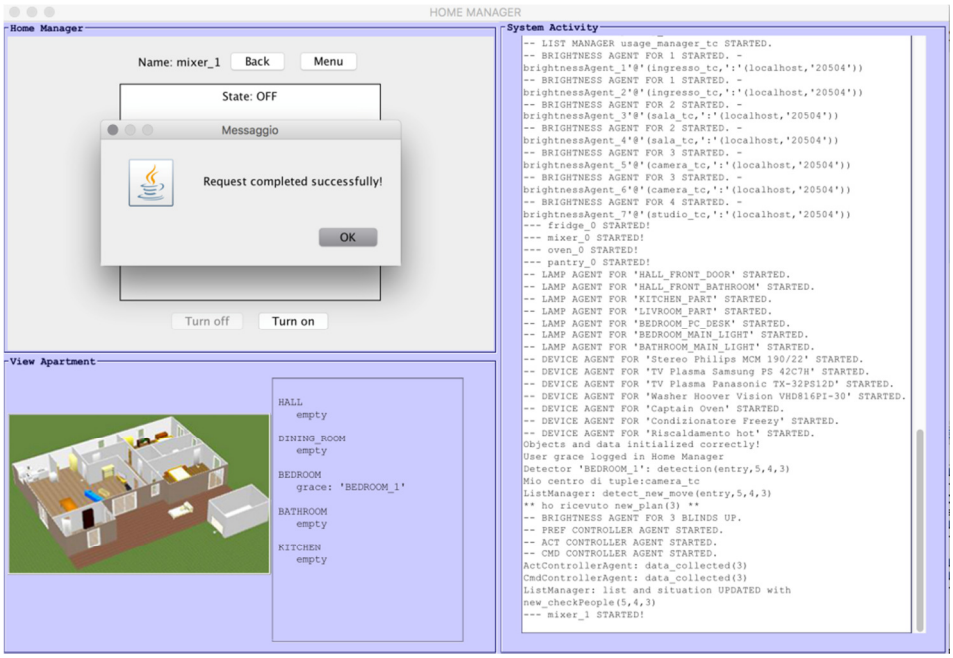

MixerAgent - mixer_1
State: ON
Turn off
Turn on

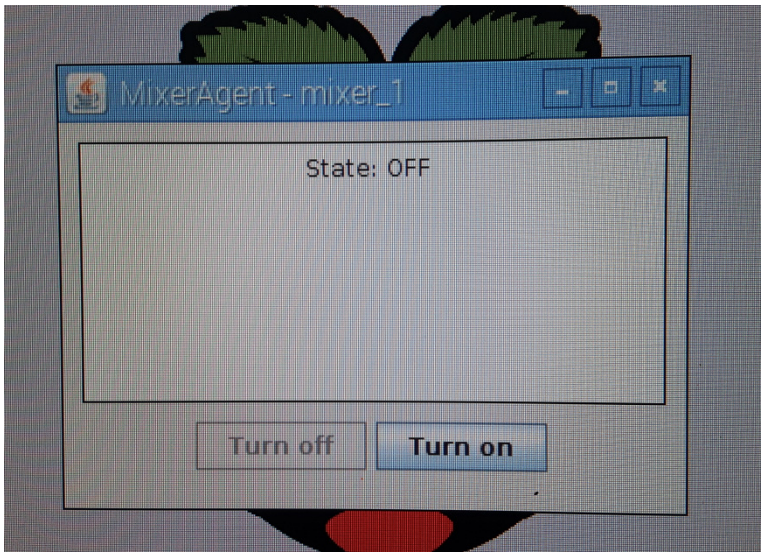


Anche in questo caso il dispositivo simulato riceverà sempre esito positivo dal gestore dei consumi, in quanto non c'è nessun altro dispositivo che consuma corrente connesso al sistema.

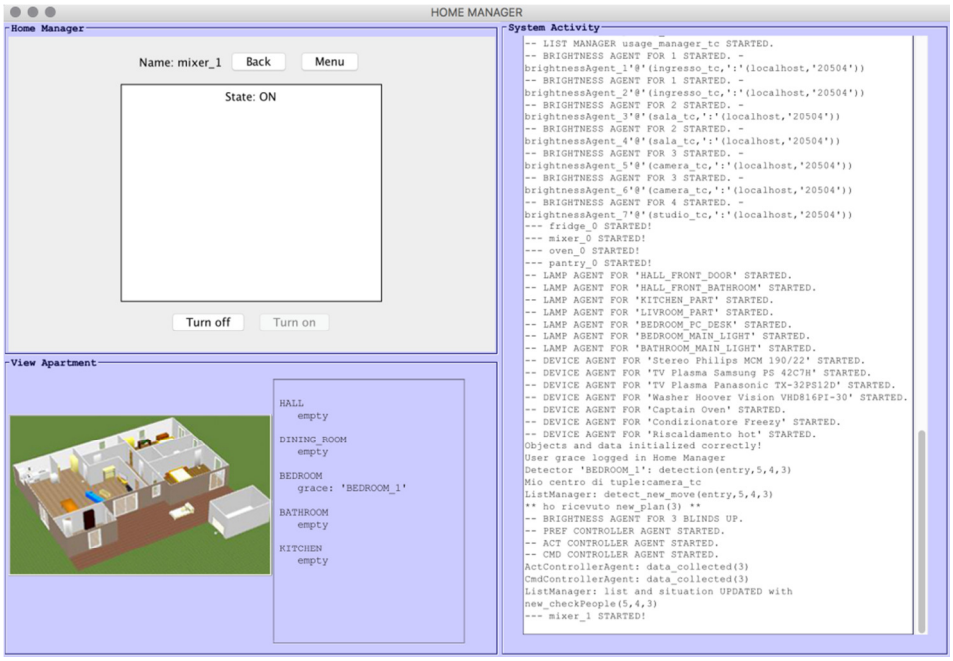
Viceversa, se nell'interfaccia grafica del dispositivo simulato viene premuto il pulsante "Turn off", il dispositivo simulato si spegne, il dispositivo fisico associato si spegne automaticamente, il led giallo si accende ed il led verde si spegne.

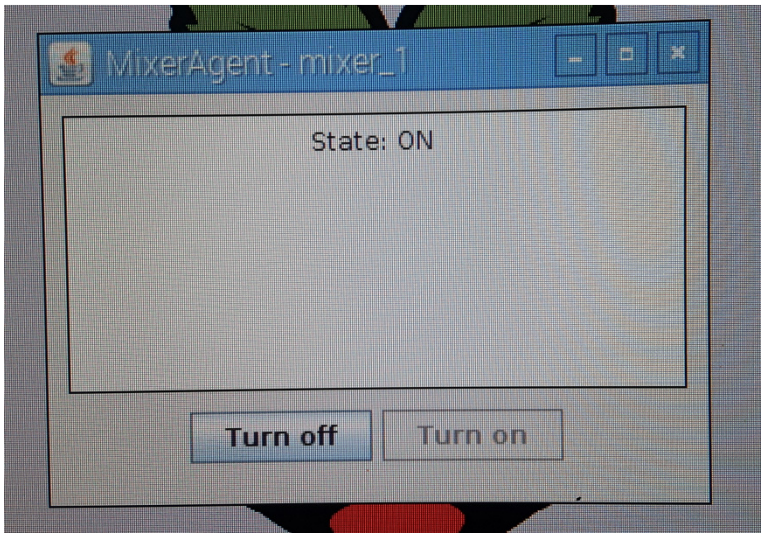



HOME MANAGER
Home Manager
Name: mixer_1
Back
Menu
State: OFF
Messaggio
Request completed successfully!
OK
Turn off
Turn on
View Apartment
HALL
empty
DINING_ROOM
empty
BEDROOM
grace: 'BEDROOM_1'
BATHROOM
empty
KITCHEN
empty
System Activity
-- LIST MANAGER usage_manager_tc STARTED.
-- BRIGHTNESS AGENT FOR 1 STARTED. -
brightnessAgent_1'@'(ingresso_tc,':'(localhost,'20504'))
-- BRIGHTNESS AGENT FOR 1 STARTED. -
brightnessAgent_2'@'(ingresso_tc,':'(localhost,'20504'))
-- BRIGHTNESS AGENT FOR 2 STARTED. -
brightnessAgent_3'@'(sala_tc,':'(localhost,'20504'))
-- BRIGHTNESS AGENT FOR 2 STARTED. -
brightnessAgent_4'@'(sala_tc,':'(localhost,'20504'))
-- BRIGHTNESS AGENT FOR 3 STARTED. -
brightnessAgent_5'@'(camera_tc,':'(localhost,'20504'))
-- BRIGHTNESS AGENT FOR 3 STARTED. -
brightnessAgent_6'@'(camera_tc,':'(localhost,'20504'))
-- BRIGHTNESS AGENT FOR 4 STARTED. -
brightnessAgent_7'@'(studio_tc,':'(localhost,'20504'))
--- fridge_0 STARTED!
--- mixer_0 STARTED!
--- oven_0 STARTED!
--- pantry_0 STARTED!
-- LAMP AGENT FOR 'HALL_FRONT_DOOR' STARTED.
-- LAMP AGENT FOR 'HALL_FRONT_BATHROOM' STARTED.
-- LAMP AGENT FOR 'KITCHEN_PART' STARTED.
-- LAMP AGENT FOR 'LIVROOM_PART' STARTED.
-- LAMP AGENT FOR 'BEDROOM_PC_DESK' STARTED.
-- LAMP AGENT FOR 'BEDROOM_MAIN_LIGHT' STARTED.
-- LAMP AGENT FOR 'BATHROOM_MAIN_LIGHT' STARTED.
-- DEVICE AGENT FOR 'Stereo Philips MCM 190/22' STARTED.
-- DEVICE AGENT FOR 'TV Plasma Samsung PS 42C7H' STARTED.
-- DEVICE AGENT FOR 'TV Plasma Panasonic TX-32PS12D' STARTED.
-- DEVICE AGENT FOR 'Washer Hoover Vision VHD816PI-30' STARTED.
-- DEVICE AGENT FOR 'Captain Oven' STARTED.
-- DEVICE AGENT FOR 'Condizionatore Freezy' STARTED.
-- DEVICE AGENT FOR 'Riscaldamento hot' STARTED.
Objects and data initialized correctly!
User grace logged in Home Manager
Detector 'BEDROOM_1': detection(entry,5,4,3)
Mio centro di tuple:camera_tc
ListManager: detect_new_move(entry,5,4,3)
** ho ricevuto new_plan(3) **
-- BRIGHTNESS AGENT FOR 3 BLINDS UP.
-- PREF CONTROLLER AGENT STARTED.
-- ACT CONTROLLER AGENT STARTED.
-- CMD CONTROLLER AGENT STARTED.
ActControllerAgent: data_collected(3)
CmdControllerAgent: data_collected(3)
ListManager: list and situation UPDATED with
new_checkPeople(5,4,3)
--- mixer_1 STARTED!


MixerAgent - mixer_1
State: OFF
Turn off
Turn on

L’ultima operazione di collaudo per le funzionalità di allineamento è quella di attivare il dispositivo fisico solo dopo aver acceso il dispositivo simulato ad esso corrispondente e verificare che il suo stato passi automaticamente da spento ad acceso.





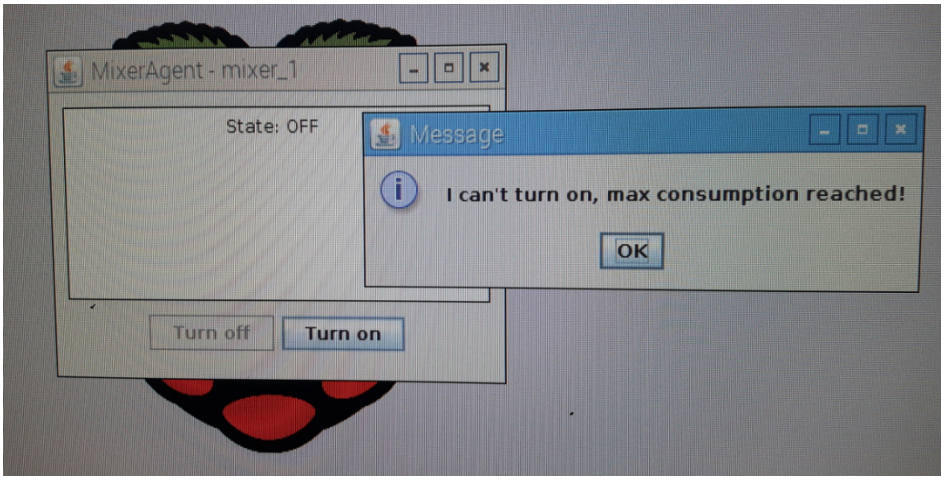
Il collaudo delle funzionalità di allineamento è stato interamente effettuato.

A questo punto si vuole collaudare la capacità dello UsageManagerAgent di impedire ad un dispositivo di accendersi quando il suo consumo comporta il superamento del valore di soglia dell'abitazione.

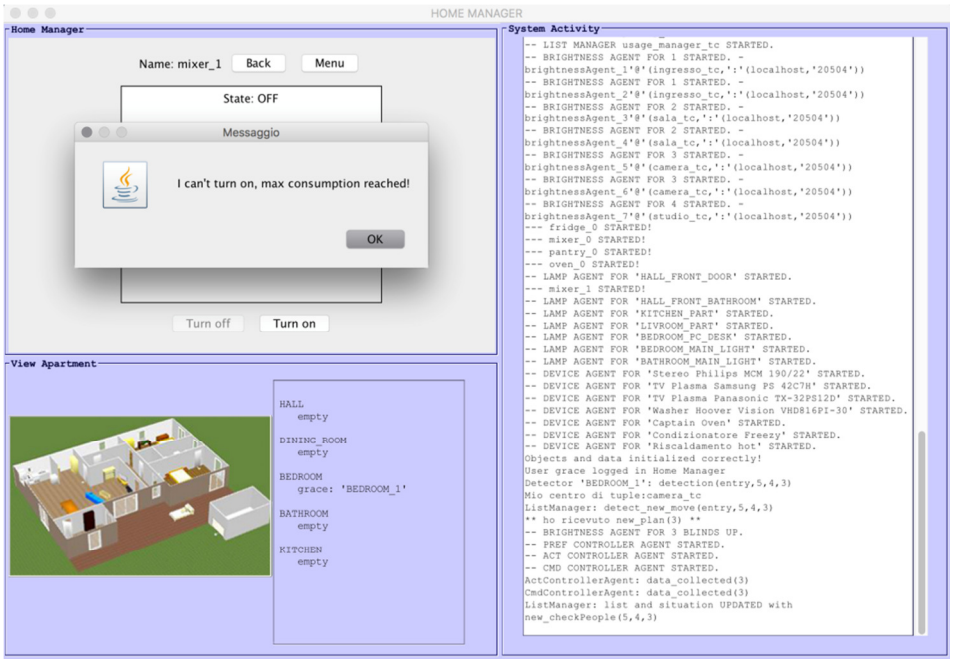
Il massimo consumo attuale dell'abitazione è impostato a 3Kw, ossia il valore che si ha di norma.

Andando a modificare il valore del consumo del dispositivo nel file Configuration.java nel package *model.configuration* del progetto su Raspberry Pi ed inserendo, per esempio, 3.2Kw, non sarà possibile accendere il dispositivo. Il sistema dovrà quindi segnalare all'utente che il dispositivo non può accendersi.

Infatti, una volta modificato il valore come indicato, se viene premuto il pulsante "Turn on" dell'interfaccia grafica del dispositivo fisico, il dispositivo rimarrà in stato spento, di conseguenza il dispositivo simulato associato non subirà alcuna modifica di stato e verrà mostrata una segnalazione all'utente.



Allo stesso modo, se viene premuto il pulsante "Turn on" dell'interfaccia grafica del dispositivo simulato, il dispositivo rimarrà in stato spento, di conseguenza il dispositivo fisico associato non subirà alcuna modifica di stato e verrà mostrata una segnalazione all'utente.



Capitolo 6 – Conclusioni e sviluppi futuri

Scopo principale della tesi era l'analisi di possibili scenari in ambiente Home Manager, partendo dal prototipo attuale.

In fase di sviluppo della tesi si è cercato di analizzare il prototipo attuale di Home Manager affinché gli scenari proposti offrissero non solo funzionalità del tutto nuove e non ancora proposte, ma anche funzionalità già esistenti nel sistema ma reingegnerizzate.
Infatti, soprattutto per quanto riguarda lo scenario della cucina proposto al capitolo 3, si è cercato di prestare attenzione ad una corretta suddivisione delle responsabilità nonché alla presentazione di scenari che, se realizzati, possano essere facilmente estendibili.

Le funzionalità proposte sotto forma di scenario potranno essere progettate ed implementate in futuro, rappresentando quindi uno sviluppo di quanto è stato presentato nella tesi.

Il secondo obiettivo della tesi era la progettazione e l'implementazione degli aspetti più rilevanti degli scenari proposti.

Per il raggiungimento di questo scopo si è deciso di progettare ed implementare le funzionalità per il controllo del consumo dell'abitazione.
Inoltre, sono state poste le basi per la realizzazione dei futuri agenti implementando i metodi di accensione e spegnimento degli stessi.

Il gestore dei consumi realizzato rappresenta l'entità intermedia di un progetto molto più ampio.
Infatti, il gestore dei consumi si appoggia alle funzionalità di più basso livello già implementate in Home Manager, che si occupano della gestione dei dispositivi, ed offre funzionalità considerabili di basso livello per tutte le entità che si appoggiano, o si appoggeranno, su di lui.
Queste entità sono gli agenti correlati ai dispositivi con cui l'utente interagisce, che sfrutteranno le funzionalità offerte dal gestore dei consumi, senza doverle offrire direttamente, e saranno dotati di ulteriori funzionalità secondo le proprie responsabilità.

Anche in quest'ottica il progetto, considerato nell'insieme, rappresenta un punto di partenza per lo sviluppo di altre funzionalità.

Bibliografia

[1] “La domotica ridisegna l’abitare. Efficienza e risparmio energetico nella casa intelligente” – Giampiero Filella

[2] “Domotica. Sistemi elettronici per la gestione della casa” – Comitato elettrotecnico italiano

[3] Prof. Enrico Denti, Ing. Roberta Calegari, “Home Manager” http://apice.unibo.it/xwiki/bin/view/Products/HomeManager

[4] Prof. Enrico Denti, Ing. Roberta Calegari, “SODA” http://apice.unibo.it/xwiki/bin/view/SODA/WebHome

[5] Prof. Enrico Denti, Ing. Roberta Calegari, “TuCSoN” http://apice.unibo.it/xwiki/bin/view/TuCSoN/WebHome

[6] Prof. Andrea Omicini, Prof. Stefano Mariani, “The TuCSoN Coordination Model & Technology” http://amsacta.unibo.it/3632/1/tucson-guide.pdf

[7] Prof. Enrico Denti, Ing. Roberta Calegari, “Butler-ising Home Manager: a Pervasive Multi-Agent System for Home Intelligence” http://apice.unibo.it/xwiki/bin/view/Publications/HomeMan2icaart2015

[8] “Raspberry Pi. Guida all’uso” – Valter Minute

[9] “The Pi4J project. Java I/O library for the Rasperry Pi” http://pi4j.com/

[10] “Pin Numering – Raspberry Pi 2 Model B” http://pi4j.com/pins/model-2b-rev1.html

Ringraziamenti

Vorrei ringraziare il Professore Enrico Denti e l'Ingegnere Roberta Calegari, in qualità di relatore e correlatore, per avermi offerto l'opportunità di collaborare con loro nello sviluppo di Home Manager, per il supporto e per la disponibilità che mi hanno dato in tutto questo periodo.

Vorrei ringraziare i miei genitori per avermi incitata nell'intraprendere questo percorso, per aver sempre creduto in me e per avermi sempre sostenuta, sia economicamente che emotivamente, nel corso di questi tre anni.

Vorrei ringraziare mia sorella Alessia per essere stata la mia "maestra di vita" in questo percorso e per essere stata sempre presente.

Vorrei ringraziare i miei nonni ed i miei zii per l'interesse dimostrato nei confronti del mio percorso e per il sostegno psicologico offertomi.

Vorrei ringraziare i miei cuginetti, Riccardo e Sofia, per avermi regalato momenti di distrazione, gioco e svago quando lo stress e lo studio erano all'ordine del giorno.

Vorrei ringraziare il mio fidanzato Mattia per aver condiviso con me l'ultimo anno di questo percorso, per essermi stato accanto nei momenti più difficili e per avermi insegnato a credere in me e nelle mie capacità.

Vorrei ringraziare Luca per la disponibilità, la pazienza e per il bellissimo "gioco di squadra" intrapreso.

Vorrei ringraziare i miei amici e compagni di corso Francesco, Alessio, Federico, Federico, Paolo, Gabriele, Nicola, Matteo, Stefano, Andrea per aver condiviso con me successi e insuccessi, momenti felici e tristi.

Infine ringrazio tutte le persone che ho incontrato lungo questo percorso e che non ho ancora citato, per aver condiviso dei momenti con me.