ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA
SCUOLA DI INGEGNERIA E ARCHITETTURA
Corso di Laurea in Ingegneria Elettronica, Informatica e delle Telecomunicazioni

COORDINAZIONE SITUATA PER LA DOMOTICA: BUTLERS IN TUCSON

Elaborata nel corso di: Sistemi Distribuiti

Relatore:
Prof. ANDREA OMICINI
Correlatori:
Prof. ENRICO DENTI
Ph.D. STEFANO MARIANI

Presentata da:
ATTILIO VERGIGLIO POMETTO

SESSIONE II, SECONDO APPELLO
ANNO ACCADEMICO 2014–2015

PAROLE CHIAVE

Domotica
Coordinazione
TuCSoN
Butlers Architecture
Situatedness

Indice

Abstract **iii**

1 Introduzione **1**

1.1 La vita con un maggiordomo virtuale 1
1.2 Introduzione alla domotica 4
1.2.1 Un po' di storia 4
1.2.2 Le soluzioni attuali 6

2 L'architettura Butlers **7**

2.1 Perché un maggiordomo virtuale? 8
2.2 Requisiti 9
2.2.1 Requisiti generali 9
2.2.2 Requisiti di coordinazione 9
2.2.3 Requisiti di configurazione 10
2.2.4 Requisiti per l'interfaccia utente 11
2.2.5 Requisiti funzionali 12
2.2.6 Requisiti per la *gamification* 13
2.3 L'architettura Butlers: i livelli 13
2.3.1 I sette livelli 13
2.3.2 Alcune possibili configurazioni 15
2.3.3 Butlers Architecture e Home Manager 17

3 TuCSoN **19**

3.1 Il modello e l'infrastruttura TuCSoN 19
3.2 Perchè scegliere TuCSoN 21
3.3 I transducer 22

4 Home Manager **26**

4.1 Il prototipo attuale 26
4.2 Architettura software 27
4.2.1 L'architettura desiderata (e desiderabile) 27

4.2.2 L'architettura del prototipo 28
4.3 L'implementazione di Home Manager 32
4.3.1 API per gli agenti 32
4.3.2 API per i Tuple Centre 32
4.3.3 API generiche 32

5 Analisi e progettazione 34
5.1 Analisi della gestione delle luci 34
5.2 Progettazione 36
5.2.1 Il transducer attuatore 37
5.2.2 Il transducer attuatore-sensore 38
5.2.3 Il probe 39
5.2.4 Le Reaction 40

6 L'implementazione 42
6.1 Obbiettivo dell'implementazione 43
6.2 LightTransducer 43
6.2.1 LightTransducer Sensore 44
6.2.2 LightTransducer Attuatore 44
6.3 ActualLight 45
6.3.1 Lettura dello stato 45
6.3.2 Scrittura dello stato 45
6.4 LampAgent 46
6.4.1 Caricamento delle specifiche 47
6.4.2 Generazione della tupla di configurazione 48
6.4.3 Cambio di stato 48
6.5 La lettura dello stato 49
6.6 Sostituibilità: un esempio 50

7 Conclusioni 52
7.1 Sviluppi futuri 52

Bibliografia 55

Abstract

Siamo ormai abituati a vivere in un mondo pieno di dispositivi intelligenti ed un sistema domotico deve essere facilmente integrato con essi.
L'obbiettivo di questa tesi è di estendere con il concetto di *transducer* il prototipo di Home Manager, applicazione per la gestione di una casa intelligente che sfrutta la tecnologia TuCSoN.
I vantaggi di questa scelta sono molteplici: permettendo al media di coordinazione di gestire le interazioni fra gli agenti e l'ambiente, si separano i problemi implementativi da quelli coordinativi, guadagnando anche un sistema più facilmente ispezionabile, con componenti sostituibili e manutenibili.

Dopo un'introduzione alla domotica, all'architettura Butlers e all'infrastruttura TuCSoN, pilastri su cui è basato Home Manager, si passerà ad una fase di analisi dello stato attuale del prototipo, per comprendere dove e perché andare a introdurre il concetto di *transducer*. Seguiranno poi le fasi di progettazione e, infine, di implementazione di questa tecnologia in Home Manager.

Capitolo 1

Introduzione

Al giorno d'oggi nelle nostre case abbiamo sempre più dispositivi intelligenti: dalle *smart* TV ai *tablet*, dai termostati che possiamo comandare con lo *smartphone* (Netatmo, 2015) ai condizionatori programmabili da remoto tramite una connessione a internet (Samsung, 2015), passando per allarmi sempre più sofisticati e connessi (iSmartAlarm, 2015) e caffettiere elettriche che fungono da sveglia (DeLonghi, 2015).
Questi oggetti così eterogenei, però, hanno una grande limitazione: **non interagiscono fra loro**. Per cui se si imposta una sveglia sul *tablet*, va ricordato di regolarla anche sulla caffettiera per poter avere il caffé pronto al risveglio, ed anche di programmare i giusti orari di funzionamento del condizionatore o dell'impianto di riscaldamento perché le stanze siano confortevoli una volta che ci si alza.
Il grande vantaggio di un sistema domotico come quello dipinto nella *Butlers Architecture*, è di poter avere tutti i dispositivi connessi fra loro come un sistema unico e integrato. (Denti, 2014)
Prima di vedere nel dettaglio questa architettura, si andrà ad illustrarne le potenzialità descrivendo la giornata tipo di una famiglia che ha scelto di venire accudita da un maggiordomo virtuale.

1.1 La vita con un maggiordomo virtuale

Chiara, oggi decide di uscire prima dal lavoro per poter fare delle commissioni. Appena lasciato l'ufficio, Battista, il suo maggiordomo, si accorge dello spostamento ed inizia ad elaborare una strategia per rendere la casa confortevole al ritorno: innanzitutto, chiede conferma del rientro anticipato; dato che Chiara aveva programmato la lavatrice per essere pronta al rientro, Battista l'avvia immediatamente in modo da terminarla per tempo, sia al fine

evitare di disturbare gli occupanti della casa con il rumore che per avere i panni pronti ad essere spostati nell'asciugatrice, senza che si stropiccino nel cestello.
Sulla strada di casa, Chiara decide di fermarsi al supermercato. Accorgendosi che stiamo controllando la lista della spesa, Battista il maggiordomo le suggerisce alcuni alimenti che nel frigorifero *smart* risultano terminati.
Controllando la posizione dell'utente, il maggiordomo virtuale decide quando arriva il momento di iniziare a preparare le stanze principali della casa accendendo il riscaldamento sulla temperatura più adatta, calcolata in modo tale che sia il miglior compromesso fra quella preferita dai membri della famiglia ed il massimo risparmio energetico possibile basato sulla temperatura esterna.
Mentre viene riempita la dispensa, Battista notifica la presenza di alcuni prodotti che potrebbero essere scaduti perché da molto tempo non vengono controllati.
Chiara vuole scaricare la tensione accumulata durante il giorno facendo un bel bagno rilassante. Battista, osservando che i preparativi rientrano in un pattern a lui noto, suggerisce una *playlist* di Spotify, il noto servizio di *streaming* musicale, e allo stesso tempo abbassa le luci del bagno una volta immersi; il relax può essere totale perché, benché sia stato messo a preriscaldare il forno per la cena, si sa che non verrà sprecata energia: Chiara verrà notificata quando la temperatura desiderata sarà quasi raggiunta, senza doversi preoccupare di impostare timer o sveglie. Inoltre il maggiordomo virtuale potrà controllare, attraverso un apposito sensore, che non ci si stia addormentando nella vasca, comportamento pericoloso.
Mentre Chiara cucina, il pannello del TV *smart* viene spento ogni qualvolta non viene guardato direttamente, per poter risparmiare al massimo la corrente elettrica, mentre l'audio continuerà ininterrotto.
Prima di andare a letto, il sistema controlla l'intera casa, notificando eventuali problemi come finestre lasciate aperte per sbaglio - a meno che non vengano esplicitamente indicate come da lasciare aperte - e attiva l'allarme perimetrico per la sicurezza degli occupanti della casa.
Conoscendo l'orario tipico in cui gli abitanti vanno a dormire, ma anche riconoscendo pattern tipici antecedenti al coricarsi, in inverno il letto verrà riscaldato alla temperatura preferita degli occupanti, mentre in estate sarà la temperatura della stanza a venire regolata in maniera tale da garantire un sonno confortevole.
Durante il sonno, il sistema non si spegne, ma monitora costantemente la casa: per esempio, viene controllato il livello di inquinamento nelle stanze, attivando eventualmente il sistema di areazione per far si che l'aria rimanga sempre pulita e respirabile. In caso di imprevisti, l'utente verrà avvisato solo

in caso di problemi non risolvibili: ad esempio, verrà monitorata la temperatura di frigoriferi e freezer in caso di blackout prolungato, ma la notifica all'utente avverrà solo nel momento in cui la differenza rispetto allo standard non sia più tollerabile. Nel caso di problemi come fughe di gas, sarà compito del sistema non solo areare la stanza colpita, ma anche disattivare l'impianto elettrico della stessa per vanificare il rischio di esplosioni dovute a cortocircuiti.
Alle 7:00 in casa di Chiara suona la sveglia. Per partire con il piede giusto, con un apertura graduale delle tapparelle (o, in caso non ci sia sufficiente luce di fuori, un'accensione graduale delle luci) viene simulata l'alba a partire da mezz'ora prima del risveglio, mentre sensori nel letto trovano il momento più adatto per far suonare la sveglia, quello di sonno più leggero. Inoltre la caffettiera elettrica ha già preparato un buon caffé per partire carichi di energia. Per evitar bruschi risvegli in inverno, la temperatura della stanza sarà mite, così come quella del bagno.
Controllando l'agenda personale ed i social network come *Facebook*, il maggiordomo può suggerire gli impegni della giornata, i compleanni prossimi ed è in grado di proporre lui stesso task di manutenzione della casa, come pulizie periodiche, controlli ai sistemi ed altro.
Il maggiordomo virtuale dovrà anche adattare gli ambienti agli utenti della casa. Per esempio, Chiara ha due bambini piccoli che possono infilare le dita nelle prese di corrente: per questo motivo sono automaticamente disattivate tutte quelle in basso nella stanza in cui si trovano, eccetto quelle già in uso e quindi meno pericolose. Il sistema è anche in grado di fungere da *baby monitor*: ogni movimento sospetto del bambino verrà immediatamente notificato al genitore. Alcune stanze potranno, infine, essere interdette al bambino, come il bagno o la dispensa.
Insieme a Chiara vive anche la nonna, che rientra nella categoria di persone anziane o con patologie particolari. Battista può prendersi cura anche di lei, grazie alla funzionalità di promemoria *smart*, non solo inteso come *reminder*, ma perfettamente integrato con i sistemi della casa: Battista è in grado di ricordare se va assunto un certo medicinale ad una determinata ora, la quantità necessaria e il luogo in cui si trova (funzionalità estensibile, ad esempio, tramite una scatola dei medicinali *smart*, in grado di indicarci qual'è esattamente la pillola da prendere). Nel caso il medicinale stia per terminare, Battista potrà avvisare Chiara di occuparsene.
In caso di situazioni di pericolo, Chiara sa che Battista può prendersi cura di ogni abitante della casa, senza esitazioni: ad esempio, oltre al classico sistema per chiamare soccorso in caso di bisogno durante un bagno o una doccia, Battista può accorgersi dell'emergenza e immediatamente svuotare la vasca e/o chiudere l'acqua e/o rinfrescare o riscaldare l'ambiente, per

mettere la persona nelle condizioni di sicurezza migliori, oltre che chiamare soccorso sia all'interno della casa stessa (es. allarme sonoro), che al di fuori (es. ambulanza).

1.2 Introduzione alla domotica

Questo scenario, così futuribile, eppure così attuale, è reso possibile dall'unione di molte discipline in un unica scienza, la **domotica**.
La parola domotica nasce in Francia come *domotique* e deriva dall'unione della parola latina *domus* con *informatique*, informatica in francese (Treccani, 2015).
La domotica è una scienza interdisciplinare, che si occupa dello studio delle tecnologie atte a migliorare la qualità della vita nella casa, o più in generale negli ambienti in cui vivono gli esseri umani. Il fine è proprio quello di studiare metodi per migliorare la qualità della vita e la sicurezza delle persone. Allo stesso tempo, l'integrazione di tecnologia in ambienti familiari ne semplifica l'uso. Il maggior controllo delle strutture permette di ridurne i costi di gestione, mentre non da meno è la possibilità di integrare e rendere maggiormente intelligenti le realtà già esistenti.

1.2.1 Un po' di storia

L'idea di un sistema automatizzato domestico appare già nella letteratura fantascientifica del XX Secolo, per esempio nel racconto di fantascienza di Ray Bradbury "Verranno le dolci piogge" (*There Will Come Soft Rains*) del 1950. Gli strumenti base necessari per realizzare un simile sistema erano già stati brevettati nel 1898 da Nikola Tesla. Egli aveva ideato un sistema per pilotare da remoto veicoli di varia natura (Tesla, 1898), che potrebbe essere tranquillamente alla base di un sistema domotico: dato che sfrutta comuni attuatori, potrebbe essere collegato, ad esempio, ad un rubinetto idraulico per poter far circolare acqua calda in dei termosifoni, eseguendo da remoto il preriscaldamento di un ambiente domestico.
Come spesso accade nella storia, ciò che spinse verso un'idea di domotica simile a quella odierna fu da un lato la necessità, dall'altro la ricerca della comodità. Durante i primi decenni del '900 la servitù domestica vide un notevole declino ed al contempo iniziò a diffondersi la corrente elettrica nelle abitazioni. Comparvero così i primi elettrodomestici: lavatrici, caldaie e scaldabagno, frigoriferi, macchine da cucire arrivarono nelle case dei più ricchi, aprendo prospettive interessanti per la mente dei più inventivi.
Negli anni successivi, numerose Esposizioni Universali (come quella del '34

a Chicago, del '39 a New York e successivamente del '64 sempre a New York) mostrarono il potenziale di case completamente automatizzate (Gerhart, 1999).
Il primo esempio di computer progettato esplicitamente per la domotica risale al 1966 (Spicer, 2000). In quell'anno, l'ingegnere Jim Sutherland creò l'*Electronic Computing Home Operator* o, in breve, ECHO IV.
ECHO IV era un sistema di automazione casalingo, costruito a mano con componenti elettronici riciclati, in grado di aiutare la famiglia nelle faccende domestiche. Era in grado di tener traccia delle finanze familiari, memorizzava ricette, controllava la temperatura della casa, poteva accendere e spegnere gli elettrodomestici ed infine fare previsioni meteo. Tutte queste funzionalità furono implementate nel tempo, grazie alla modularità ed estensibilità del sistema.

Figura 1.1: *Electronic Computing Home Operator* (Spicer, 2000)

Pochi anni dopo questo primo, affascinante esperimento casalingo, nel 1969, la Neiman Marcus, specializzata in oggetti di lusso, inserì nel proprio catalogo il *Kitchen Computer*, un minicomputer della Honeywell inserito in un futuristico tavolino. (Spicer, 2000) Questo calcolatore era in grado di memorizzare ricette e, in un secondo momento, proporre cosa cucinare in base agli ingredienti che si avevano sottomano, in maniera molto simile a quello che Ikea propone nella sua *Concept Kitchen* 2025. (Ikea, 2015)
Con il diffondersi dei microcontrollori, progressivamente più piccoli, potenti ed economici, fu possibile controllare in maniera remota ed intelligente un numero via via maggiore di strumenti e oggetti, con costi sempre più contenuti. Con l'avvicinarsi della fine del secolo, l'argomento divenne piuttosto popolare. Nel 1984 si iniziò a parlare di *smart house*, termine coniato dalla

American Association of Housebuilder (Harper, 2003).
Nel 1998, a Watford in Inghilterra, venne costruita la *INTEGER Millennium House*, rinominata poi nel 2013 in *The Smart Home*, dopo un generale rinnovamento. Si tratta di una casa appositamente studiata per mostrare tecnologie di vario tipo legate alla domotica (International Energy Agency, 1998). Tuttavia, nonostante l'interesse, non ci fu la diffusione sperata di questo tipo di sistemi, che rimasero confinati agli appassionati e ai ricchi. Il motivo era probabilmente da ricercarsi nella mancanza di standard e nell'alto costo iniziale per adottare queste soluzioni.
Al giorno d'oggi l'argomento è più attuale che mai. Secondo ABI Research, nel 2012 sono stati installati un milione e mezzo di sistemi di automazione domestica, numero che nel 2017 dovrebbe superare gli otto milioni (ABI Research, 2012).

1.2.2 Le soluzioni attuali

Sia Apple che Google che Microsoft, i tre *big* dell'informatica di consumo, intuendo il potenziale di questo settore, offrono supporto per sviluppare sistemi domotici, adottando però strategie diverse.

- Da un lato abbiamo Apple, che con il suo *HomeKit* punta alla realizzazione di un framework per comunicare e controllare accessori automatizzati domestici, mettendo l'accento anche sull'interazione vocale mediante la tecnologia di Siri, l'assistente vocale Apple (Apple, 2015).

- Google, che da sempre ama far suoi i progetti che reputa interessanti integrandoli nel suo *Alphabet*, ha invece preferito acquisire Nest, azienda leader nella realizzazione di hardware per l'automazione domestica (Nest Labs, 2014).

- Infine, Microsoft ha adottato un approccio ancora differente, pensando Windows 10 come un sistema progettato per adattarsi ai device più diversi: così è nato Windows 10 IoT Core, per processori RISC, pensato appositamente per l'*Internet of Things*. Microsoft insieme ad Adafruit ha realizzato un apposito kit comprendente un Rasperry Pi 2 e sensori, appositamente pensato per creare oggetti smart da integrare in sistemi della più svariata natura (Microsoft, 2015).

Capitolo 2

L'architettura Butlers

Nel piccolo spaccato di vita quotidiana descritto nell'introduzione, oltre all'intelligenza del maggiordomo virtuale, si nota subito un'importante caratteristica del sistema descritto: i vari componenti, dispositivi, sensori, attuatori ed elettrodomestici, interagiscono fra loro, creando una forte sinergia. Ciò è reso possibile implementando un'architettura come la Butlers Architecture, in grado di rendere omogenei i vari agenti in gioco e farli interagire fra loro in maniera semplice.

L'architettura Butlers (Denti, 2014) è un'architettura a *layer* che permette la realizzazione di sistemi domotici intelligenti. L'idea è quella di unire concetti di domotica, risparmio e gestione dell'energia, sistemi ad agenti intelligenti, tecnologie pervasive e *gamification*. Con *gamification* si intende l'applicazione concetti di *game design* in ambiti diversi da quelli in cui sono nati, per renderli più divertenti e interessanti (Gamification Community, 2012).

Questa architettura si pone vari obbiettivi, permettendo di controllare la casa in ogni suo aspetto: consumo energetico, sicurezza, la programmazione di eventi e degli elettrodomestici, con l'obbiettivo finale del massimo comfort possibile, basato sui gusti personali di ogni utente. Un sistema dunque estremamente personalizzabile, ma che non deve perdere d'occhio l'usabilità e l'attrattiva per l'utente. La *gamification* è intesa proprio per quest'ultimo scopo: permette di interagire con i dispositivi in maniera interessante e innovativa, sfruttando anche le tecnologie pervasive dei moderni dispositivi *smart*, in modo da intrattenere l'utente durante la configurazione e, più in generale, l'interazione con il sistema.

2.1 Perché un maggiordomo virtuale?

Ci sono almeno due motivi per scegliere di adottare un sistema domotico che implementi l'architettura Butlers.

Innanzitutto, il sistema deve prendersi cura degli abitanti della casa. Deve essere in grado di conoscere le preferenze degli utenti, come le loro priorità e loro abitudini. L'obbiettivo è quello di **anticipare le decisioni e i bisogni dell'utente**, esattamente come dovrebbe fare un maggiordomo in carne ed ossa. La qualità della vita ne potrebbe giovare in vari aspetti:

- Miglioramento del **comfort** per l'utente: per esempio ci si troverebbe la casa alla temperatura ideale secondo le proprie preferenze, anche in caso di rientro anticipato.
- Miglioramento della **sicurezza**: per esempio non ci si potrebbe dimenticare di attivare l'allarme, verrebbero mantenuti controllati i livelli di inquinamento ambientale e rilevate eventuali fughe di gas non solo attraverso un allarme sonoro passivo, ma anche in modo attivo attraverso la ventilazione automatica della stanza e la disattivazione dell'impianto elettrico nella stessa.
- **Risparmio di energia** e di denaro, grazie alla gestione intelligente dei consumi: per esempio, programmazione dell'utilizzo degli elettrodomestici più *energy-demanding* nella fascia oraria in cui l'elettricità costa meno.

Il secondo aspetto interessante per l'adozione di una figura antropomorfa riguarda la ***user-friendliness***. Innanzitutto, poter interagire in maniera naturale attraverso messaggi e social network con l'entità che governa la casa rende possibile l'uso del sistema anche a chi non è particolarmente avvezzo alla tecnologia. Inoltre, è chiaro che il successo e la diffusione di una tecnologia è strettamente legato all'intrattenimento e all'appagamento che è in grado di fornire.
Se un sistema con un'interfaccia piena di menu e sottomenu può sicuramente gratificare l'utente più smanettone, alla ricerca del dettaglio tecnico da personalizzare, l'utente comune è alla ricerca del dettaglio estetico più appagante, dell'effetto *wow* dovuto magari al proprio maggiordomo personalizzato, in grado di rispondere alle nostre domande in linguaggio naturale con la personalità che abbiamo scelto per lui.

2.2 Requisiti

La Butlers architecture sviluppa la propria architettura a livelli su requisiti ben delineati. Vengono definiti sei domini (Denti, 2014):

1. i requisiti architetturali **generali**;
2. i requisiti di **coordinazione**;
3. i requisiti di **configurazione**;
4. i requisiti per l'**interfaccia utente**;
5. i requisiti **funzionali**;
6. i requisiti per la ***gamification***.

2.2.1 Requisiti generali

I requisiti base delineati per l'architettura sono riassumibili in:

- deve essere presente un **sensore di consumo** per ogni dispositivo.
- i sensori devono essere in grado di **comunicare** fra loro e con un *hub* centrale; la tecnologia scelta (*wireless* o *wired*, Bluetooth piuttosto che ZigBee o Ethernet) non dev'essere vincolante.
- deve essere presente un **coordinatore**, in grado di ottenere le informazioni dai suddetti sensori e garantire che vengano rispettare le leggi che governano il sistema e le preferenze degli utenti.

2.2.2 Requisiti di coordinazione

Il coordinatore ha un ruolo centrale nell'architettura Butlers. I requisiti per la coordinazione riguardano molti aspetti del sistema, da problemi architetturali e di interoperabilità, alla malleabilità e ispezionabilità delle *policy*, alla sicurezza e ad un appropriato supporto per l'intelligenza.

- il coordinatore deve essere progettato in maniera da **evitare colli di bottiglia**: dovrebbe essere centralizzato da un punto di vista logico, ma fisicamente il più distribuito possibile;
- il coordinatore deve essere basato su **protocolli di comunicazione aperti**, in modo tale da poter potenzialmente interagire con dispositivi di ogni marca;

- il coordinatore deve essere **accessibile** e facilmente configurabile da utenti non esperti, anche da remoto;

- il coordinatore deve supportare **politiche di coordinazione definibili e modificabili, anche in maniera dinamica, dall'utente**: le *policy* dovrebbero essere facilmente riviste, integrate e sostituite in ogni momento, anche da remoto; inoltre il sistema non dovrebbe strettamente legato a nessuna tecnologia particolare, né avere delle preconfigurazioni non modificabili dall'utente.

- il coordinatore dovrebbe essere in grado di **migliorare se stesso** interagendo con altri coordinatori, che presidiano altre case, condividendo esperienze e conoscenze.

- il coordinatore deve essere in grado di sfruttare le tecnologie di geolocalizzazione dei dispositivi mobili per **dedurre le intenzioni** degli utenti (es. nello scenario del paragrafo 1.1, il sistema è cosciente del fatto che Chiara sta tornando in anticipo).

- il coordinatore deve avere un'intelligenza tale da essere in grado di elaborare un **piano di azione** basandosi sulle priorità dell'utente, sugli obbiettivi cardinali del sistema, ma anche sullo stato attuale di comunicazione e coordinazione (Papadopoulos and Arab, 1998) (Omicini and Papadopoulos, 2001). Questo è il tipico scenario in cui le tecnologie e infrastrutture multi agente danno del loro meglio, quando l'autonomia degli agenti e i *tool* di coordinazione funzionano in maniera sinergica (Ciancarini et al., 1999).

- per supportare l'implementazione dei servizi più avanzati, il coordinatore deve essere in grado di **interagire con** i principali ***social network***, direttamente o via *proxy* realizzati ad-hoc.

2.2.3 Requisiti di configurazione

Anche il processo di configurazione, come già accennato in precedenza, ha requisiti ben specifici.

- la **configurazione** del sistema deve essere **semplice**. Può essere ad esempio guidata, sotto forma di *wizard*, e progettata accuratamente in modo da non essere noiosa o troppo lunga; a tal fine, può anche sfruttare i dispositivi già in possesso dell'utente, come *smartphone*, *tablet*, *smart* TV o console per videogiochi.

- la procedura di configurazione locale dovrebbe sfruttare i suddetti dispositivi per essere il più possibile familiare all'utente; è possibile anche integrare un sistema di configurazione vocale, mentre sensori di presenza possono essere usati per attivare autonomamente i dispositivi più vicini con cui interagire.

- la procedura di configurazione remota, invece, sfruttando ad esempio *smartphone* e *tablet* potrebbe utilizzare altri canali come SMS e *instant messaging*.

- il linguaggio con cui viene configurato il sistema, inteso come il set di comandi da usare per controllare il processo di configurazione, deve essere **espressivo ed efficiente**: abbastanza flessibile da supportare un'ampia varietà di dispositivi, elettrodomestici e *policy*, permettendo di utilizzare praticamente qualunque comando mantenendo però contemporaneamente facilità d'uso e semplicità; allo stesso tempo, dovrebbe essere disaccoppiato dal linguaggio dell'utente: il sistema deve essere in grado di supportare utenti di diversa nazionalità anche simultaneamente, senza che questo causi problemi.

2.2.4 Requisiti per l'interfaccia utente

Al fine di soddisfare i requisiti del paragrafo 2.2.3 e realizzare lo scenario ideale descritto nel capitolo 1, anche l'interfaccia utente deve essere modellata partendo da requisiti ben definiti.

- deve supportare sia l'**accesso locale**, sfruttando i *device* disponibili nella casa dell'utente come *smart* TV, schermi *touch* e sistemi di interazione vocale, che l'**accesso remoto**, tramite SMS, applicazioni per dispositivi mobili, siti web, ecc.

- dal punto di vista concettuale, deve essere in grado di rendere **facile e naturale** per l'utente **specificare preferenze e impartire comandi** anche molto complessi; se ben sviluppato, questo requisito porta vantaggi non solo agli utenti, ma è anche necessario al fine di fornire un supporto di coordinazione con i gradi di libertà auspicabili per la Butlers architecture.

- deve permettere di specificare più obbiettivi simultaneamente, accettare specifiche del tipo "fai la lavatrice entro oggi, ma risparmia il più possibile energia" oppure "fai queste cose insieme, ma non causare *blackout*" ma esprimendole in maniera chiara e comprensibile, in modo che l'utente possa apprenderle rapidamente.

2.2.5 Requisiti funzionali

Anche se molti requisiti funzionali sono stati già indirettamente menzionati nei precedenti paragrafi, si riassumono di seguito al fine di fornire una visione completa.

- le parti del sistema complementari all'interfaccia utente (*view*), ovvero *model* e *controller*, devono supportare l'espressione di goal ad alto livello, dei desideri e delle preferenze, oltre ad ogni altro comando e sequenza di comandi, **a livello del linguaggio**, includendo priorità, vincoli definiti dall'utente, ecc. È necessario dunque adottare un linguaggio espressivo e facile da utilizzare, sfruttabile con interfacce anche molto differenti: a partire da quelle basate sul testo, passando per *touch screen* e sistemi vocali, concludendo con *social network* come Facebook o Twitter.

- dev'essere supportata la **riconfigurazione dinamica del sistema** in ogni suo aspetto, in ogni momento e da ogni luogo, sia localmente che da remoto, sia per mezzo di comandi espliciti dell'utente che in base ad una scelta autonoma effettuata del sistema.

- dev'esserci la possibilità di specificare più obbiettivi simultanei (come già spiegato nel paragrafo 2.2.4); pertanto deve esistere un **sistema di individuazione e risoluzione dei conflitti**, in grado di elaborare un piano di azione a partire dagli obbiettivi specifici dell'utente, dalle *policy* globali, dalle priorità e dai vincoli specificati.

- le **informazioni geografiche** devono essere **gestite e sfruttate adeguatamente**: la posizione dell'utente fornita esplicitamente (es. servizi di geolocalizzazione dello *smartphone*) o indirettamente (es. informazioni ottenute da post su social network, ecc.) può essere usata sia immediatamente per prendere decisioni (es. rientro anticipato), che memorizzata per determinare pattern comportamentali e altre informazioni sull'utente.

- un'**intelligenza artificiale**, realizzata per esempio da un apposito agente, oltre che prendere in carico il suddetto punto, può essere sviluppata per essere in grado di anticipare le esigenze e i desideri dell'utente, suggerendo *task* e azioni che vanno oltre al piano specificato esplicitamente dall'utente.

2.2.6 Requisiti per la *gamification*

Come già accennato nell'introduzione a questo capitolo, l'esperienza utente è fondamentale per il successo del sistema, sia a livello commerciale che a livello di soddisfazione dell'utente. Si vanno quindi a specificare dei requisiti appositi per la *gamification*.

- il sistema deve promuovere la **condivisione degli obbiettivi** raggiunti attraverso i *social network*.
- il sistema deve promuovere l'utilizzo di **comunità dedicate** agli utenti, per ottenere consigli, suggerimenti, *best practices*, ecc.
- il sistema deve puntare ad avere un'**interazione** con l'utente il più **naturale** possibile, attraverso ad esempio *avatar* virtuali; gli obbiettivi sono di rendere più facile accettare il sistema pervasivo, incrementare l'adozione volontaria della tecnologia ed, infine, aumentare l'interesse verso l'utilizzo del sistema.

2.3 L'architettura Butlers: i livelli

L'architettura Butlers è strutturata in sette livelli (vedi figura 2.1). Ciascun *layer* si occupa di un differente aspetto del sistema domotico. Nella progettazione del sistema si è tuttavia liberi di scegliere quali livelli implementare e, quindi, su quali caratteristiche si vorrà mettere l'accento: l'adozione dei singoli *layer* è facoltativa, eccezion fatta per il primo, obbligatorio in quanto abilitante.
Come è possibile osservare in figura 2.1, ciascun livello logico, per poter funzionare e quindi offrire all'utente i suoi vantaggi, ha bisogno di alcuni requisiti tecnologici: un vantaggio dell'architettura *Butlers* è che per soddisfarli non sono necessari dispositivi costosi: bastano infatti pochi sensori e attuatori ed un calcolatore su cui eseguire il nodo TuCSoN, mentre grazie al potenziale fornito dai *transducer* qualunque dispositivo già in possesso dell'utente, con piccole modifiche può essere integrato nel sistema, anche se non è stato appositamente pensato per esso.

2.3.1 I sette livelli

- Layer 1 - **Information**: è il livello abilitante, pertanto necessario al funzionamento della maggior parte dei sistemi. Permette al sistema

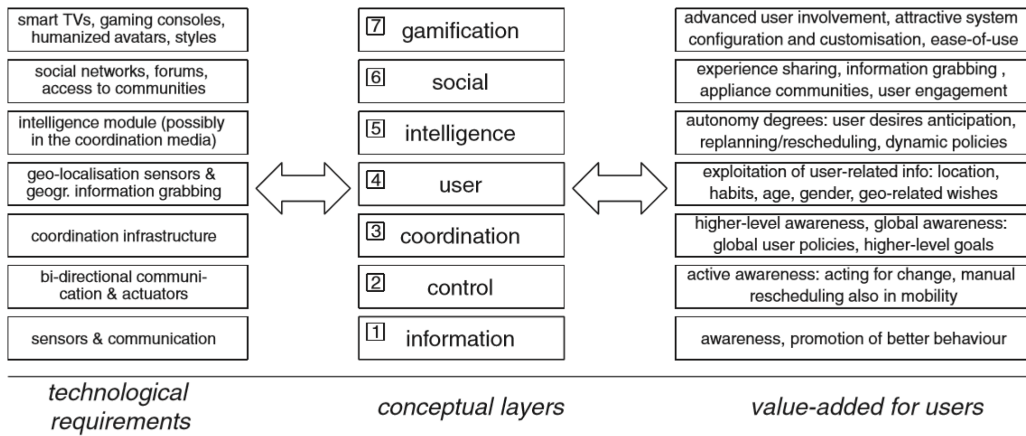

Figura 2.1: I *layer* della Butlers Architecture (Denti, 2014)

(e di conseguenza all'utente che lo utilizza) di conoscere lo stato dei dispositivi, per esempio il consumo attuale di un elettrodomestico.

- Layer 2 - **Control**: fornisce al sistema la possibilità di intervenire nella realtà, permettendogli di adattare la casa alle esigenze dell'utente. Per esempio, permette di interrompere il funzionamento controllato di un'elettrodomestico che sta consumando troppa energia, o di regolare la temperatura di una stanza. È il livello fondamentale per qualunque sistema che voglia andare oltre il semplice monitoraggio dello stato dell'abitazione.

- Layer 3 - **Coordination**: permette di coordinare le varie entità, pertanto rispetto ai livelli 1 e 2 rende possibile implementare e attuare politiche di gestione più complesse e personalizzate.

- Layer 4 - **User**: con questo livello l'utente entra a far parte del sistema, con il suo stato attuale (ad esempio, la sua posizione, ma anche il suo stato di salute, l'età anagrafica, le eventuali disabilità o restrizioni, ecc.), le sue preferenze e il suo profilo.

- Layer 5 - **Intelligence**: il sistema che adotta questo livello è un sistema intelligente, in grado di elaborare le informazioni con vari livelli di complessità. Si parte dal semplice riconoscimento di pattern comportamentali, fino a giungere all'elaborazione di proposte pertinenti alle preferenze dell'utente in maniera del tutto autonoma.

- Layer 6 - **Social**: aggiunge al sistema la possibilità di condividere informazioni, al fine di migliorare se stesso. Per esempio, potrebbe condividere i dati sui consumi energetici in modo da ottenere suggerimenti su come ottimizzarli.

- Layer 7 - **Gamification**: aggiunge al sistema quelle feature, di cui si è già discusso all'inizio di questo capitolo, per cui approcciarsi al sistema diventa divertente e interessante.

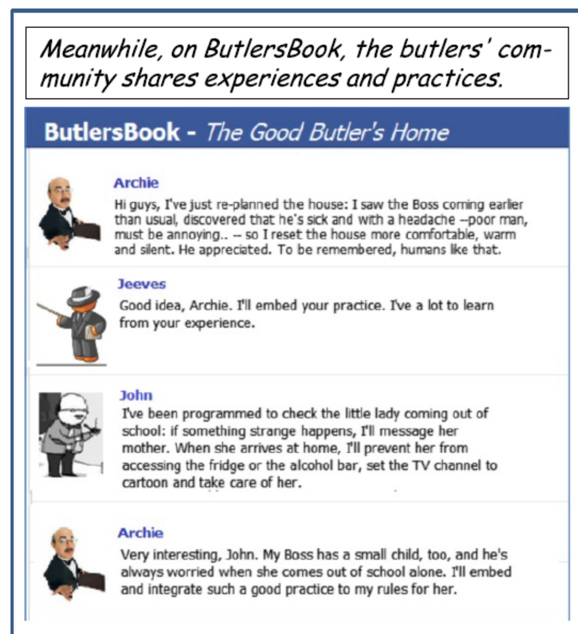

Figura 2.2: I maggiordomi virtuali della Butlers Architecture interagiscono condividendo informazioni su *social network*. (Denti, 2014)

2.3.2 Alcune possibili configurazioni

Come già spiegato nella sezione 2.3, la Butlers Architecture può essere implementata anche solo in parte. Si descrivono quindi alcune possibili configurazioni del sistema, andando ad osservare quali sono le potenzialità che si possono ottenere aggiungendo livello per livello.
Nel paragrafo successivo si andrà invece ad analizzare quanto di questa architettura è oggi presente in Home Manager.

Il sistema informatore

È un sistema che implementa solo il *layer* 1.
Alcuni dei sistemi domotici attuali sono realizzati in questo modo: forniscono informazioni sullo stato del sistema (anche fruibili da remoto, come lo stato dell'allarme o la temperatura ambientale), però non sono in grado di modificarne i parametri di funzionamento, se non in maniera marginale.

Il sistema "telecomando"

Questo sistema implementa i livelli 1 e 2.
È in grado di fornire informazioni sul sistema e allo stesso tempo intervenire direttamente. Molti sistemi domotici funzionano in questa maniera, basti pensare al termostato intelligente di cui si è accennato nel capitolo 1: permette da un lato di controllare la temperatura di una stanza e dall'altro modificarla da remoto.

Il sistema "telecomando" evoluto

È possibile estendere il sistema precedente perché sia maggiormente social, o adotti aspetti della *gamification*. Pertanto, può facilmente integrare dei concetti dai livelli 6 e 7.

Il maggiordomo basico

Estendendo il sistema "telecomando" con i concetti del livello 3, ecco che abbiamo finalmente un primo sistema che ha le caratteristiche che si desiderano in un sistema che adotti la Butlers Architecture: i livelli dall'1 al 3 è dunque il set di livelli minimo richiesto per il sistema voluto.
Un sistema dotato del livello *coordination* è in grado di gestire e, appunto, coordinare le varie entità del sistema, permettendo l'adozione di politiche maggiormente complesse come per esempio controllare che i consumi non superino una certa soglia, seppur mantenendo la temperatura ambientale desiderata e completando la lavatrice prima delle 19:00.

Il maggiordomo *user-aware*

Andando ad inserire l'utente nello scenario precedente, dunque adottando il livello 4, è possibile rendere il maggiordomo *user-aware*, ovvero far sì che il comportamento del sistema vari in base alle preferenze dell'utente, alla sua età, alle sue abitudini, ecc. Questo maggiordomo può usufruire anche dei

servizi del livello 6, per andare ad estrapolare informazioni sull'utente dai *social network* che frequenta.

Il maggiordomo intelligente

Implementando il livello 5 della Butlers Architecture si rende il sistema intelligente, capace di anticipare l'utente nelle sue decisioni e nei suoi bisogni e di sfruttare ogni informazione in suo possesso, ottenuta e rielaborata sostanzialmente da ogni fonte.
L'intelligenza può anche andare a toccare aspetti linguistici, per supportare concetti di alto livello, e pone le basi per il livello 7.

2.3.3 Butlers Architecture e Home Manager

Home Manager, come sarà spiegato nel capitolo 4, è stato recentemente esteso per integrare l'architettura Butlers. Al giorno d'oggi, Home Manager non integra ancora perfettamente la Butlers Architecture e analizzando il prototipo possiamo asserire che:

- il sistema è in grado di monitorare lo stato della casa, benché solo in maniera simulata, pertanto il *layer* 1 è presente nel prototipo di Home Manager. Per completare l'adozione di questo livello è opportuno introdurre nel sistema il concetto di *transducer* (si vedano i paragrafi 3.3 e 4.2.2 per maggiori dettagli).

- il sistema è in grado di modificare direttamente il sistema, anche in questo caso sul modello simulato, pertanto il livello 2 è presente nel prototipo. Come per il *layer* 1, per completare l'adozione di questo livello è opportuno introdurre il concetto di *transducer*.

- Home Manager adotta i *tuple centre* di TuCSoN, pertanto è presente uno strumento di coordinazione molto potente: il livello 3 è dunque fortemente presente nel prototipo.

- il sistema conosce le preferenze degli utenti della casa, il loro stato attuale e programma il piano di azione in base a queste informazioni: pertanto il livello 4 è implementato in Home Manager.

- Home Manager ha alcuni algoritmi di risoluzione dei conflitti, è in grado di prendere decisioni di aprire e chiudere le tapparelle in base all'orario di alba e tramonto (Celi, 2015), ma non ha attualmente una vera e propria intelligenza. Dunque il livello 5 non è presente nel prototipo.

- recentemente Home Manager è stato esteso per poter accedere ai *social network*, grazie ad appositi agenti (Bevilacqua, 2015). Il *layer* 6 è pertanto presente, almeno in parte, nel prototipo attuale.

- Home Manager non ha, al momento, elementi dal livello 7: non è ancora stato pensato per integrare i concetti della *gamification*.

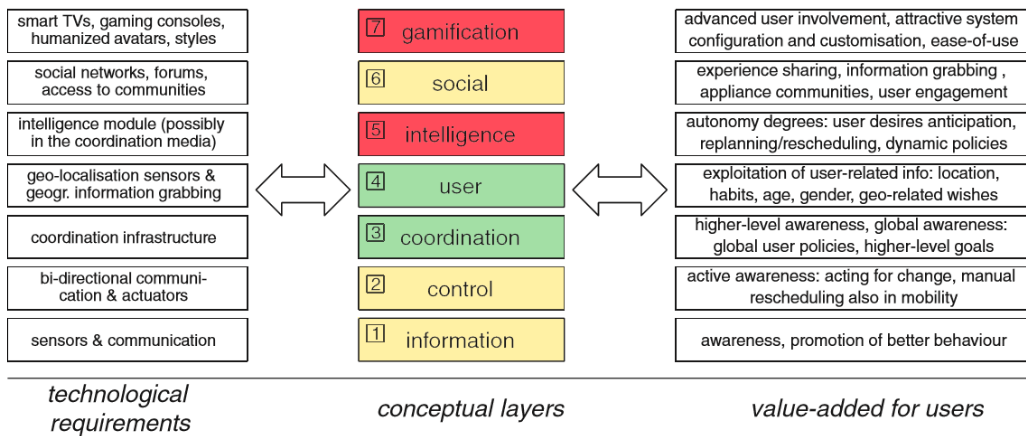

Figura 2.3: I livelli della Butlers Architecture integrati in Home Manager: in verde, quelli fortemente presenti, in rosso, quelli assenti, in giallo quelli su cui vi è margine di miglioramento.

Capitolo 3

TuCSoN

TuCSoN (*Tuple Centres Spread over the Network*) nasce come libreria Java per fornire un supporto alla coordinazione per agenti Java e tuProlog (TuCSoN, 2015).
Si tratta, in sostanza, di un *middleware* distribuito, disponibile con licenza GNU LGPL e si basa sul modello per la coordinazione di processi distribuiti omonimo: il **modello TuCSoN**, usando *tuple centre* come media di coordinazione, è in grado di coordinare agenti autonomi, intelligenti e mobili. I *tuple centre* sono spazi di tuple potenziati, in cui è possibile, attraverso linguaggio di programmazione ReSpecT, andare a inserire delle specifiche comportamentali per le tuple, sotto forma di reazioni a determinati eventi.

3.1 Il modello e l'infrastruttura TuCSoN

Come già accennato, la tecnologia abilitante per TuCSoN sono i ***tuple centre***. Essi rappresentano la struttura con cui viene organizzato il sistema e forniscono il supporto necessario per governare l'interazione.
In estrema sintesi i *tuple centre* sono degli spazi di tuple potenziati. Uno spazio di tuple può essere visto come uno spazio di memoria condivisa fra vari processi, che comunicano tra di loro mediante **tuple**, che vengono emesse, lette e rimosse attraverso apposite primitive. Una tupla è una collezione di elementi informativi ordinati, possibilmente eterogenei. Ciò che distingue un *tuple centre* da un *tuple space* è la possibilità per il primo di specificare il comportamento che deve avere il centro di tuple in risposta a determinate interazioni, pertanto è possibile specificare delle *reaction*, delle reazioni a determinati eventi. Il linguaggio con cui vengono specificate queste reazioni è ReSpecT.
Un sistema TuCSoN è costituito da nodi distribuiti in rete, che comunicano

```
philosopher(I,J) :-
    think,                        % thinking
    table ? in(chops(I,J)),       % waiting to eat
    eat,                          % eating
    table ? out(chops(I,J)),      % waiting to think
!,  philosopher(I,J).

reaction( out(chops(C1,C2)), (operation, completion), (      % (1)
    in(chops(C1,C2)), out(chop(C1)), out(chop(C2)) )).
reaction( in(chops(C1,C2)), (operation, invocation), (       % (2)
    out(required(C1,C2)) )).
reaction( in(chops(C1,C2)), (operation, completion), (       % (3)
    in(required(C1,C2)) )).
reaction( out(required(C1,C2)), internal, (                  % (4)
    in(chop(C1)), in(chop(C2)),  out(chops(C1,C2)) )).
reaction( out(chop(C)), internal, (                          % (5)
    rd(required(C,C2)), in(chop(C)), in(chop(C2)),
    out(chops(C,C2)) )).
reaction( out(chop(C)), internal, (                          % (5')
    rd(required(C1,C)), in(chop(C1)), in(chop(C)),
    out(chops(C1,C)) )).
```

Figura 3.1: *Dining philosophers* in TuCSoN, con le reazioni specificate in ReSPeCT (Omicini, 2012).

e cooperano fra di loro. Gli agenti, che, in parole povere, altro non sono che dei processi, sono eseguiti sui nodi. Il comportamento del sistema è definito all'interno dei *tuple centre*: essendo dei media programmabili tramite ReSpecT, che è un linguaggio Turing completo, è possibile specificare praticamente qualunque legge di coordinazione: è possibile anche risolvere problemi complessi come quello dei *dining philosophers* in maniera elegante e concisa, senza bisogno di toccare in nessun modo il codice degli agenti da coordinare, come mostrato in figura 3.1.
ReSpecT supporta numerosi tipi di specifiche: sensibili al contesto, al tempo, supporta sensori e attuatori mediante l'astrazione dei *transducer*, argomento di cui si tratterà più approfonditamente nel paragrafo 3.3.
Un altro vantaggio dei *tuple centre* è quello di essere ispezionabili in qualunque momento, sia dagli agenti che dagli utenti: questo si traduce nella possibilità di conoscere precisamente lo stato del sistema in ogni momento, comprese le leggi che lo governano; inoltre forniscono i mezzi per cambiare queste *policy* in ogni momento (ovviamente possedendo i giusti permessi), sia in maniera statica che dinamicamente.
I *tuple centre* sono in grado di cooperare, attraverso quella che viene definita la proprietà di *linkability*, ovvero la capacità di un *tuple centre* di andare ad agire su un altro *tuple centre*.
La sicurezza è garantita da un modello basato su ruoli, chiamato **RBAC**

(*Role Based Access Control*). Ogni agente è in grado di negoziare i propri ruoli, e di conseguenza i propri permessi, sia in maniera statica, nel momento in cui si unisce alla comunità, sia in maniera dinamica, nel momento del bisogno (*by need*). Gli agenti sono tenuti a interagire con l'infrastruttura attraverso una sorta di *gateway proxy*, definito ***Agent Coordination Context*** (ACC), che filtra le operazioni che l'agente è in grado di eseguire, in base a ruoli e diritti. RBAC consente di progettare il sistema con granularità differenti a seconda delle esigenze.

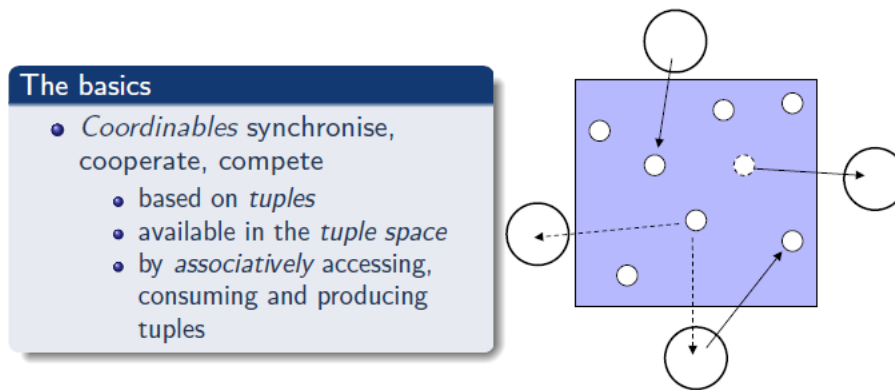

Figura 3.2: Il meta-modello dei *tuple space* (Omicini, 2012).

3.2 Perchè scegliere TuCSoN

Esistono varie motivazioni per cui TuCSoN è una tecnologia adatta allo sviluppo di un sistema che implementi l'architettura Butlers.

- Innanzitutto, TuCSoN fornisce un modello organizzativo gerarchico completo, e implementa un modello di sicurezza basato sui ruoli (si vedano i requisiti di coordinazione nel paragrafo 2.2.2).
- Supporta intrinsecamente l'interoperabilità fra agenti eterogenei (si vedano i requisiti generali descritti nel paragrafo 2.2.1).
- I *tuple centre*, in quanto media di coordinazione programmabile con un linguaggio touring-completo, permette di relizzare un sistema intelligente (si vedano i requisiti funzionali descritti nel paragrafo 2.2.5).
- I *tuple centre* forniscono politiche definibili in maniera dinamica dall'utente, e sono inoltre ispezionabili in ogni momento, per conoscere lo stato della comunicazione e della coordinazione (si vedano i requisiti

di configurazione nel paragrafo 2.2.2 ma anche i requisiti funzionali nel 2.2.5).

- I *tuple centre* sono accessibili e configurabili sia localmente che da remoto, a livello di infrastruttura. Inoltre né il modello, né l'infrastruttura costringono all'uso di una specifica tecnologia, né costringono a configurazioni predefinite non modificabili (si vedano i requisiti di coordinazione nel paragrafo 2.2.2).
- TuCSoN è specificatamente progettato per evitare i colli di bottiglia (si vedano i requisiti di coordinazione nel paragrafo 2.2.2).

D'altro canto vi sono anche alcuni possibili svantaggi. Innanzitutto, al momento l'accessibilità e la configurabilità a livello di infrastruttura non sono semplici da utilizzare, e per questo motivo sono funzionalità sfruttabili solo da utenti esperti, sufficientemente familiari con la struttura di TuCSoN ed i linguaggi utilizzati. Questo problema può essere facilmente aggirato aggiungendo un ulteriore *layer* che traduca le scelte di configurazione dell'utente fatte in un linguaggio *user-friendly*.
Un'altro svantaggio riguarda le performance. Benché TuCSoN si comporti bene nelle applicazioni in cui è stato utilizzato fino ad oggi, situazioni maggiormente stressanti devono ancora essere testate.

3.3 I transducer

Come già accennato nel paragrafo 3.1, i *transducer* sono un componente fondamentale per realizzare dei sistemi situati sfruttando i *tuple centre* e, più in generale, la tecnologia TuCSoN.
Analogamente agli ACC per gli agenti, i *transducer* sono i componenti architetturali che si occupano di rappresentare e mediare cambiamenti ambientali riguardante *probes*. Un *probe* può essere un sensore (ad esempio un termometro), un attuatore (ad esempio un relè che comanda l'accensione di una lampada) o un qualunque altro dispositivo si possa essere interessati ad integrare nel sistema. Ciascun *probe* è assegnato ad un *transducer* specializzato per gestire eventi da/verso questo specifico *probe*: ci saranno pertanto dei *transducer* attuatori, dei *transducer* sensori e così via. Ogni *transducer* può gestire anche più di un *probe* tramite un'apposita procedura di registrazione.
Possiamo anche dire che i *transducer* traducono cambiamenti di proprietà dei *probe* in eventi, che verranno poi gestiti dai tuple centre, e viceversa eventi del sistema in cambiamenti di proprietà da essere inviati ai *probe* (Mariani and Omicini, 2014).

L'utilizzo dei *transducer* porta numerosi vantaggi. Sostanzialmente si sposta la coordinazione dell'interazione fra gli agenti e l'ambiente nel media di coordinazione - che è, ovviamente, anche il punto in cui dovrebbe trovarsi. In questo modo si separano i problemi implementativi da quelli coordinativi e si ottiene un sistema maggiormente ispezionabile, con componenti sostituibili e manutenibili. In particolare sulla sostituibilità va notato che grazie all'uso di questa tecnologia, nel momento in cui si decida di sostituire un *probe* con un altro, è possibile lasciare assolutamente inalterato il codice del programma e andare a sostituire unicamente il *probe*. Verrà fatto un esempio pratico nel capitolo 6.
Dal punto di vista dell'interazione, i *transducer* possono operare sia in maniera sincrona che asincrona: a questo proposito, si mostrano di seguito, nelle figure 3.3, 3.4, 3.5 e 3.6, alcuni esempi.

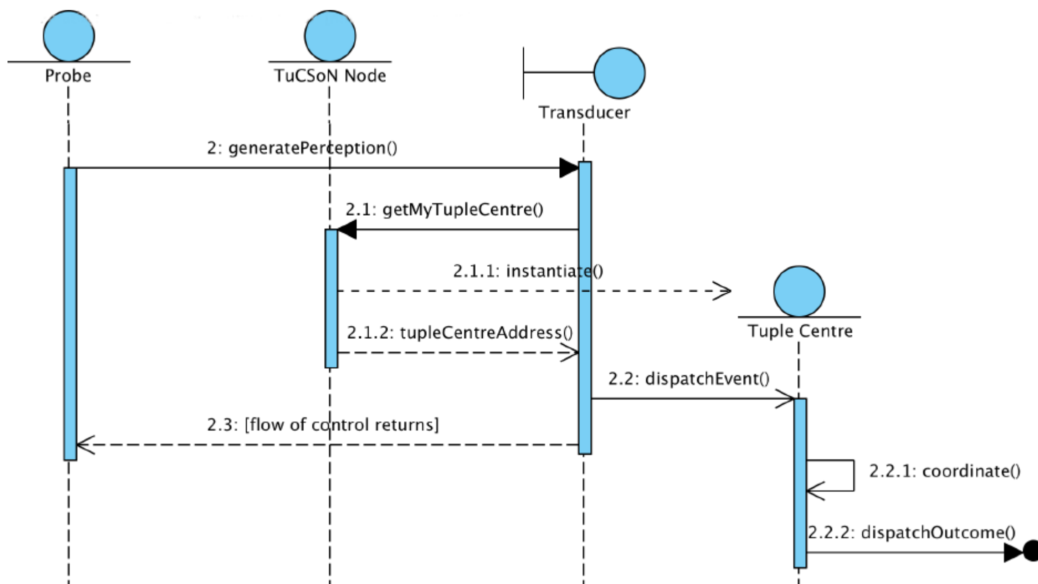

Figura 3.3: Interazione asincrona con un *probe* sensore. Il cambiamento di stato del sensore fa sì che il *tranducer* generi una tupla evento (Mariani and Omicini, 2014).

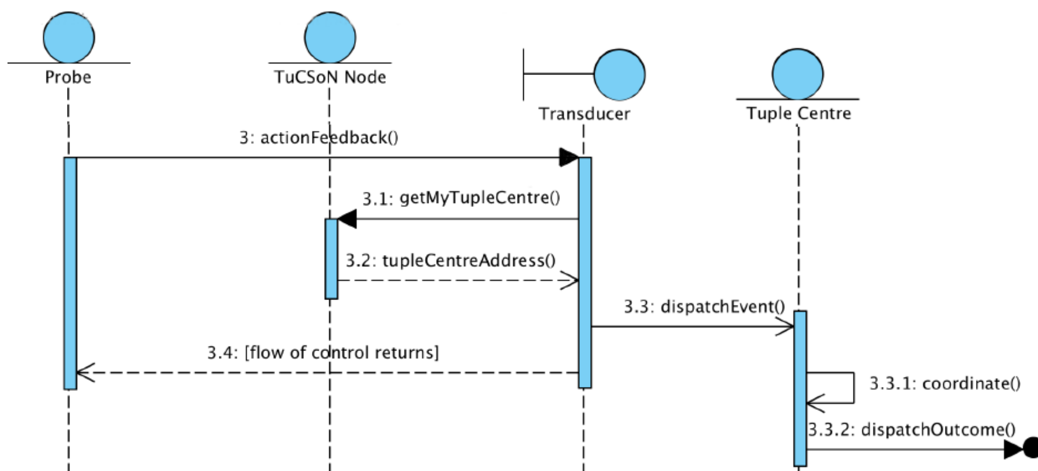

Figura 3.4: Interazione asincrona con un *probe* attuatore. In questo caso il cambiamento di stato è dato dal *feedback* di un'azione effettuata dall'attuatore (Mariani and Omicini, 2014).

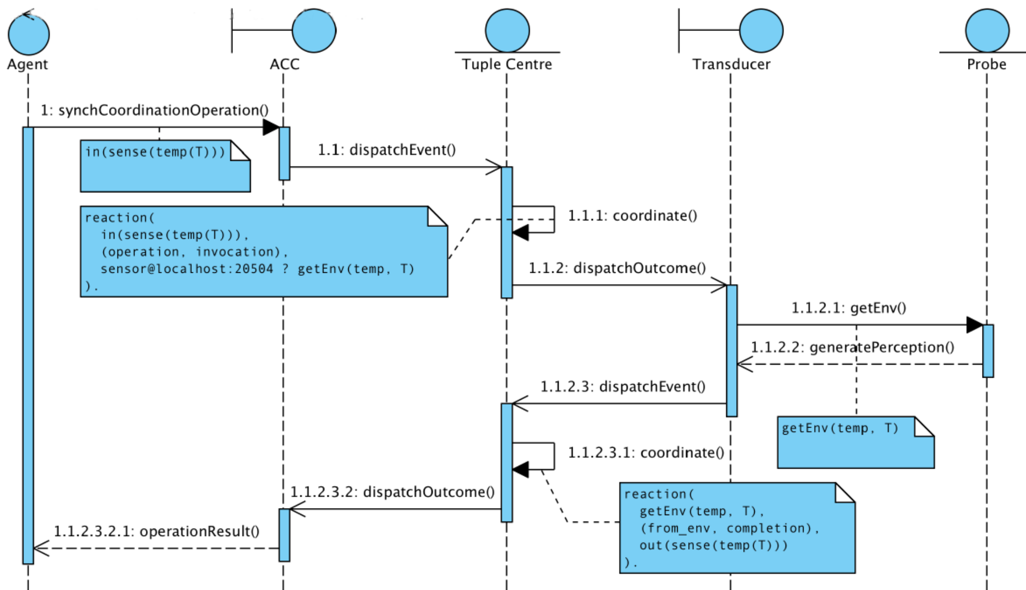

Figura 3.5: Interrogazione sincrona di un sensore. ReSpecT gioca un ruolo fondamentale nel coordinare correttamente le operazioni (Mariani and Omicini, 2014).

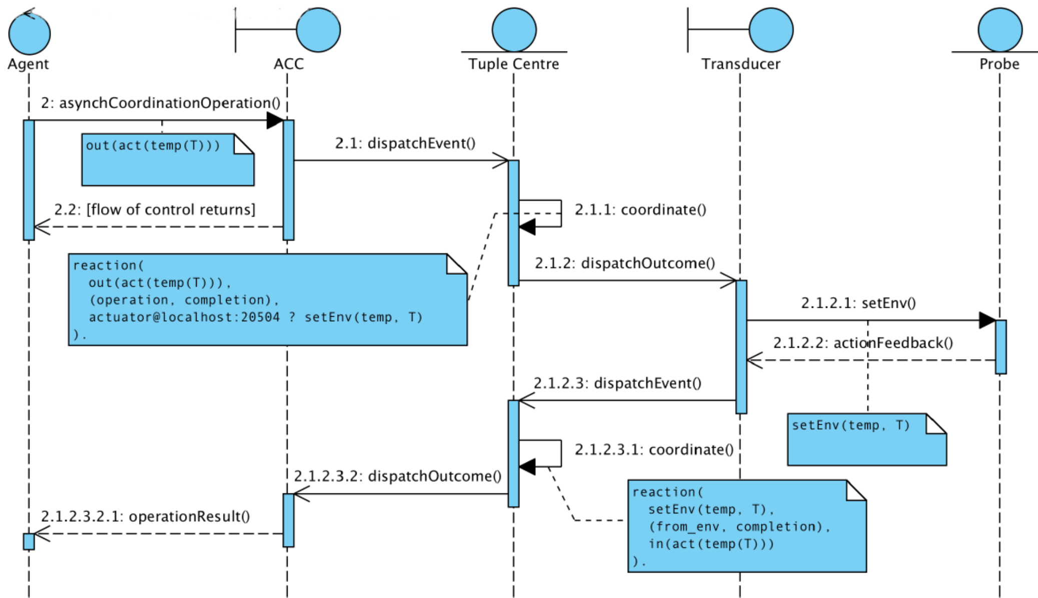

Figura 3.6: Operazione asincrona su un attuatore. Come nella figura 3.5 ReSpecT gioca un ruolo fondamentale nel coordinare correttamente le operazioni (Mariani and Omicini, 2014).

Capitolo 4

Home Manager

Home Manager è un applicazione prototipale per la gestione di una casa intelligente (Calegari and Denti, 2014).
È un sistema multiagente, progettato attraverso SODA (*Societies in Open and Distributed Agent spaces*), una metodologia per l'analisi e la progettazione di sistemi complessi basati su agenti (SODA, 2012).
Alla base di Home Manager vi è l'infrastruttura di coordinazione TuCSoN, che è stata descritta nel capitolo 3.
Il progetto, sviluppato inizialmente nel 2009, è stato recentemente esteso per abbracciare l'architettura Butlers, di cui si è parlato nel capitolo 2: l'idea è quella di giungere ad un sistema in grado di anticipare i bisogni degli utenti e prendere decisioni autonomamente (almeno in parte).
Home Manager considera ciascun dispositivo della casa (es. elettrodomestici, sistema di illuminazione, allarme, ecc.) come un agente che partecipa ad una società. L'infrastruttura di coordinazione programmabile, costituita dai centri di tuple, incorpora le leggi di coordinazione necessarie sia per soddisfare le diverse richieste degli utenti, che per raggiungere gli obbiettivi globali del sistema, scelti dal progettista o dall'amministratore, ad esempio limitare al massimo il consumo di energia.

4.1 Il prototipo attuale

Nel momento in cui si scrive, il prototipo di Home Manager abbraccia diversi aspetti della *Butlers Architecture.* È infatti in grado di:

- Gestire tipi diversi di utenti: amministratori, ordinari e visitatori; ogni utente può avere accesso all'intero sistema o ad una parte di esso, secondo diritti personalizzabili;

- Assegnare preferenze ai vari utenti (ad esempio, la temperatura preferita in una determinata stanza);
- Assegnare *policy* per la gestione personalizzata del sistema (ad esempio, è possibile scegliere se dare la priorità al risparmio energetico e alle preferenza degli utenti);
- Eseguire comandi manuali specifici per le varie stanze e dispositivi;
- Interfacciarsi con *social network* (Twitter);
- Ottenere le informazioni meteo a partire da *web services* dedicati, e utilizzarle, ad esempio, per aprire e chiudere le tapparelle in base all'ora di alba e tramonto.

Per maggiori dettagli su questo argomento, si rimanda all'analisi fatta nel paragrafo 2.3.3.

4.2 Architettura software

L'architettura software di Home Manager è quella tipica di un sistema basato su TuCSoN: come meglio discusso nel capitolo 3, semplificando di molto, gli agenti vengono coordinati da centri di tuple, mentre le risorse sono gestite tramite *transducer*: per approfondimenti, si veda il capitolo 3.
Vi sono tuttavia alcune discrepanze fra l'architettura desiderata, che si descrive nel paragrafo 4.2.1, e l'effettiva implementazione nel prototipo, descritta invece nel paragrafo 4.2.2.

4.2.1 L'architettura desiderata (e desiderabile)

L'architettura auspicabile per Home Manager è composta da tre elementi principali:

- Le **risorse**, che possono essere qualunque tipo di oggetto più o meno *smart*, principalmente sensori e attuatori, sono incapsulate in *trasducer*. Come suggerisce il nome si occupano di "tradurre" le primitive ReSPeCT in un linguaggio specifico per il *device* o la classe di *device* per cui il *trasducer* è stato specificamente pensato.
- I **Tuple Centre** (TC) contengono informazioni sotto forma di tuple e si occupando di coordinare gli agenti. In Home Manager vi sono: un *Tuple Centre* per ciascuna stanza, altri due centri di tuple dedicati alla gestione dei *social network* e del meteo, e infine due TC speciali,

dotati di persistenza: `db`, che memorizza utenti con le relative preferenze, dispositivi, sensori, e altri dati importanti, e `rbac`, che contiene le informazioni sulla gestione dei permessi.

- Gli agenti si interfacciano con i centri di tuple per mezzo degli **Agent Coordination Context** (ACC). Ogni agente ha un suo ACC che gli permette di effettuare operazioni su uno o più centri di tuple. In Home Manager esistono agenti per svariati compiti: ad esempio, vi sono agenti per la gestione degli elettrodomestici, per la gestione del meteo e di Twitter, per la gestione dell'illuminazione e della temperatura.

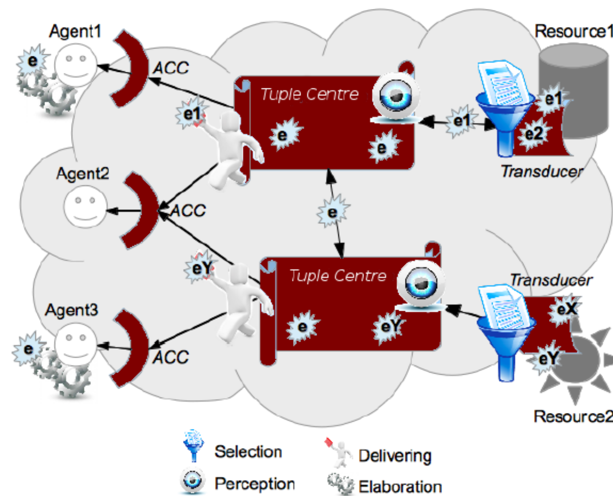

Figura 4.1: La tipica architettura di un sistema basato su TuCSoN.

4.2.2 L'architettura del prototipo

Approfondendo l'analisi del prototipo, si nota quanto di questa architettura sia effettivamente presente nell'implementazione attuale. Sostanzialmente, la discrepanza principale è rappresentata dal modo in cui vengono gestite le risorse.

Le risorse

Nel prototipo attuale le risorse sono rappresentate esclusivamente da tuple all'interno dei *tuple centre*. Non vi è il concetto di *transducer*, pertanto non vi è la possibilità di interfacciare ad alto livello con astrazioni dedicate il

mondo reale al prototipo sperimentale (benché sia comunque possibile farlo a basso livello, con componenti software sviluppati ad-hoc).
Alcuni esempi di risorse sono le seguenti tuple:

```
sensor(1,'HALL_1',1)
lamp(1,'HALL_FRONT_DOOR',1)
device(1,'Stereo Philips MCM 190/22',20,'on;off;cd;tuner','R',2)
```

- La prima tupla rappresenta un sensore di ingresso e uscita da una stanza, è identificato dal numero 1, si chiama `HALL_1` e si riferisce alla stanza con id 1;

- La seconda tupla rappresenta invece un punto luce. Anche questa è identificata dal numero 1, si chiama `HALL_FRONT_DOOR` ed è controllato dal sensore con id 1; lo stato (`on-off`) è gestito dal `LampAgent`, che si occuperà di creare una nuova tupla (e fare la *out* sul centro di tuple) contentente anche lo stato;

- La terza tupla rappresenta un dispositivo generico, in particolare uno stereo, è identificato dal numero 1, si chiama "Stereo Philips MCM 190/22", ha un consumo energetico di 20 W, ha come comandi possibili "`on;off;cd;tuner`", è di tipo R e si trova nella stanza con id 2.

I centri di tuple

In Home Manager vi sono numerosi *tuple centre*, che sono suddivisibili in:

- **Stanze**: ogni stanza della casa virtuale di Home Manager ha un suo centro di tuple;

- **Casa**: un centro di tuple per rappresentare la casa;

- **DB**: è un centro di tuple fondamentale per il funzionamento del sistema, memorizza al suo interno numerose informazioni, quali gli utenti e le loro preferenze, gli URL dei *web services* utilizzati dal servizio meteo, le credenziali di Twitter. Contiene inoltre al suo interno tuple che identificano sensori e dispositivi;

- **RBAC**: un altro centro di tuple fondamentale, contiene al suo interno le informazioni per il funzionamento del sistema *Role-Based Access Control* e quindi chi può accedere a cosa, quali sono le azioni consentite a ciascun utente, ecc..

Va notato che i *tuple centre* DB e RBAC sono persistenti, ovvero vengono memorizzati su disco sotto forma di file di testo scritto in XML. All'avvio del nodo, le tuple memorizzate al loro interno verranno caricate automaticamente.

```
TucsonTupleCentreId ingresso_tc = new TucsonTupleCentreId("ingresso_tc", "localhost", "20504");
TucsonTupleCentreId cucina_tc = new TucsonTupleCentreId("cucina_tc", "localhost", "20504");
TucsonTupleCentreId sala_tc = new TucsonTupleCentreId("sala_tc", "localhost", "20504");
TucsonTupleCentreId studio_tc = new TucsonTupleCentreId("studio_tc", "localhost", "20504");
TucsonTupleCentreId corridoio_tc = new TucsonTupleCentreId("corridoio_tc", "localhost", "20504");
TucsonTupleCentreId bagno_tc = new TucsonTupleCentreId("bagno_tc", "localhost", "20504");
TucsonTupleCentreId camera_tc = new TucsonTupleCentreId("camera_tc", "localhost", "20504");
TucsonTupleCentreId ripostiglio_tc = new TucsonTupleCentreId("ripostiglio_tc", "localhost", "20504");
TucsonTupleCentreId camera_doppia_tc = new TucsonTupleCentreId("camera_doppia_tc", "localhost", "20504");
TucsonTupleCentreId bagno_priv_tc = new TucsonTupleCentreId("bagno_privato_tc", "localhost", "20504");
TucsonTupleCentreId garage_tc = new TucsonTupleCentreId("garage_tc", "localhost", "20504");
```

Figura 4.2: I *tuple centre* delle stanze virtuali di Home Manager.

Gli agenti

Gli agenti di Home Manager, che gestiscono il comportamento dell'intera casa, possono essere classificati in varie categorie. Si può riassumere questa sotto-architettura nel seguente modo: vi sono alcuni agenti, definibili "pianificatori", che si occupano di realizzare un piano d'azione in base ai dati ottenuti dagli agenti predisposti al tracciamento dei movimenti e da alcuni agenti "esecutori". Questo piano verrà quindi realizzato concretamente dagli agenti "esecutori".

- **Agenti "pianificatori"**: in generale si occupano di elaborare il piano attraverso cui gestire la casa.
  - **ActControllerAgent**: determina la presenza di eventuali attività sospese, ad esempio un lavaggio della lavatrice.
  - **CmdControllerAgent**: gestisce i comandi inviati da terminale dagli utenti.
  - **ConflictsManagerAgent**: è in grado di valutare le informazioni fornite dagli altri agenti e risolvere i conflitti, generando un nuovo piano apposito.
  - **PrefControllerAgent**: gestisce le preferenze degli utenti in una data stanza di cui si deve elaborare il piano.
- **Agenti "esecutori"**: si occupano di agire direttamente sulle risorse, attuando il piano degli agenti "pianificatori".

- **DeviceAgent**: è l'agente che gestisce un dispositivo o un elettrodomestico. Per esempio, può eseguire operazioni, subordinate alle regole di coordinazione dettate dal *tuple centre* "`tc_cucina`", sul dispositivo "Stereo Philips MCM 190/22", che ha come parametri, ovvero come operazioni ammesse su di esso, "`on;off;cd;tuner`". Importante notare che l'agente sa come interfacciarsi al sistema grazie al suo ACC, che viene specificato nel momento dell'inizializzazione.
- **BrightnessAgent**: è l'agente che gestisce i sensori di luminosità per regolare di conseguenza le luci e le tapparelle di una stanza. In base alla presenza o assenza di luce esterna, gestisce luci e tapparelle in modo da ottimizzare il risparmio di energia elettrica.
- **PlanDistributorAgent**: sostanzialmente si occupa di distribuire i comandi fra le entità coinvolte nell'attuazione di un determinato piano. Nel prototipo, aggiorna la temperatura di una stanza inviando comandi a tutti gli attuatori.
- **LampAgent**: si occupa di regolare lo stato delle lampade della casa.
- **WindowAgent**: gestisce infissi automatizzati.
- **BlindAgent**: gestisce tapparelle automatizzate.

• **Agenti per il tracciamento dei movimenti**: si tratta di agenti che monitorano la posizione degli abitanti della casa, per permettere al sistema di conoscere sia se c'è qualcuno nelle varie stanze, sia chi c'è nelle varie stanze, in modo da applicare le preferenze di ognuno.

- **DetectorAgent**: si occupa di controllare i movimenti delle persone fra le varie stanze, sfruttando dei sensori appositi presenti in ciascuna stanza.
- **ListManager**: si occupa di tenere aggiornata una lista contenente le informazioni sulle presenze in ciascuna stanza. Grazie a questi dati, il sistema provvede a regolare le luci della stanza.

• **Agenti generici**

- **PlanEntitiesAgent**: è l'agente che coordina gli altri agenti pianificatori.

4.3 L'implementazione di Home Manager

Home Manager è implementato interamente in Java, linguaggio di programmazione orientato agli oggetti e specificatamente progettato per essere il più possibile indipendente dalla piattaforma di esecuzione. Per questo motivo, il prototipo è già eseguibile sia su sistemi Windows che Unix (Linux e MacOS X), ma anche su differenti architetture hardware, ad esempio x86 e ARM (Carano, 2015).
Nell'implementare l'architettura vengono ovviamente utilizzate apposite Java API. In questo paragrafo verranno descritte le API più utilizzate nel prototipo attuale: per maggiori dettagli sulle API TuCSoN si rimanda a (Omicini and Mariani, 2015).
La maggior parte delle API fanno parte del *package* `alice.tucson.api`.

4.3.1 API per gli agenti

- `TucsonAgentId`: espone metodi per ottenere un agent ID TuCSoN e per accedere ai suoi campi. È inoltre necessario per ottenere un ACC.
- `TucsonMetaACC`: fornisce un meta-ACC, che è necessario ad ottenere un ACC, che è a sua volta necessario ad interagire con un *tuple centre*.
- `AbstractTucsonAgent`: classe astratta da usare come base per agenti TuCSoN personalizzati. Costruisce automaticamente il `TucsonAgentId` e ottiene il `EnhancedACC` necessario al suo funzionamento.
  Tutti gli agenti di Home Manager descritti nel paragrafo 4.2.2 estendono questa classe.

4.3.2 API per i Tuple Centre

- `TucsonTupleCentreId`: espone metodi per ottenere un tuple centre ID e per accedere ai suoi campi. È necessario per fare operazioni TuCSoN su un ACC.

4.3.3 API generiche

- `ITucsonOperation`: espone metodi per accedere ai risultati di un operazione TuCSoN:
  - `isResultSuccess(): boolean` - serve per controllare il successo di una determinata operazione.

- getLogicTupleResult(): LogicTuple - serve per ottenere il risultato di una determinata operazione.
- getLogicTupleListResult(): List<LogicTuple> - serve per ottenere una lista di risultati di una determinata operazione.

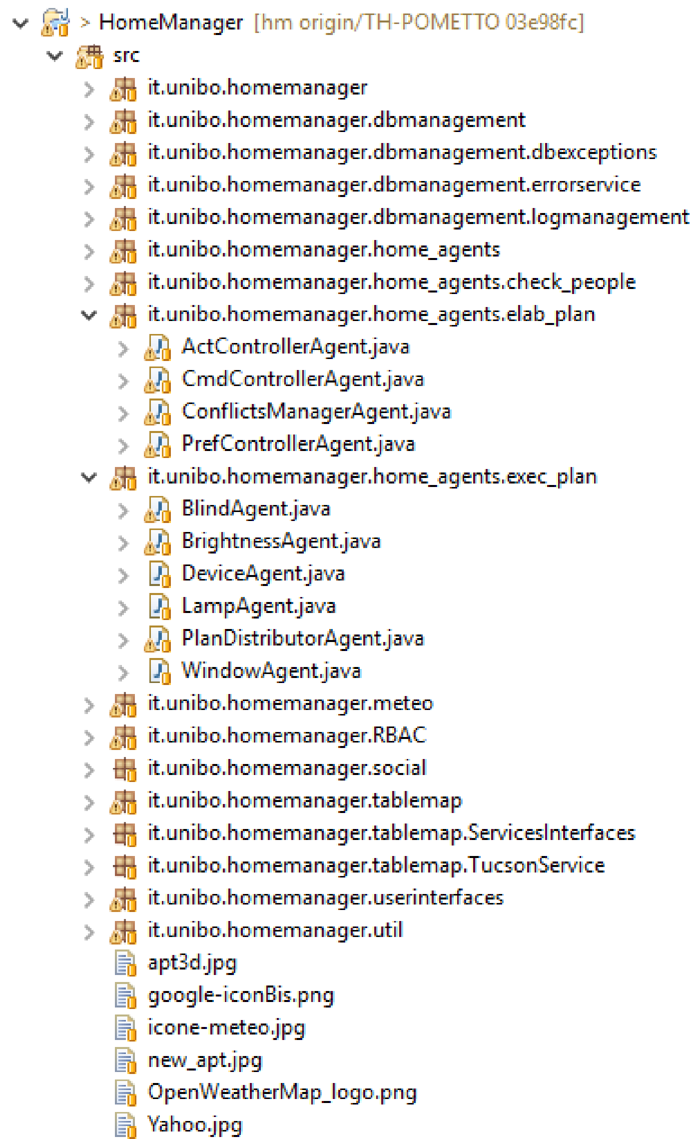

Figura 4.3: I *packages* in cui è organizzato Home Manager.

Capitolo 5

Analisi e progettazione

Il primo passo da fare per integrare il concetto di transducer in Home Manager è quello di trovare un punto in cui andare ad inserire la tecnologia omonima.
Essendo un software per la gestione di un sistema domotico, sensori e attuatori - benché siano simulati - sono largamente presenti anche nel prototipo attuale: tapparelle, sensori luminosi e di temperatura, elettrodomestici sono tutti candidati ottimi all'integrazione di un *transducer*.
Al fine di questa tesi, si decide di andare ad intervenire nella gestione dell'illuminazione.

5.1 Analisi della gestione delle luci

Al momento, in Home Manager una lampadina è sostanzialmente la tupla `light_curr_st(ID, ST)`, dove `ID` rappresenta un identificativo univoco della luce, mentre `ST` è il suo stato corrente (es. `on`, `off`).
Le tuple `light_curr_st(ID, ST)` si trovano nei tuple centre relativi alle varie stanze, per cui, ad esempio, in `ingresso_tc` potremmo avere le tuple `light_curr_st(1, 'on')` e `light_curr_st(2, 'off')`.
In figura 5.2, grazie alla proprietà di ispezionabilità dei *tuple centre* e al tool TuCSoN Inspector è possibile osservare il centro di tuple `camera_tc`, che contiene al sui interno una tupla rappresentante una lampadina.

In figura 5.1 viene mostrato come avviene il cambiamento di stato di una lampadina. Chi si occupa di questo compito è l'agente `LampAgent`, che si mette in attesa, con una primitiva dalla semantica sospensiva, della tupla `light_mode(ID, ST)` dove `ID` è l'identificativo univoco della lampada, mentre `ST` è lo stato che si vuole avere (`on` o `off`).

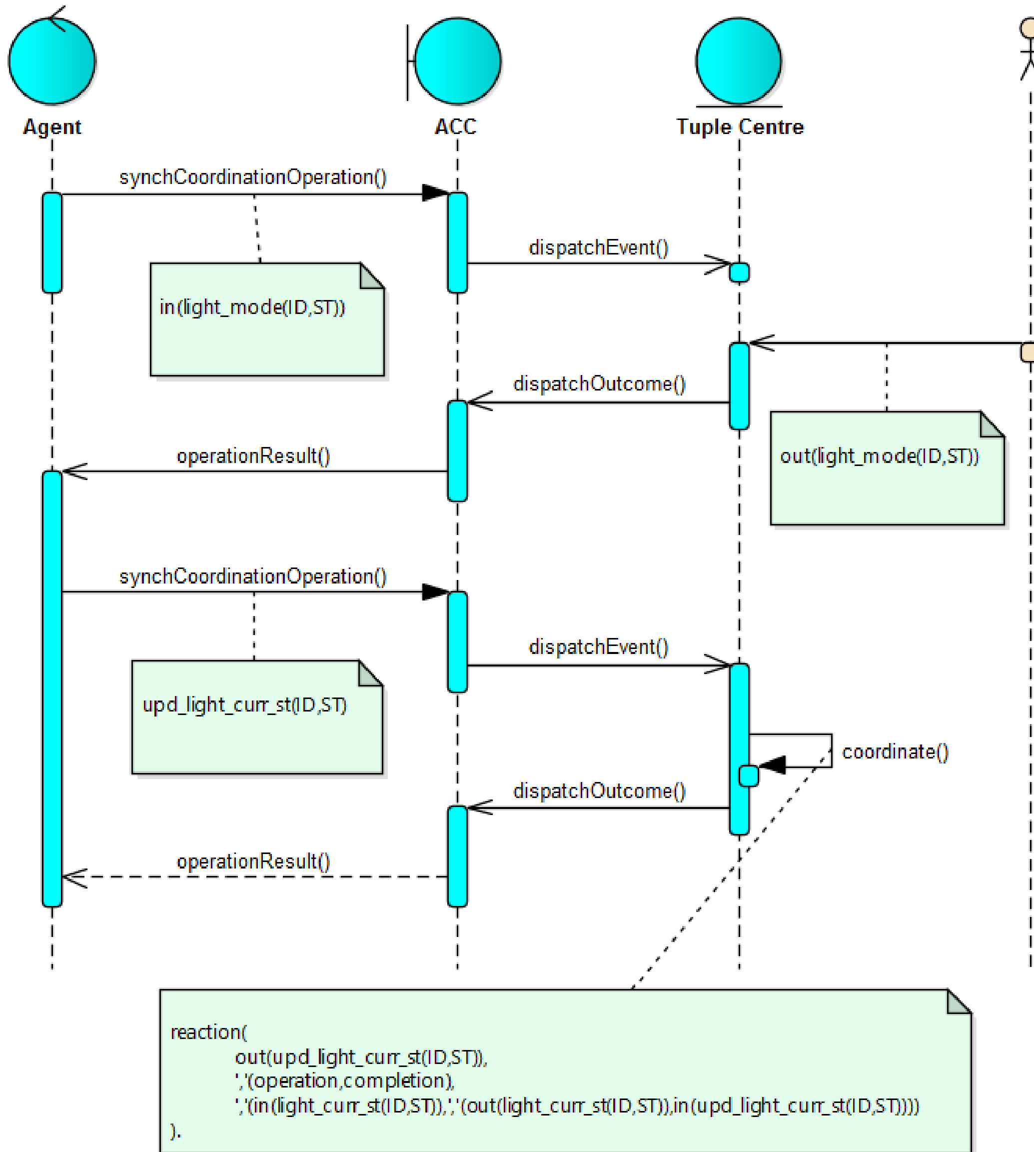

Figura 5.1: Il funzionamento attuale delle luci in Home Manager.

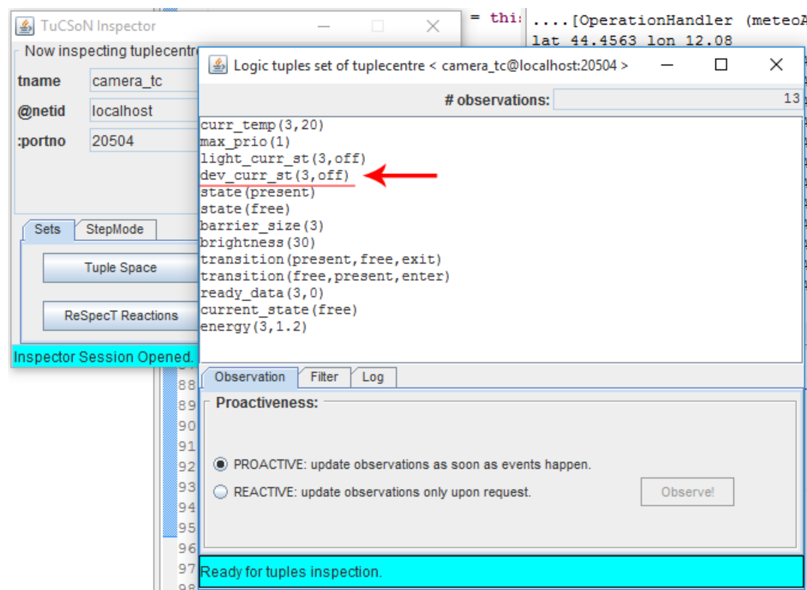

Figura 5.2: Una lampada come rappresentata attualmente in Home Manager: una semplice tupla in un *Tuple Centre*.

Nel momento in cui nel *tuple centre* compare una tupla `light_mode(ID, ST)`, il `LampAgent` emetterà una tupla `upd_light_curr_st(ID, ST)`, che scatenerà un'apposita reazione (vedi fig. 5.1) nel centro di tuple della stanza, che andrà a modificare lo stato della tupla `light_curr_st(ID, ST)`.

Dal punto di vista dell'interfaccia grafica, è possibile osservare lo stato attuale delle luci nella pagina *View Plan* (vedi figura 5.3), dove vi è un apposito spazio che mostra, appunto, se le luci collegate ad un determinato sensore di luminosità sono accese o spente.

5.2 Progettazione

Nella situazione attuale si nota subito come non sia immediato trasformare una lampadina "tupla" in una lampadina reale.
Una possibile soluzione potrebbe essere quella di inserire un apposito agente che vada a leggere (a *polling* o grazie ad una reaction programmata sul *tuple centre*) lo stato della tupla `light_curr_st(ID, ST)` e reagisca ai cambiamenti della stessa, Questa soluzione non è però l'ideale, in quanto andrebbe scritto ulteriore codice che a ogni piccola modifica sarebbe da cambiare. Si avrebbe anche un agente in più per ogni lampada, appesantendo il sistema.

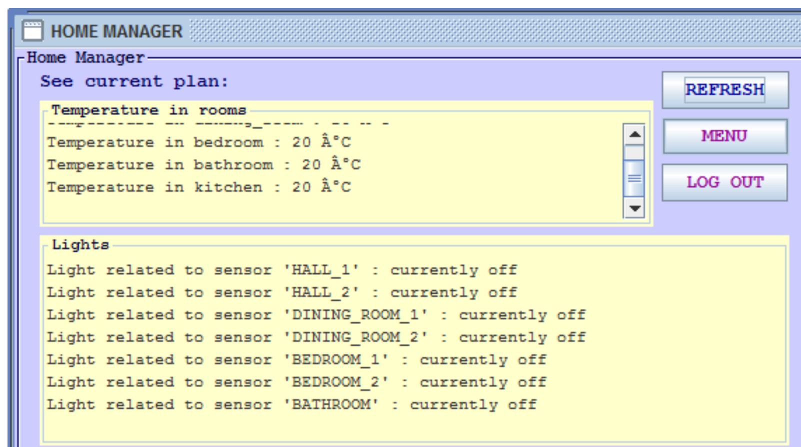

Figura 5.3: La pagina *View Plan* di Home Manager.

In questa fase progettuale si cercherà di mantenere quanto più invariato possibile il funzionamento del sistema, eccezion fatta, ovviamente, per le modifiche tecniche necessarie al funzionamento dei *transducer*.
Come già visto precedentemente (si veda la sezione 3.3), un *transducer* è in realtà diviso in tre parti: è necessario progettare il *transducer* vero e proprio, vanno scelte le *reaction* da programmare nel/nei *tuple centre* e infine va progettato un *probe*, ovvero il dispositivo (o, per meglio dire, il “ponte” con il dispositivo, una sorta di *driver*) che vogliamo andare a gestire.
Si vogliono anche mantenere i `LampAgent`, che continueranno ad aggiornare lo stato delle luci presenza di una tupla `light_mode(ID, ST)`.

5.2.1 Il transducer attuatore

Inizialmente l'idea è quella di progettare un *transducer* attuatore.
Mantenendo invariato il ruolo del `LampAgent`, questo si troverà ancora una volta ad attendere che venga emessa la tupla di comando `light_mode(ID, ST)`, come in figura 5.1. Il comportamento successivo però è completamente diverso, e le interazioni dovranno avvenire come in figura 5.4. Sostanzialmente, il `LampAgent` dovrà emettere una tupla `act(light_mode(ID, ST))`, che, attraverso un'apposita *reaction*, farà scatenare un evento ambientale nel `LightTransducer`, finalizzato all'accensione o allo spegnimento della lampada, modellata da `ActualLight`.
A questo punto della progettazione, tuttavia, ci si rende però conto che, per mantenere il funzionamento attuale del sistema, che interroga le luci per sa-

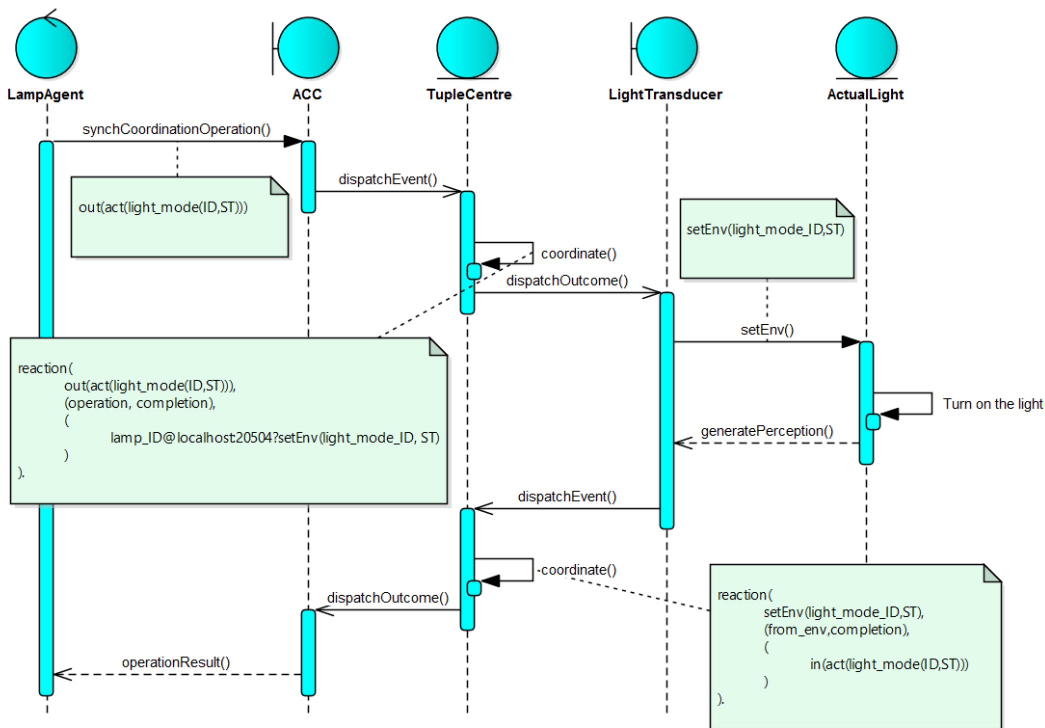

Figura 5.4: Le interazioni nel prototipo dopo l'inserimento di un *transducer* attuatore.

pere se sono accese o spente, può essere utile un *transducer* attuatore-sensore.

5.2.2 Il transducer attuatore-sensore

Come accennato nel paragrafo precedente, si decide di estendere il comportamento del *transducer* da semplice attuatore a attuatore-sensore, per permettere al sistema di interrogare una luce circa il proprio stato. In un primo momento, lo stato sarà solamente `on` oppure `off`, ovvero lampadina accesa o spenta. Tuttavia questa implementazione pone già le basi per un attuatore intelligente, in grado di dire se ci sono problemi sulla lampada, ad esempio se è fulminata o se vi è un corto circuito.
In figura 5.5 è possibile osservare come deve comportarsi il sistema nel momento in cui viene richiesto lo stato della lampadina. Nell'esempio viene mostrato il processo `ViewPlanPanel`, che si occupa di mostrare lo stato attuale del piano d'azione di Home Manager. Il funzionamento è simile a quello che dovrà avere l'agente `LampAgent`, ma funziona al contrario: `ViewPlanPanel`

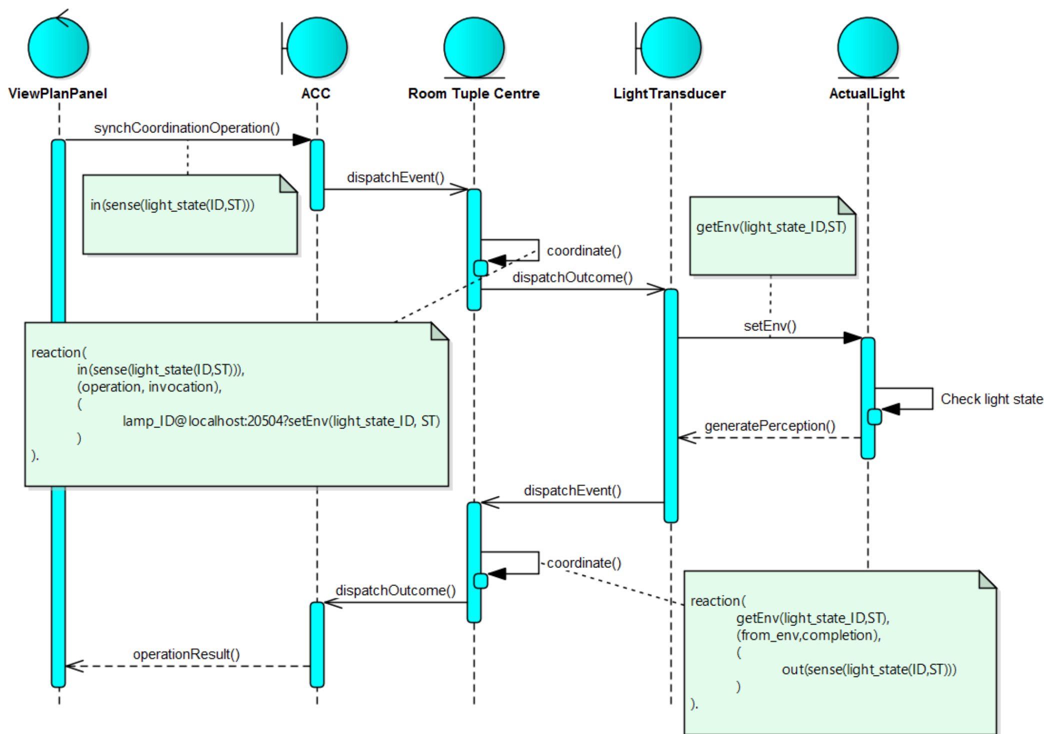

Figura 5.5: Il sistema interroga il *transducer* per conoscere lo stato della lampadina.

richiede al *transducer*, tramite una tupla `sense(light_state(ID,ST))`, di emettere a sua volta una tupla contenente lo stato attuale della lampada.

5.2.3 Il probe

Il *probe* (vedi sezione 3.3 per maggiori dettagli) di questo prototipo, che verrà chiamato `ActualLight`, come accennato sopra non sarà collegato ad un dispositivo reale, ma rappresenterà una lampadina simulata nel funzionamento. Si decide di memorizzare lo stato in un apposito *tuple centre*, da utilizzare come database, `illumination_tc`. Accendere o spegnere una lampada si tradurrà quindi nel modificare una tupla su questo database.
Dato che è il *probe* dev'essere in grado di comportarsi sia da attuatore che da sensore, andranno previsti due diversi comportamenti:

- il comportamento da attuatore, che consisterà nell'andare a eliminare la vecchia tupla dal *tuple centre* e ad emetterne una nuova con lo stato aggiornato;

- il comportamento da sensore, in cui dovrà restituire lo stato attuale della "lampada" leggendo la tupla che lo memorizza.

È possibile osservare il comportamento progettato per l'`ActualLight` nelle figure 5.6 e 5.7.

5.2.4 Le Reaction

Benché siano già state mostrate nelle figure 5.4 e 5.5, si riportano per comodità anche di seguito le *reaction* progettate, necessarie per il funzionamento del sistema. Di seguito le reazioni per la modalità attuatore:

```
reaction(
  out(act(light_mode(ID, ST))),
  (operation, completion),
  (lamp_ID@localhost:20504 ? setEnv(light_mode_ID, ST))
).

reaction(
  setEnv(light_mode_ID, ST),
  (from_env, completion),
  (in(act(light_mode(ID, ST))))
).
```

E quelle per la modalità sensore:

```
reaction(
  in(sense(light_state(ID, ST))),
  (operation, invocation),
  (lamp_#@localhost:20504 ? getEnv(light_state_ID, ST))
).

reaction(
  getEnv(light_state_ID, ST),
  (from_env, completion),
  (out(sense(light_state(ID, ST))))
).
```

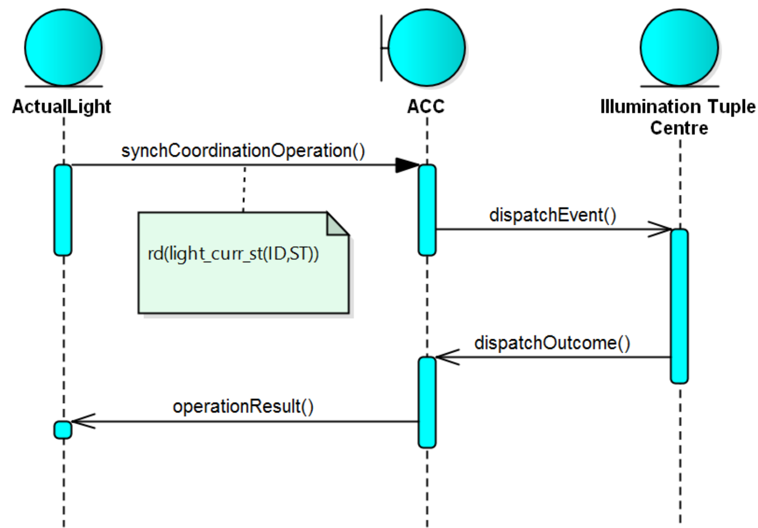

Figura 5.6: Comportamento di `ActualLight` quando interrogato come sensore.

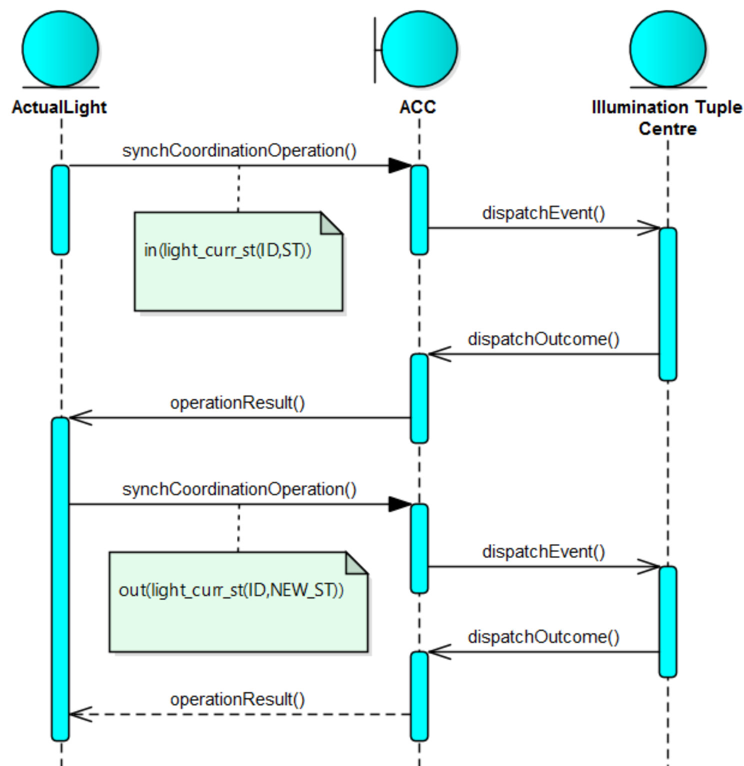

Figura 5.7: Comportamento di `ActualLight` quando utilizzato come attuatore.

Capitolo 6

L'implementazione

Come deciso in fase di progettazione, andranno realizzati:

- un *transducer*, implementato nella classe Java `LightTransducer` che estende la classe astratta `AbstractTransducer`;
- un *probe*, implementato nella classe Java `ActualLight` che implementa l'interfaccia `ISimpleProbe`.

Possiamo osservare queste classi in figura 6.1. Sarà anche necessario adattare il codice attuale dell'agente `LampAgent`, implementato come classe che estende `AbstractTucsonAgent`, perché vada a eseguire nuovi *task*, che non erano necessari nella versione di Home Manager senza *transducer*.

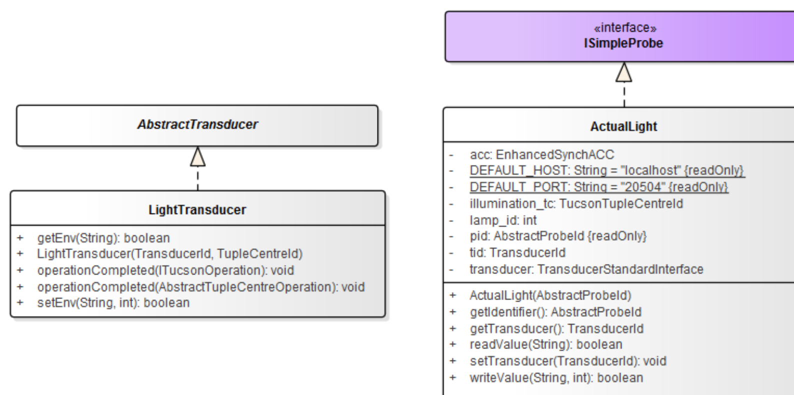

Figura 6.1: Le classi `LightTransducer` e `ActualLight`

6.1 Obbiettivo dell'implementazione

Dal punto di vista concettuale, una volta implementate le nuove classi sopra citate ed effettuate le altre modifiche richieste, l'architettura Butlers - almeno relativamente alla gestione delle luci - risulterà adottata quasi completamente, come mostrato in figura 6.2. Pertanto, a differenza del prototipo attuale che non sfrutta i vantaggi dei *transducer* per implementare i livelli 1 e 2 (per maggiori dettagli, si rimanda al paragrafo 2.3.3) e dunque l'interazione è limitata - a livello coordinativo - solo con al mondo simulato, le modifiche previste, sfruttando la coordinazione situata dei *transucer*, permettono al prototipo di interagire con il mondo reale.

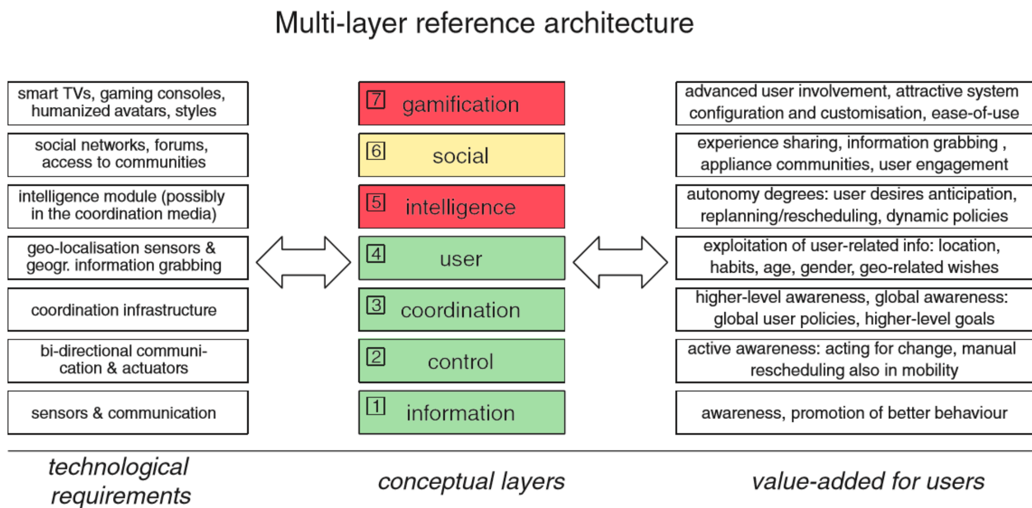

Figura 6.2: L'architettura Butlers in Home Manager dopo le modifiche.

6.2 LightTransducer

L'implementazione dei *transducer* non varia di molto fra i vari tipi. Nel caso del `LightTransducer`, la particolarità sta nel fatto che saranno implementate sia la funzione `getEnv`, che permette al *transducer* di comportarsi da sensore, sia la `setEnv`, che permette al *transducer* di comportarsi da attuatore, in modo che, come deciso in fase di progettazione, sia possibile sia modificare che leggere lo stato di una luce.

6.2.1 LightTransducer Sensore

Tramite la `getEnv` il *transducer* stimola ciascuna lampada ad esso associata ad analizzarsi per conoscere il suo stato.

```
public boolean getEnv(final String key) {
  this.speak("[" + this.id + "]: Reading...");
  boolean success = true;
  final Object[] keySet = this.probes.keySet().toArray();
  for (final Object element : keySet) {
    if (!((ISimpleProbe) this.probes.get(element)).readValue(key)) {
      this.speakErr("[" + this.id + "]: Read failure!");
      success = false;
      break;
    }
  }
  return success;
}
```

6.2.2 LightTransducer Attuatore

Tramite la `setEnv` il *transducer* stimola ciascuna lampada ad esso associata a modificare il proprio stato, modificando l'ambiente in cui si trova.

```
public boolean setEnv(String key, int value) {
  this.speak("[" + this.id + "]: Writing...");
  boolean success = true;
  final Object[] keySet = this.probes.keySet().toArray();
  for (final Object element : keySet) {
    if (!((ISimpleProbe) this.probes.get(element))
                                    .writeValue(key, value)) {
      this.speakErr("[" + this.id + "]: Write failure!");
      success = false;
      break;
    }
  }
  return success;
}
```

6.3 ActualLight

Si mostrano ora alcuni passaggi implementativi significativi per l'`ActualLight`. Come deciso in fase di analisi, il probe `ActualLight` si limiterà a leggere e scrivere lo stato della lampadina utilizzando una tupla `light_curr_st(ID,ST)` e il *tuple centre* `illumination_tc` come fosse un database.

6.3.1 Lettura dello stato

Questa funzione, subito dopo aver letto lo stato attuale dal centro di tuple-database, deve tradurre lo stato logico della lampadina in un valore utilizzabile con i *transducer*. Per una limitazione attuale, infatti, una tupla ambientale può avere come parametro solo un valore `int`. Pertanto si decide di assegnare allo stato di acceso il valore `1`, mentre allo stato di spento il valore `o`. Dopo questa "traduzione" dello stato, viene notificato il *transducer* di competenza che lo stato è quello letto.

```
public boolean readValue(String key) {
[...]
final LogicTuple template =
    = LogicTuple.parse("light_curr_st(" + this.lamp_id + ",_)");
final ITucsonOperation op =
    = this.acc.rd(this.illumination_tc, template, null);
if (op.isResultSuccess()) {
  final String stato = op.getLogicTupleResult().getArg(1).toString();
  final int state;
  if (stato.equals("off")) {
    state = 0;
  } else if (stato.equals("on")) {
    state = 1;
  } else state = 3;
  this.transducer.notifyEnvEvent(key, state, AbstractTransducer.GET_MODE);
}
[...]
```

6.3.2 Scrittura dello stato

Questa funzione in primo luogo analizza la tupla ambientale che è stata ricevuta, esattamente come per la funzione `readValue`: il valore `0` rappresenta lo stato da impostare spento, mentre il valore `1` rappresenta lo stato da impostare acceso. Qualunque altro valore non è considerato valido. Dopo questa

"traduzione" dello stato, viene eliminata la tupla con lo stato attuale (se presente) con una `inp`, e subito dopo viene emessa la tupla con il nuovo stato.

```
public boolean writeValue(String key, int state) {
  [...]
  String stato;
  if (state == 0) {
    stato = "off";
  } else if (state == 1) {
    stato = "on";
  } else {
    System.err.println("[" + this.pid + "]: stato non valido: " + state);
    return false;
  }
  [...]
  final LogicTuple template =
      = LogicTuple.parse("light_curr_st(" + lamp_id + ",_)");
  final ITucsonOperation op =
      = this.acc.inp(this.illumination_tc, template, null);
  if (op.isResultSuccess()) {
    final LogicTuple nuovaTupla =
        = LogicTuple.parse("light_curr_st(" + lamp_id + "," + stato  + ")");
    this.acc.out(this.illumination_tc, nuovaTupla, null);
    this.transducer.notifyEnvEvent(key, state, AbstractTransducer.SET_MODE);
    return true;
  }
[...]
```

6.4 LampAgent

Si evidenziano ora alcuni punti significativi delle modifiche nell'implementazione del `LampAgent`: questo agente avrà ora alcuni compiti importanti, che consistono:

- nel caricamento delle specifiche delle *reaction* da file e, di conseguenza, nella programmazione delle reazioni nei *tuple centre* delle varie stanze;
- nella generazione della tupla di configurazione e nella sua emissione nel *tuple centre* speciale di configurazione del sistema;

- nell'adempiere alla sua funzione principale, permettere il cambio di stato della lampada.

Vediamo ora nel dettaglio queste fasi.

6.4.1 Caricamento delle specifiche

Innanzitutto, ogni LampAgent si occupa di programmare le *reaction* sul *tuple centre* relativo alla stanza in cui si trova la specifica luce che gestisce.
Le specifiche delle reazioni, memorizzate in un file `rsp`, includono un marcatore `#`. Questo *marker* viene sostituito con l'id della lampada: in questo modo le reazioni programmate saranno univoche per ciascuna lampadina, anche se nella stessa stanza ve ne sono più di una.
Per comodità, è inoltre stata implementata una versione ad hoc della funzione `outS`, che prende in ingresso una stringa di configurazione anziché singole tuple (`E`, `G`, `R`), in maniera del tutto simile a quello che fa la `setS` già presente nelle librerie: questa funzione permette perciò il caricamento di specifiche in blocco in un dato centro di tuple.
In maniera estremamente semplificata, ecco i passaggi che esegue il `LampAgent` per adempiere a questo primo compito.

```
String config = Utils.fileToString("[...]/lightSpec.rsp");
config = config.replaceAll("#", "" + l.idL);
this.outS(room_tc, config);
```

Di seguito un esempio di specifica per una reazione. In fase di progettazione sono state elencate completamente nel paragrafo 5.2.4. Ogni marcatore `#` verrà sostituito dall'ID della lampada a cui si riferiscono. Ad esempio:

```
reaction(
  out(act(light_mode(#, T))),
  (operation, completion),
  (lamp_#@localhost:20504 ? setEnv(light_mode_#, T))
).
```

Diventerà:

```
reaction(
  out(act(light_mode(3, T))),
  (operation, completion),
  (lamp_3@localhost:20504 ? setEnv(light_mode_3, T))
).
```

6.4.2 Generazione della tupla di configurazione

Il `LampAgent` si occupa anche di generare la tupla di configurazione del *transducer* nel *tuple centre* speciale di configurazione del sistema. I valori della tupla rappresentano:

- La chiave `createTransducerActuator`;
- L'ID del tuple centre;
- La classe Java che implementa il *transducer*;
- L'ID del *transducer*;
- La classe Java che implementa il *probe*;
- L'ID del probe.

Nel codice si ha quindi:

```
final LogicTuple actuatorTuple = new LogicTuple(
  "createTransducerActuator",
  new TupleArgument(room_tc.toTerm()),
  new Value("it.unibo.homemanager.situatedness.LightTransducer"),
  new Value("lightTransducer_" + l.idL),
  new Value("it.unibo.homemanager.situatedness.ActualLight"),
  new Value("lamp_" + l.idL)
);
acc.out(configTc, actuatorTuple, null);
```

6.4.3 Cambio di stato

L'ultimo task di questo agente è permettere il cambio di stato della luce. Come già spiegato più volte, il `LampAgent` si mette in attesa della tupla che determina il cambiamento dello stato della lampadina. Nel momento in cui viene sbloccato, emette la tupla `act(light_mode(ID,ST))`, andando così ad agire sull'ambiente, modificandone lo stato.

```
template_in =
     = new LogicTuple("light_mode",new Value(l.sensId),new Var("X"));
ITucsonOperation op_in =
     = acc.in(room_tc, template_in, Long.MAX_VALUE);

LogicTuple result = op_in.getLogicTupleResult();
```

```
LogicTuple action = null;
action = LogicTuple.parse("act(" + result +")");

acc.out(room_tc, action, null);
```

6.5 La lettura dello stato

Per completezza, si mostra anche il punto in cui viene letto lo stato delle luci in Home Manager, da parte di `ViewPlanPanel`. Il funzionamento è piuttosto complesso ed è legato al modo in cui funziona Home Manager. In poche parole, per avere la lista di tutte le luci e delle stanze in cui si trovano sfrutta l'elenco dei sensori di luminosità: va ad interrogare il *transducer* relativo alla luce associata a ciascun sensore, per conoscerne lo stato e farne un elenco.

```
[...]
Vector sensors =
    =  this.sensorService.getSensors(database.getDatabase());
int size = sensors.size();
for(int i=0; i<size ;i++)
{
  Sensor sensor = (Sensor) sensors.get(i);

  TucsonTupleCentreId stanza_tc =
        = (TucsonTupleCentreId)tid.elementAt(sensor.roomId-1);

  LogicTuple template = LogicTuple.parse(
         "sense(light_state(" + sensor.idSens + ",_))");

  ITucsonOperation op = acc.in(stanza_tc, template, null);

  if (op.isResultSuccess()) {
    LogicTuple l = op.getLogicTupleResult();
    light.add(l);
  }
}
[...]
```

Dopo aver fatto questa operazione, genera delle stringhe con lo stato effettivo delle luci (ricordando che le tuple usate dai *transducer* possono avere solo un parametro di tipo `int`).

```
[...]
String value = lt.getArg(0).getArg(1).toString();
if (Integer.parseInt(value) == 0) {
  stato = "'off'";
} else if (Integer.parseInt(value) == 1) {
  stato = "'on'";
} else {
  stato = "'uknown state'";
}
str += s.name+" : currently "+ stato;
[...]
```

6.6 Sostituibilità: un esempio

Come spiegato nel paragrafo 3.3, un grande vantaggio dei *transducer* è quello della sostituibilità. A tal proposito, per sperimentare questo fatto è stato realizzato un nuovo *probe*, `ToyLight`, ed è stato sostituito ad `ActualLight` senza cambiare alcuna riga di codice in Home Manager, eccezion fatta, ovviamente, per la tupla di configurazione che diventa, rispetto a quella mostrata nel paragrafo 6.4.2:

```
final LogicTuple actuatorTuple = new LogicTuple(
  "createTransducerActuator",
  new TupleArgument(room_tc.toTerm()),
  new Value("it.unibo.homemanager.situatedness.LightTransducer"),
  new Value("lightTransducer_" + l.idL),
  new Value("it.unibo.homemanager.situatedness.ToyLight"),
  new Value("lamp_" + l.idL)
);
```

Il colorato risultato di questa sostituzione è mostrato in figura 6.3.

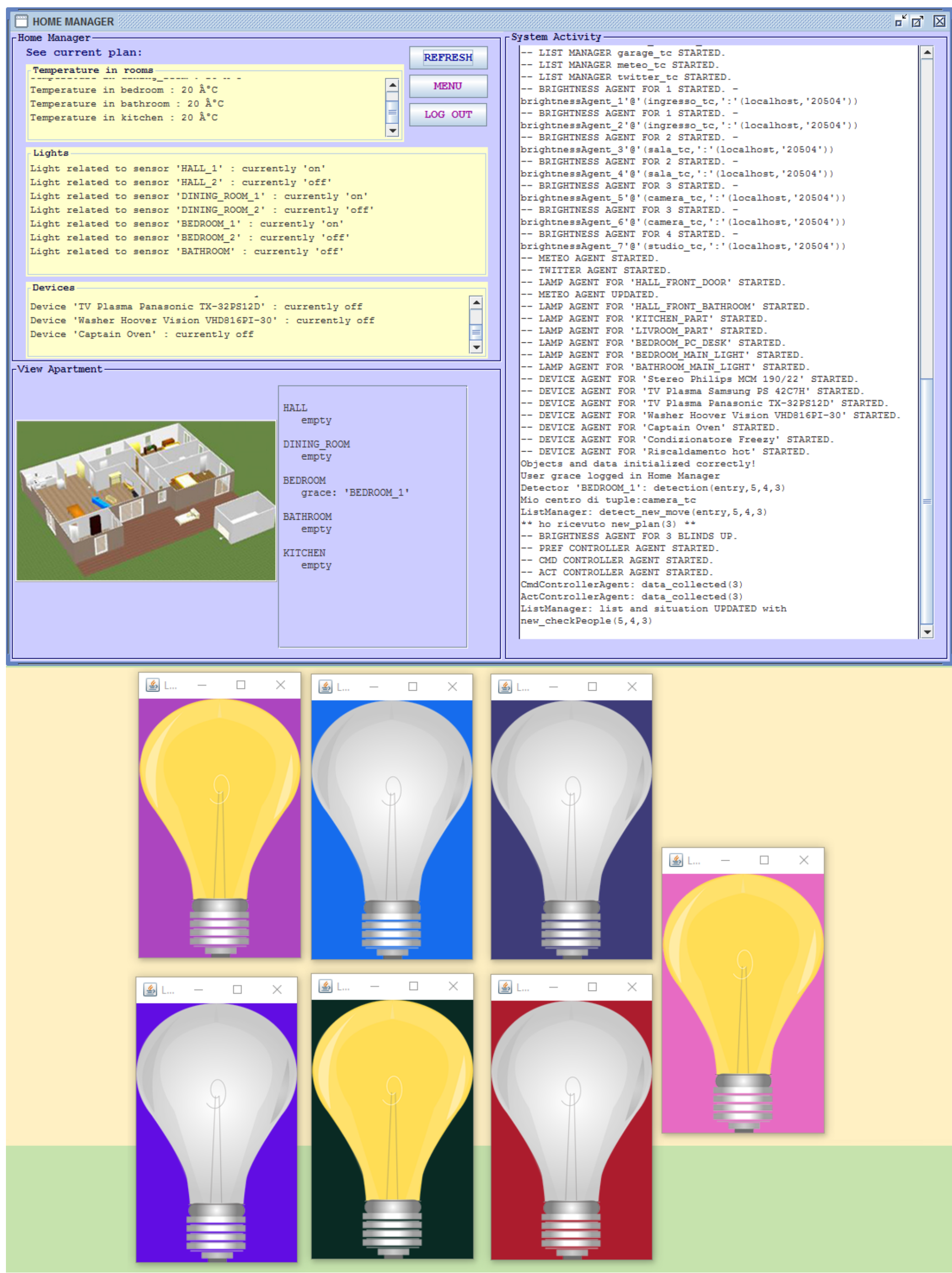

Figura 6.3: `ToyLight`: un *probe* alternativo a `ActualLight` che mostra graficamente lo stato delle lampade.

Capitolo 7

Conclusioni

L'obbiettivo di questa tesi, ovvero integrare nel prototipo di Home Manager il concetto e la tecnologia dei *transducer* è stato raggiunto. L'architettura Butlers è ora più solidamente presente: anche i livelli 1 e 2 sono stati completamente implementati, sfruttando la coordinazione situata che, in TuCSoN, forniscono i *transducer*.

È evidente come, sebbene siano state necessarie alcune modifiche al sistema attuale, questa scelta porti indubbi vantaggi: basti pensare all'estrema facilità con cui sarà ora possibile interfacciare il prototipo con il mondo reale, semplicemente sostituendo il *probe* simulato `ActualLight`.

7.1 Sviluppi futuri

Attualmente sono stati modellati *transducer* e *probe* relativi all'illuminazione. Un ulteriore passo da compiere sarà quello di migrare verso questa tecnologia tutti i dispositivi, i sensori e gli attuatori presenti in Home Manager.

Come citato sopra, grazie alle modifiche effettuate sarà più semplice interfacciare Home Manager con il mondo esterno: per questo motivo sarà sicuramente affascinante progettare nuovi *probe* che sfrutteranno microcontrollori e *smart device* per rendere meno prototipo e più reale Home Manager.
Ad esempio, si potrebbe realizzare un `ArduinoLight`, interfacciata tramite porta seriale alla nota scheda per prototipi: lasciando assolutamente inalterato il programma attuale, sarà possibile inviare comandi di accensione e spegnimento ad una lampada reale, e allo stesso tempo interrogarla per conoscerne lo stato.

Ringraziamenti

Innanzitutto, un ringraziamento per i prof. Andrea Omicini e Enrico Denti, che mi hanno dato l'opportunità di conoscere l'affascinante argomento di questa tesi e di contribuire a questo progetto così interessante. E ai dott. Stefano Mariani e Roberta Calegari, per l'aiuto ed i preziosi consigli che mi hanno dato.

Il primo pensiero va, come sempre, alla mia ragazza Chiara, che mi ha incoraggiato e sostenuto in tutti questi anni di studio, è sempre stata presente al mio fianco e mi ha dato la forza per arrivare fino in fondo. Senza di lei, sicuramente, non sarei la persona che sono oggi.

Vorrei poi ringraziare i miei genitori, che hanno sempre supportato le mie scelte, aiutandomi a portare a termine questo percorso di studi, e sopportandomi nei momenti più difficili. Un grazie anche a mia sorella Sabrina, che con la sua energia da quindicenne mi ricorda costantemente quanto io sia un punto di riferimento per lei e mi sprona a cercare di migliorarmi costantemente. Ai genitori di Chiara, che mi hanno accolto come un figlio. E che mi hanno permesso di studiare al fresco nonostante le estati caldissime degli ultimi anni, nella tranquillità di Verucchio. Al fratello di Chiara, Lorenzo, che sa sempre come risolvere le situazioni con una delle sue battute e freddure.

Ai miei amici più cari va un'altra fetta importante della mia riconoscenza.
A Irene, ma soprattutto a Marco, grazie per tutte le belle esperienze che abbiamo fatto, alle avventure fantastiche, sempre nuove, che riusciamo ad inventarci insieme.
A Francesca, le nostre serate cinema sono fra le cose che più mi mancano dei primi anni universitari. Il saperti sempre disponibile per un consiglio o semplicemente per liberarmi di qualche peso parlandone, è stato ed è tutt'ora per me fondamentale. Grazie per esserci sempre!
A Patryk e Silvia, cosa non darei per avervi conosciuto prima! Sarebbe stato

senza dubbio un percorso più spensierato. Grazie per la vostra allegria e per avermi fatto vedere come tutti i problemi si possano risolvere con un sorriso.
Un ringraziamento speciale anche all'Associazione Culturale "Rilego e Rileggo", in cui ho potuto fare tantissime esperienze diverse, con le quali ho imparato e sono cresciuto, oltre ad avermi fatto conoscere tante persone su cui so di poter contare: Antonia, Sabrina, Domenico, Nadia, Alessandro, Francesca, Davide, Edoardo, e tutti coloro che non possono, ovviamente, stare in questa pagina per motivi di spazio.
Un grazie anche ai miei due amici di infanzia, Matteo e Lorenzo. Matteo, per avermi tenuto compagnia durante questi anni, in cui abbiamo condiviso i momenti felici e quelli più difficili. Lorenzo, con la sua incredibile attitudine, per essere stato un esempio (irraggiungibile) di tenacia e di perseveranza nel raggiungere i propri obbiettivi.

Per ultimo, ma non meno importante, un ringraziamento anche a zia Liliana, che ha contribuito a far cominciare la mia carriera universitaria. E agli altri zii e zie, nonni e nonne, cugini e cugine, che nonostante le distanze mi hanno fatto sentire il loro sostegno.

Grazie anche a tutte quelle persone che ho dimenticato - perdonatemi, ma vi sarà sicuramente ben nota la mia scarsa memoria! - e che mi sono state vicine in questi anni.

Bibliografia

ABI Research. 1.5 million home automation systems installed in the us this year, 2012. URL https://www.abiresearch.com/press/15-million-home-automation-systems-installed-in-th/.

Apple. Homekit, 2015. URL https://developer.apple.com/homekit/.

Sara Bevilacqua. Integrazione di social network in un sistema prototipale di home intelligence: il caso twitter, 2015. URL http://apice.unibo.it/xwiki/bin/view/Theses/HMSocialLM.

Roberta Calegari and Enrico Denti. Home manager, 2014. URL https://apice.unibo.it/xwiki/bin/view/Products/HomeManager.

Matteo Carano. Sperimentazione di tecnologie raspberry in contesti di home intelligence, 2015. URL http://apice.unibo.it/xwiki/bin/view/Theses/HMRaspberryLT.

Alessandro Celi. Integrazione di servizi meteo in un sistema prototipale di home intelligence, 2015. URL http://apice.unibo.it/xwiki/bin/view/Theses/HMSocialLM.

P. Ciancarini, A. Omicini, and F. Zambonelli. Coordination technologies for internet agents. 1999.

DeLonghi. Moka elettrica alicia, 2015. URL http://www.delonghi.com/it-it/prodotti/caffe/macchine-da-caffe/caffettiere-elettriche-moka/alicia-plus-emkp-42b-0132038010.

Enrico Denti. Novel pervasive scenarios for home management: the butlers architecture. page 52, 2014. URL http://www.springerplus.com/content/3/1/52.

Gamification Community. The gamification community portal, 2012. URL http://www.gamification.org.

J. Gerhart. *Home Automation and Wiring*. Complete construction. McGraw-Hill, 1999. ISBN 9780070246744. URL http://books.google.it/books?id=CR_D8aJU7ksC.

R. Harper. *Inside the Smart Home*. Springer, 2003. ISBN 9781852336882. URL http://books.google.it/books?id=SvcHvHuv86gC.

Ikea. Concept kitchen 2025, 2015. URL http://www.conceptkitchen2025.com/.

International Energy Agency. The integer millenium house, 1998. URL http://www.ecbs.org/docs/Annex_38_UK_Watford.pdf/.pdf.

iSmartAlarm. ismartalarm - protect your home intelligently, 2015. URL https://www.ismartalarm.com/en/.

Stefano Mariani and Andrea Omicini. TuCSoN coordination for MAS situatedness: Towards a methodology. In Corrado Santoro and Federico Bergenti, editors, *WOA 2014 – XV Workshop Nazionale "Dagli Oggetti agli Agenti"*, volume 1260 of *CEUR Workshop Proceedings*, pages 62–71, Catania, Italy, 24–26 September 2014. Sun SITE Central Europe, RWTH Aachen University.

Microsoft. L'approccio internet delle tue cose, 2015. URL https://dev.windows.com/it-it/iot.

Nest Labs. Welcome home. 2014. URL https://nest.com/blog/2014/01/13/welcome-home/.

Netatmo. The thermostat for smartphone, 2015. URL http://www.netatmo.com/it-IT/prodotto/thermostat.

A. Omicini and G. A. Papadopoulos. Editorial: why coordination models and languages in ai. 2001. doi: 10.1080/08839510150204581.

Andrea Omicini. Tuple based coordination, 2012.

Andrea Omicini and Stefano Mariani. The tucson coordination model and technology - a guide, 2015. URL http://www.slideshare.net/andreaomicini/the-tucson-coordination-model-technology-a-guide.

G. A. Papadopoulos and F. Arab. Coordination models and languages. 1998. doi: 10.1016/S0065-2458(08)60208-9.

Samsung. Climatizzatore ar9000 con smart wi-fi, 2015. URL http://www.samsung.com/it/consumer/home-appliances/air-conditioners/wall-mount/AR12HSSFAWKNEU.

SODA. Soda, 2012. URL https://apice.unibo.it/xwiki/bin/view/SODA/.

Dag Spicer. If you can't stand the coding, stay out of the kitchen: Three chapters in the history of home automation, 2000. URL http://www.drdobbs.com/architecture-and-design/if-you-cant-stand-the-coding-stay-out-of/184404040.

N. Tesla. Method of and apparatus for controlling mechanism of moving vessels or vehicles, 1898. URL http://www.google.com/patents/US613809. US Patent 613,809.

Treccani. Treccani, 2015. URL http://www.treccani.it/vocabolario/domotica/.

TuCSoN. Tucson home, 2015. URL http://apice.unibo.it/xwiki/bin/view/TuCSoN/.