Alma Mater Studiorum
Università di Bologna

Campus di Cesena
Scuola di Ingegneria e Architettura
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

# Home Manager come middleware per la coordinazione situata in ambito IoT

Tesi in
Sistemi Autonomi

*Relatore*
Prof. Andrea Omicini

*Candidato*
Richiard Casadei

*Correlatore*
Dott. Stefano Mariani

Anno Accademico 2015/2016 - Sessione II

Ai miei genitori,
ai miei nonni,
ai miei zii e alla mia cuginetta,
e alla mia ragazza
che hanno sempre creduto in me.

# Indice

# Introduzione

Negli ultimi anni la rete ha subito una straordinaria evoluzione e sempre più dispositivi sono connessi. Basti pensare che ad oggi vi sono più dispositivi connessi che esseri umani sul pianeta ed entro il 2020 si prospetta che ci saranno circa 30 miliardi di dispositivi in rete. Questa evoluzione è stata denominata come Internet of Things (IoT). Il grande e continuo sviluppo tecnologico ha portato alla creazione di dispositivi sempre più smart e connessi con la rete, con capacità di percezione dell'ambiente e possibilità di inviare le informazioni attraverso Internet fornendo una modalità semplice ed intuitiva di interazione tra i vari dispositivi e le persone. Questi dispositivi hanno inoltre posto i presupposti per la nascita di ambienti intelligenti e connessi finalizzati al miglioramento della vita dell'uomo. In un ambiente dotato di intelligenza (denominata *Ambient Intelligence, AmI*) i dispositivi diventano parte integrante dell'ambiente stesso attraverso l'utilizzo di connessioni, metodi di comunicazione ed interazione più semplici e costituiscono un supporto attivo per l'utente nelle operazioni di routine.

Un esempio sono i dispositivi e gli elettrodomestici che popolano le nostre case. Essi sono sempre più smart e posseggono la capacità di svolgere funzioni e compiti in modo parzialmente autonomo e programmati dall'utente (ad esempio impostare l'orario di accensione o spegnimento). Oggi, grazie all'evoluzione degli smartphone e alla diffusione della connessione in mobilità a costi minimi, è anche possibile controllare in remoto questi dispositivi avviando o terminando la loro attività attraverso applicazioni installabili su qualunque smartphone. Tutto ciò ha permesso inoltre la creazione di applicazioni e scenari fino a poco fa impensabili su larga scala nel mercato consumer.

Recenti proposte, come l'architettura Butlers, definiscono la smart home come uno scenario in cui il sistema deve essere in grado di interagire con i suoi abitanti, non solo per monitorare e in remoto controllare gli elettrodomestici, ma deve anche prendere in considerazione le abitudini degli utenti, il loro comportamento, la loro ubicazione e le loro preferenze per prendere decisioni autonome e possibilmente anticipare le esigenze degli utenti, attraverso la gestione automatica dei dispositivi domestici. Tali decisioni potrebbero essere influenzate anche dalla supplementare valutazione di informazioni raccolte esternamente al sistema stesso poichè ritenute rilevanti al fine di svolgere un compito all'interno della abitazione (ad esempio previsioni meteo per comandare tapparelle automatiche, ecc.). Perciò un ambiente con AmI deve poter essere in grado di recuperare questo tipo di informazioni attraverso la rete a cui sono collegati i suoi dispositivi e sfruttando in particolare servizi Cloud, che grazie alla loro natura dis-embodied e scalabile permettono di avere un flusso continuo di informazioni provenienti da diverse fonti ed un accesso on-demand a queste informazioni computate e visualizzabili in real-time.

Un esempio di possibile implementazione di un sistema di gestione di una smart home, con architettura Butlers, è fornito dal prototipo di Home Manager, basato sull'infrastruttura TuCSoN in cui si suppone la presenza di dispositivi intelligenti, cioè che abbiano modo di comunicare tra loro e che siano in grado di eseguire alcune operazioni, e nel quale l'organizzazione ed il modello adottato è quello di agenti, entità software con delle capacità decisionali e autonome.

Lo scopo di questa tesi è quello di apprendere i principali concetti dietro questi nuovi scenari entrando nelle definizioni di IoT, AmI, architettura Butlers e studiandone le caratteristiche e tutti i punti chiave che li contraddistinguono, per poi andare ad analizzare l'attuale prototipo Home Manager ed esternderlo con il concetto di coordinazione situata. Il lavoro che segue è così organizzato:

- Nel primo capitolo si introducono i concetti di Internet of Things (IoT) ed ambiente dotato di intelligenza (AmI) andando ad analizzarne le principali caratteriste e punti chiave, per continuare presentando il

modello prototipale Home Manager e l'architettura su cui si basa, l'architettura Butlers.

- Nel secondo capitolo si analizzano gli aspetti fondamentali dell'infrastruttura di coordinazione TuCSoN alla base di Home Manager e si prosegue evidenziandone le parti utilizzate in quest'ultimo.
- Nel terzo capitolo si introduce il concetto di *Transducer* all'interno del prototipo Home Manager andando ad analizzare lo stato attuale del sistema e fornendo poi una soluzione progettuale ed implementativa, permettendo così al sistema di estendere le sue funzionalità e di eseguire forme di coordinazione situata.
- Nel quarto capitolo si introduce il concetto di coordinazione spaziale e geolocalizzazione offerta dai centri di tuple TuCSoN e dalle reazioni spaziali offerte dal linguaggio di coordinazione ReSpecT andando prima ad analizzare come viene utilizzata la geolocalizzazione nel prototipo attuale di Home Manager e fornendo poi una soluzione progettuale ed implementativa, motivando le scelte effettuate ed i benefici ottenuti.

Infine sono delineate le conclusioni sul lavoro svolto ed inseriti alcuni possibili sviluppi futuri.

# Capitolo 1

# IoT, AmI e Home Manager

L'Internet delle cose (IoT) è un nuovo paradigma che sta rapidamente guadagnando terreno nello scenario delle moderne tecnologie. L'idea di base di questo concetto è la presenza pervasiva di una varietà di "cose" o oggetti come sensori, attuatori, telefoni cellulari, ecc che sono in grado di interagire tra loro e cooperare con i loro vicini per raggiungere obiettivi comuni. Tale paradigma ha creato anche la visione generale dell'ambiente con intelligenza (AmI), la quale prende in esame la possibilità di arricchire i luoghi comunemente frequentati dalle persone (abitazioni, uffici, ma anche città) con capacità di percezione, elaborazione e comunicazione e dispositivi le cui funzioni e compiti portino miglioramenti nella qualità di vita delle persone. Proprio per quanto riguarda le abitazioni sempre piú applicativi sono in fase di realizzazione e commercializzazione ed essi hanno lo scopo di fornire un supporto all'utente, permettendogli di monitorare lo stato dei dispositivi e fornendogli una sorta di controllo, anche in remoto, semi-automatizzato dell'abitazione. La visione di ambiente dotato di AmI però deve essere considerata piú completa e complessa infatti l'abitazione stessa, in quanto intelligente, deve poter aver capacità di interazione con i propri abitanti per mantenerli aggiornati sullo stato dei dispositivi e permetter loro il controllo di quest'ultimi. L'abitazione inoltre deve poter esser in grado di poter raccogliere informazioni, preferenze ed abitudini degli utenti che la abitano per poter agire autonomamente sui dispositivi presenti in essa, osservandone il loro comportamento tramite i dispositivi mobile in loro possesso. Il prototipo applicativo Home

Manager[1] si presenta proprio come sistema di casa intelligente, dove l'abitazione è vista come un ambiente smart fatto di dispositivi indipendenti che partecipano ad una società di agenti.

# 1 IoT

In questa sezione verrà introdotto il concetto di Internet of Things secondo la visione di diverse fonti e ne saranno analizzate le caratteristiche ed i principali modelli di comunicazione.

## 1.1 Un paradigma, diverse visioni

All'interno della comunità di ricerca e in letteratura troviamo differenti definizioni di Internet of Things. La ragione di ciò risiede proprio nella sintassi del termine stesso, che sintatticamente è composto dai due termini Internet e Things [1]. Il primo porta ad una visione Internet-oriented dell'IoT, mentre il secondo si focalizza sulle "cose" e porta ad una visione Things-oriented. Non deve essere dimenticato, in ogni caso, che le due parole, quando messe insieme, assumono un significato che introduce una fortissima innovazione nel mondo ICT. Semanticamente IoT significa una rete mondiale di oggetti interconnessi in modo univoco indirizzabili, sulla base di protocolli standard di comunicazione [2]. Tale definizione ci porta ad una terza visone dell'IoT, quella Semantic-oriented.
In Fig. 1.1, i concetti, le tecnologie e gli standard principali sono evidenziati e classificati con riferimento alla visione specifica dell'IoT. e da essa risulta chiaramente che tale paradigma coincida nella convergenza delle tre visioni principali affrontati sopra. La prima definizione di IoT deriva dal punto di vista Things-oriented in cui gli elementi considerati sono i principali standard progettati per migliorare la visibilità e la tracciabilità dell'oggetto (cioè il suo status, la posizione corrente, ecc). Essi comprendono Radio-Frequency Identification (RFID), Universal/Ubiquitous Identifier (UID), Near Field Communications (NFC), Wireless Sensor and Actuators Networks (WSAN), Wireless

[1] https://apice.unibo.it/xwiki/bin/view/Products/HomeManager

Identification Sensing Platform (WISP), Spime (un neologismo per indicare un oggetto virtuale caratteristico per l'IoT, che può essere monitorato attraverso lo spazio e il tempo per tutta la sua vita e i cui dati risiedono nel cloud), ecc.
Tale visione però non può che risultare incompleta se considerata da sola, infatti l'IoT può essere considerato come un'infrastruttura globale che collega oggetti generici virtuali e fisici. In questo senso, esso diventa l'architettura che consente la diffusione di servizi e applicazioni indipendenti, caratterizzate da un elevato grado di acquisizione autonoma dei dati, connettività di rete e interoperabilità. Questa definizione può essere considerata come un collegamento tra ciò che prima abbiamo indicato come la visione Things-oriented e quella Internet-oriented. All'interno di quest'ultima si collocano l'IP for Smart Objects (IPSO), Internet 0 (un livello fisico che ha lo scopo di ridurre la complessità dello stack IP per realizzare un protocollo per instradare "IP su tutto") ed il Web of Things (secondo la quale gli standard Web vengono riutilizzati per collegare e integrare nel Web tutti gli oggetti che contengono un dispositivo embedded o un computer).
La terza visione, quella Semantic-oriented, prende in considerazione tutti questi aspetti citati precedentemente e li colloca in una visione di base dove il numero degli elementi coinvolti, le tecnologie utilizzate e le comunicazioni tra di esse è destinato a diventare estremamente elevato. Le questioni relative al modo di rappresentare, di ricercare ed organizzare le informazioni generate dai dispositivi IoT diventerà molto impegnativo e le tecnologie semantiche potrebbero svolgere un ruolo chiave. Esse infatti potranno sfruttare soluzioni di modellazione appropriate per la descrizione degli oggetti e dispositivi, ragionando sui dati generati da essi, fornendo ambienti di esecuzione, architetture adatte alla struttura dei sistemi futuri e che permettano di soddisfare le esigenze dei sistemi IoT attraverso l'utilizzo di infrastrutture di comunicazione e stoccaggio dati scalabili e a volte addirittura trasparenti dal punto di vista dell'utente (il Cloud).

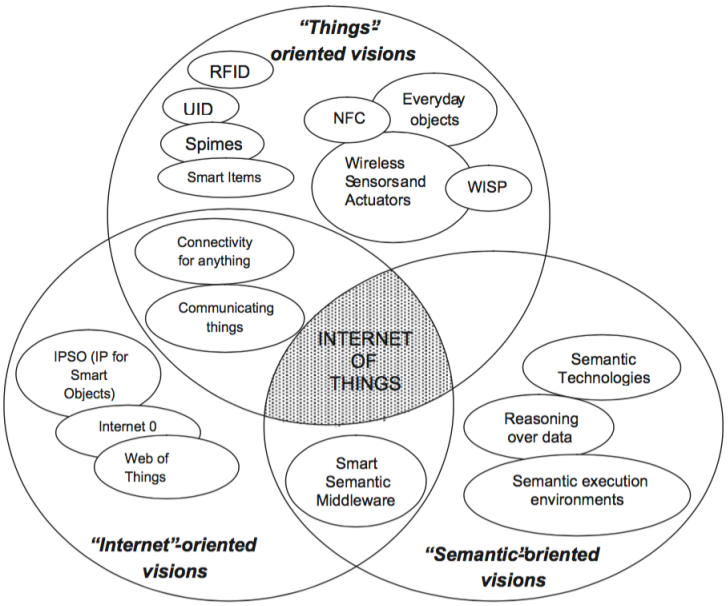


Figura 1.1: Il paradigma IoT come risultato della convergenza di differenti visioni

## 1.2 Caratteristiche

Dall'analisi delle visioni precedentemente presentate possiamo affermare che la grande ascesa delle diverse e nuove tecnologie previste dall'IoT e l'utilizzo sempre più costante della rete hanno reso possibile aver connessi tra di loro sempre più dispositivi, di natura eterogenea, ed in modo sempre più semplice ed economico per un utente finale. Il procedimento di crescita esponenziale dell'IoT infatti può essere riassunto da 6 punti chiave, che ne costituiscono le principali caratteristiche:

*Ubiquità della connessione* - la pervasività delle attuali connessioni di rete ha permesso di abbassarne i costi e l'evoluzione tecnologica ha permesso di avere una connessione e alte velocità di connessione su qualsiasi dispositivo;

*Diffusa adozione di reti IP-based* - il protocollo IP è diventato lo standard globale dominante per la rete, esso appartiene alla suite di protocolli TCP/IP e fornisce una piattaforma di instradamento di rete ben definita e ampiamente implementata da software e tool che possono essere incorporati in una ampia gamma di dispositivi, anche quelli più economici;

*Capacità di calcolo economica* - i grandi investimenti delle industrie nella ricerca, sviluppo e produzione, hanno permesso di avere un continuo aumento della potenza di calcolo prevista pur mantenendo basso (in alcuni casi anche abbassando) il relativo consumo energetico;

*Miniaturizzazione* - i progressi costruttivi dei processi CMOS permettono di incorporare la capacità di calcolo e la comunicazione in oggetti sempre più piccoli. Questo aspetto, associato alla capacità di calcolo a basso costo, ha incentivato l'ascesa di piccoli e economici sensori e processori di calcolo che ora sono alla base di molte applicazioni IoT;

*Avanzamento nell'analisi dei dati* - la nascita di nuovi algoritmi di analisi, il rapido aumento della potenza di calcolo e l'innovazione del calcolo in parallelo associato allo spazio di archiviazione e alla distribuzione sempre crescente dei servizi cloud ha favorito l'aggregazione e l'analisi di enormi quantità di dati fornendo nuove soluzioni per ricavare conoscenza da esse;

*Ascesa della computazione cloud* - il paradigma del Cloud Computing ha permesso di sfruttare una computazione ed un'elaborazione remota dei processi di lavoro, gestione e archiviazione di dati e ha dato la possibilità a piccoli dispositivi distribuiti in rete di collegarsi ed interagire attraverso un back-end con potenti e scalabili capacità analitiche di calcolo.

## 1.3 Modelli di comunicazione

L'Internet Architecture Board (IAB) ha rilasciato nel marzo del 2015 il documento RFC 7452 [3] che spiega l'architettura di rete degli smart object e delinea quattro principali modelli di comunicazione utilizzati dai dispositivi IoT negli attuali sistemi in commercio.

### Comunicazione Device-to-Device

Nel modello di comunicazione Device-to-Device sono presenti due o più dispositivi connessi direttamente tra loro e che hanno la capacità di comunicare attraverso diversi tipi rete (reti IP o Internet) senza la presenza di un server che faccia da intermediario. Spesso questo tipo di comunicazione utilizza

protocolli come Bluetooth, Z-Wave, o ZigBee per stabilire una connessione device-to-device come mostrato nella figura sotto.

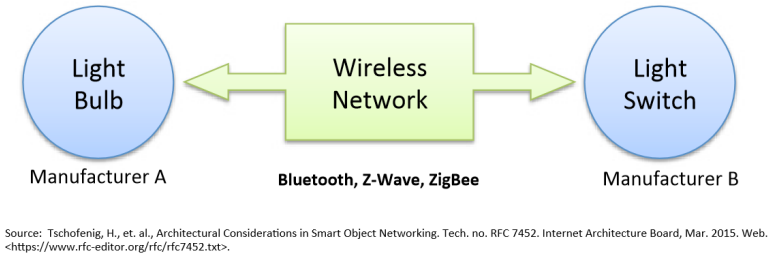


Figura 1.2: Modello di comunicazione Device-to-Device.

Questo modello di comunicazione è comunemente usato in scenari come home automation, nei quali la comunicazione tra i dispositivi tipicamente avviene mediante scambio di piccoli pacchetti e nel quale esiste una diretta relazione tra tutti i dispositivi coinvolti.
Dal punto di vista dell'utente questo tipo di comunicazione può portare a problemi di compatibilità, poiché l'utente è vincolato a scegliere una famiglia di dispositivi che utilizza uno specifico protocollo. Per esempio i dispositivi che utilizzano Z-Wave non sono nativamente compatibili con quelli della famiglia ZigBee. Questo vincolo però può essere anche inteso come un punto a favore perchè scegliendo una famiglia di dispositivi compatibili si ha la certezza che essi comunichino in maniera efficiente tra di loro.

### Comunicazione Device-to-Cloud

Nel modello di comunicazione Device-to-Cloud i vari dispositivi IoT si connettono direttamente, tramite metodi di comunicazione tradizionali come Wi-Fi o Ethernet, ad un servizio offerto da una piattaforma Cloud.
La connessione con il cloud permette spesso all'utente di utilizzare il dispositivo anche da remoto tramite smartphone o tramite interfaccia web, tuttavia anche in questo modello possono nascere problemi di incompatibilità quando si cerca di integrare dispositivi di differenti produttori. Molto spesso, a causa della moltitudine di aziende sul mercato e dei provider di servizi Cloud, i dispositivi ed il servizio Cloud non sono dello stesso produttore e questo limita fortemente l'utente perchè lo vincola ad utilizzare dispositivi dello stesso

produttore compatibili con la piattaforma del servizio Cloud scelta, oppure utilizzare app o interfacce web differenti per ogni singolo dispositivo di marca differente.

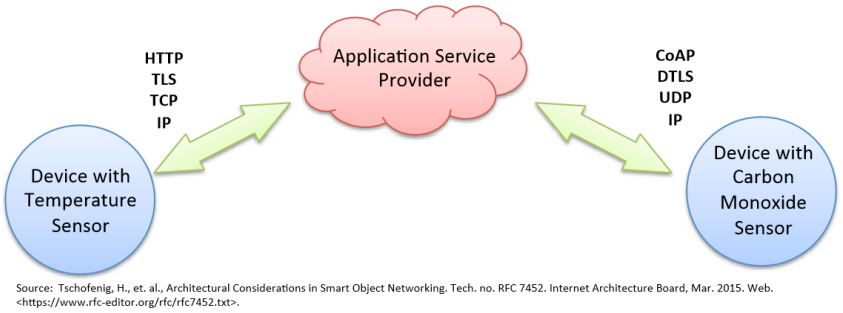


Figura 1.3: Modello di comunicazione Device-to-Cloud.

### Comunicazione Device-to-Gateway

Nel modello di comunicazione Device-to-Gateway, il dispositivo IoT si connette al servizio Cloud mediante un Gateway, il quale ricopre quindi il ruolo di intermediario e ricoprendo tre ulteriori ruoli principali:

- Il primo ruolo è quello di trasformare e normalizzare i dati. Un semplice sensore spesso non possiede anche capacità di computazione e l'unica cosa che può fare è restituire il dato "letto", sarà poi compito del gateway completare l'informazione con dati aggiuntivi utili alla sua interpretazione. I dati generati dai sensori possono anche essere in diversi formati e quindi è il gateway che si occupa di normalizzarli in un unico formato. Il Gateway ha quindi il compito di acquisire dati eterogenei, di completarli se necessario e infine di convertirli in un formato standard.
- Il secondo ruolo del Gateway è quello di supportare diversi protocolli di comunicazione. Come detto precedentemente l'eterogeneità dei dispositivi IoT porta ad avere un utilizzo di protocolli di comunicazione differenti. Il Gateway deve quindi supportare protocolli per le connessioni entranti ed uscenti. Alcuni dei più popolari protocolli usati in questo contesto sono: ReST, MQTT, CoAP, STOMP e anche SMS.

- Il terzo ed ultimo ruolo è quello che riguarda la gestione della sicurezza del sistema poichè consiste nel dispositivo di confine del sistema e deve proteggere i dispositivi IoT dalla rete pubblica, incrementando così notevolmente la sicurezza.

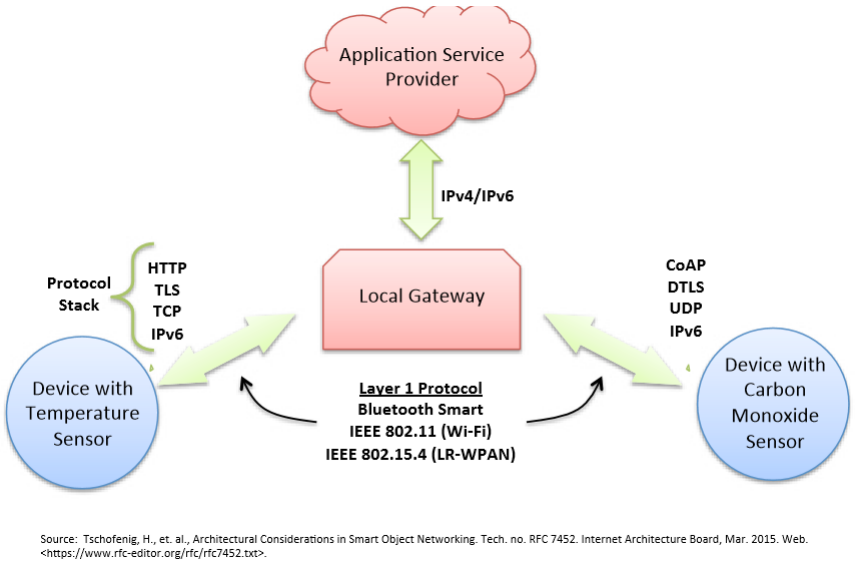


Figura 1.4: Modello di comunicazione Device-to-Gateway.

Questo modello di comunicazione sembra essere ultimamente il più utilizzato in molti scenari.

### Modello Back-End Data-Sharing

Il modello di condivisione dei dati di back-end si riferisce ad una architettura di comunicazione che consente agli utenti di esportare e analizzare i dati dei dispositivi IoT da un servizio Cloud in combinazione con i dati provenienti da altre fonti. Questo approccio è un'estensione del modello di comunicazione single device-to-Cloud, dove il dispositivo IoT carica i propri dati ad un unico fornitore di servizi applicativi. Consiste in una architettura di back-end di condivisione che permette ad un singolo flusso di dati trasmessi da un dispositivo IoT di essere aggregato ed analizzato assieme ad altri dati provenienti da altre fonti.

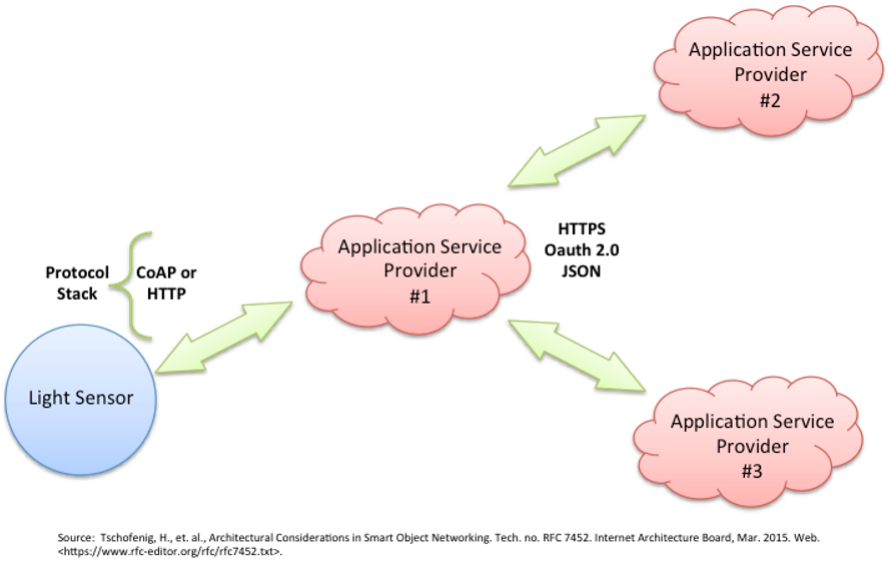


Figura 1.5: Modello Back-End Data-Sharing

# 2 AmI

La rapida ascesa dell'IoT ha permesso lo sviluppo di scenari in cui i dispositivi utilizzati sono sempre più smart portando a pensare l'ambiente in cui sono situati come un ambiente dotato di intelligenza (AmI). Esso consiste in un ambiente digitale in cui i dispositivi cooperano per fornire un supporto alle persone durante lo svolgimento delle loro attività quotidiane, fornendo una forma di interazione con essi il più possibile semplice ed intuitiva.
In questa sezione ne analizzaremo le caratteristiche principali.

## 2.1 Caratteristiche

In un ambiente con AmI le persone sono circondate da interfacce intelligenti ed intuitive inserite nei dispositivi distribuiti nell'ambiente. Queste interfacce riconoscono la presenza ed il comportamento di un determinato individuo, reagiscono in modo pertinente e personalizzato e servono l'utente in modo flessibile, adattandosi a situazioni diverse ed eterogenee [4].
AmI ha origine dalla convergenza di tre concetti chiave: *ubiquitous computing, ubiquitous communication ed intelligent user interface (IUI).*

Per ubiquitous computing si indica la progressiva integrazione di capacità computazionale nei dispositivi dell'ambiente i cui punti di accesso sono distribuiti in tutto l'ambiente. Per ubiquitous communication si denota la capacità che hanno tali dispositivi di comunicare tra loro e con l'utente, e in questo senso le reti wireless occupano un ruolo fondamentale e sono largamente utilizzate per supportare dispositivi mobili ed accessi remoti rendendo la tecnologia utilizzata estremamente integrata nell'ambiente, quasi trasparente dal punto di vista dell'utente. Le intelligent user interface IUI sono interfacce che hanno il ruolo di fornire un supporto più semplice e naturale all'interazione utente-dispositivo mediante il supporto di input e output multimodale, come gestures ed interazioni vocali. L'utente dovrebbe poter interagire con i servizi virtuali forniti dal sistema AmI e dai suoi dispositivi, come se essi fossero di oggetti fisici e dovrebbe comunicare in modo intuitivo e comprensibile come se parlasse con un altro essere umano.
Date tali caratteristiche le tecnologie utilizzate all'interno di un ambiente con AmI, perciò, dovranno essere:

- *Embedded* - i dispositivi necessitano di essere integrati nell'ambiente;
- *Context-aware* - i dispositivi devono possedere le capacità di riconoscere gli utenti ed il contesto in cui sono situati;
- *Personalizzate ed adattive* - scopo dei dispositivi di un AmI è quello di soddisfare le necessità degli utenti al suo interno ma esse saranno, presumibilmente, diverse da utente ad utente;
- *Anticipative* - i dispositivi dovrebbero riuscire ad anticipare le necessità dell'utente, senza attendere una richiesta esplicita. È necessario che il sistema abbia una profonda conoscenza delle preferenze e delle attività abituali dell'utente.

La visione di AmI come un sistema in grado di percepire, soddisfare ed anticipare i bisogni e le necessità dell'utente risulta dunque fondata principalmente su due aspetti fondamentali:

*Anticipazione* - un AmI può essere visto come un meta-ambiente, esso deve includere gli strumenti per anticipare le necessità delle entità presenti al suo interno. L'individuazione dei desideri dell'utente deve essere

basata su pattern di comportamento ricavati tramite lo studio del suo uso dei dispositivi mobili ad esempio ricavando informazioni sulla sua posizione, grazie alla connettività situata abilitata mediante il GPS e le altre tecniche di geolocalizzazione.

*Adattamento/Apprendimento* - AmI è un sistema in continua evoluzione che ha come obiettivo primario anticipare le richieste degli utenti. È necessario quindi che esso sia in grado di rilevare anche nuove tipologie di eventi o comportamenti per migliorare la sua capacità di capire le esigenze degli individui e proporre soluzioni adatte. In molti scenari vengono utilizzati anche feedback da parte degli utenti per raggiungere livelli ottimali di correttezza delle previsioni.

# 3 Home Manager

L'obiettivo di questa sezione è quello di introdurre l'architettura Butlers e offrire una panoramica dell'attuale prototipo Home Manager[2] senza soffermarsi in dettaglio sulle scelte progettuali e sul punto di vista implementativo complessivo.

## 3.1 L'architettura Butlers

L'architettura Butlers [5] definisce un framework con 7 layer concettuali di riferimento per sistemi domotici che consente di mettere in relazione le caratteristiche e le tecnologie presenti in un'applicazione con il valore aggiunto percepito dall'utente. Butler, dall'inglese maggiordomo, consiste in un componente specializzato in una certa attività e con alto livello di intelligenza che possiede la capacità di imparare in modo autonomo le esigenze e le preferenze di un utente, osservandone il comportamento o interagendo con altri Butler. In questo modo il Butler, una volta acquisite le informazioni rilevanti sui dispositivi presenti in casa, le preferenze e le necessità dell'utente, è in grado di anticipare i bisogni di quest'ultimo.

[2] https://apice.unibo.it/xwiki/bin/view/Products/HomeManager

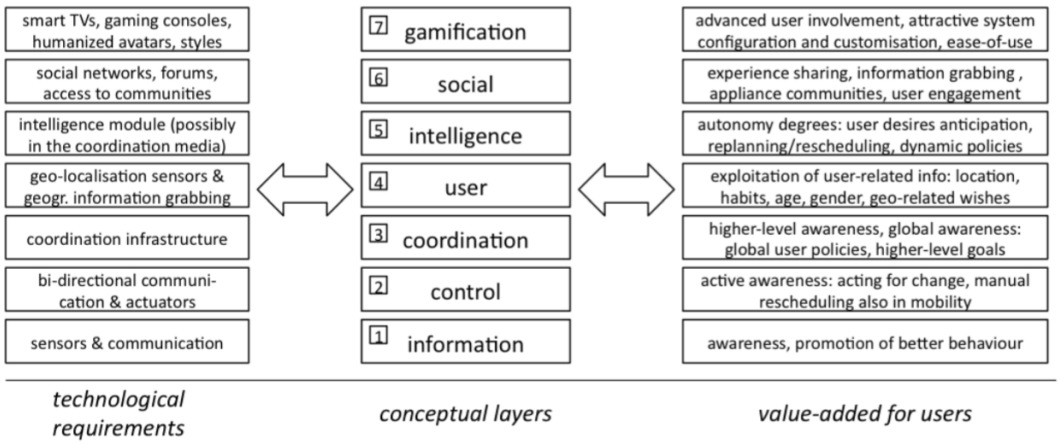


Figura 1.6: L'architettura multi-layer Butlers.

Come rappresentato dalla Fig. 1.6 i livelli più bassi previsti dall'architettura Butlers riguardano le tecnologie a disposizione, i livelli intermedi consistono in strati infrastrutturali ed infine i livelli più alti riguardano intelligenza, aspetti sociali e di gamification.
In dettaglio:

Il livello più basso, quello di *Information*, è quello che utilizza le informazioni di tipo fisico estrapolandole dai dispositivi embedded e permettendo di ottenere informazioni, come ad esempio, i consumi energetici attuali della casa;

Il secondo livello, quello di *Control*, aggiunge una forma di controllo remoto, accessibile tramite app Android o iOS, permettendo di introdurre l'idea di casa automatizzata;

Il terzo livello, quello di *Coordination*, permette di introdurre politiche di coordinazione e comunicazione tra i componenti all'interno dell'abitazione;

Il quarto livello, quello *User*, introduce rispetto al livello sottostante, la conoscenza dell'utente, come la sua posizione (acquisita sfruttando strumenti di geo-localizzazione), i suoi dati anagrafici, le preferenze e le

abitudini. Tutte queste informazioni sono utilizzate per decidere e coordinare le attività su misura per l'utente in questione. In questo livello non è presente ancora una forma di intelligenza, intesa come la capacità di anticipare le possibili richieste e necessità dell'utente. Le decisioni in questo livello sono prese solo seguendo le regole incorporate al sistema di coordinazione e apprese durante il periodo di configurazione;

Il quinto livello, quello di *Intelligence*, introduce l'agente Butler intelligente. Esso presenta tutte le caratteristiche descritte in precedenza e aggiunge ad esse la capacità di anticipare le decisioni e i desideri degli utenti tramite tutte le risorse e informazioni utili presenti all'interno del sistema. Esso è inoltre in grado di risolvere autonomamente eventuali problematiche e proporne soluzioni;

Il sesto livello, quello *Social*, introduce la possibilità di integrare i Butlers con i social network, sia per accedere ad ulteriori informazioni sull'utente, sia per migliorare le proprie prestazioni scambiando informazioni con altri Butlers di altre abitazioni;

Il settimo ed ultimo livello, quello di *Gamification*, prevede la possibilità del Butler di coinvolgere gli utenti attraverso forme di divertimento, prendere decisioni complesse e conoscere gli abitanti della casa.

Gli ultimi 3 livelli (*Intelligence, Social e Gamification*) insieme alla conoscenza delle abitudini dell'utente e della sua posizione, permettono di andare oltre la pura gestione della casa. Muovendosi verso i livelli alti del framework si possono ipotizzare architetture in cui il sistema agisce come un vero e proprio maggiordomo intelligente. Il Butler possiede specifiche competenze in una determinata area e osservando il comportamento di un utente ed interagendo con gli altri Butler presenti ha la capacità di imparare ad anticipare le necessità di tale utente, a patto che siano disponibili tutte le informazioni sui dispositivi interni alla casa e sull'utente.
Questa architettura è definita in modo generale e technology-independent così da poter tener conto di ogni possibile sistema.

## 3.2 Il prototipo attuale

Home Manager è un prototipo di applicazione per il controllo di una casa intelligente, progettato come un sistema multi-agente tramite la metodologia SODA (Societies in Open Distributed Agent environments) che adotta il metamodello Agents & Artifacts (A&A), e implementato poggiandosi al di sopra dell'infrastruttura di coordinazione TuCSoN[3] [6].
Il sistema considera una casa con dispositivi indipendenti (condizionatori d'aria, luci, ecc) ciascuno dotato di un agente. L'infrastruttura di coordinazione, programmabile tramite centri di tuple, incorpora le leggi di coordinazione che hanno il fine sia di mediare tra le diverse preferenze dell'utente sia di perseguire gli obiettivi generali del sistema (ad esempio per gestire/-limitare il consumo complessivo di energia). Più di recente, il sistema Home Manager è stato reinterpretato, dati i suoi obiettivi e le caratteristiche, nella prospettiva dell'architettura Butlers. In particolare, sfruttando la posizione dell'utente, monitorata in tempo reale grazie al GPS e le altre tecniche di geo-localizzazione integrate negli smartphone moderni, si è voluto consentire ad un agente intelligente di prendere alcune decisioni in modo del tutto autonomo (per esempio, la regolazione della temperatura del condizionatore d'aria), e di anticipare anche le esigenze dell'utente gestendo direttamente i dispositivi per conto dell'utente stesso (per esempio, dedurre la possibilità di accendere il forno, o posticipare l'accensione della lavatrice, ecc).

Applicando concretamente l'idea di AmI ad un ambiente domestico Home Manager ha quindi lo scopo di di supportare l'utente nella gestione di una tipica abitazione famigliare (gestione della temperatura, gestione delle luci, gestione degli elettrodomestici ecc.) cercando di soddisfarne le richieste o anticipandone i bisogni basandosi sulla conoscenza delle preferenze dell'utente e sulle regole impostate dall'amministratore del sistema abitativo. Inoltre permette all'utente di inviare comandi diretti ai dispositivi/elettrodomestici e di tenere sotto controllo il consumo energetico.
Nel prototipo sviluppato è stato ipotizzato che ogni locale sia dotato di sensori, i quali permettono di identificare i soggetti che entrano ed escono dalle

[3] http://apice.unibo.it/xwiki/bin/view/TuCSoN/WebHome

rispettive stanze, e di appositi terminali per permettere un'identificazione esplicita dell'utente. Gli utenti identificati nel sistema sono suddivisi in abitanti della casa, che possono esprimere preferenze ed ai quali viene fornita tutta l'assistenza possibile e semplici visitatori, che non posseggono nessun privilegio ma gli è comunque garantita un'assistenza di base e di cui viene notificata la presenza. A loro volta gli abitanti della casa sono suddivisi in amministratori e utenti ordinari. I primi hanno il pieno controllo sul sistema, potendo specificare tutte le politiche di gestione e i privilegi associati ad ogni utente, i secondi invece possono esclusivamente esprimere le loro preferenze ed impartire comandi senza poter agire sugli altri utenti e sul sistema. Questa distinzione in ruoli permette l'integrazione efficace di un sistema di sicurezza RBAC (Role-Based Access Control), in cui l'accesso alle risorse dipende dal ruolo che si riveste: si esprimono dei vincoli di accesso/utilizzo alle risorse, in questo caso elettrodomestici e/o locali, associandole ad un ruolo (associazione ruolo-risorsa) e, successivamente, assegnando uno o più ruoli agli utenti (associazione ruolo-utente), prestando attenzione al fatto che un utente può rivestire più ruoli, ma non contemporaneamente. Se due o più utenti si trovano nella stessa stanza potrebbero sorgere dei conflitti nelle preferenze preimpostate in quanto il sistema cerca di soddisfare i bisogni di tutti, ma ciò viene evitato tramite le politiche impostate dall'amministratore, che permettono al sistema di prendere decisioni al fine di soddisfare tutti gli utenti che si trovano all'interno della stessa stanza.

In Home Manager è prevista inoltre la gestione della casa da remoto poichè è sviluppata anche una versione mobile che, comunicando con il prototipo Home Manager fisso su pc/server permette l'accesso a tutte le informazioni dei dispositivi registrati e consente all'utente di monitorare e gestire tali dispositivi dal suo smartphone. L'utente infatti, dopo essersi autenticato nell'app, attraverso la schermata di gestione potrà scegliere quale dispositivo monitorare o sul quale agire, in tale schermata infatti l'applicazione mostrerà lo stato del dispositivo in tempo reale, permettendo all'utente di apportare modifiche in base alle proprie esigenze. Queste modifiche saranno poi propagate al sistema e disponibili a tutti gli utenti fissi e mobili.

Grazie alle tecniche di geo-localizzazione, come il GPS, incorporate nei moderni smartphone, è inoltre possibile rilevare la posizione dell'utente. Questo

permette al sistema di avere informazioni più dettagliate sulle abitudini dell'utente per poter decidere, in modo autonomo, sulla gestione dei dispositivi, ed informare l'utente della decisione presa mediante notifica nell'app mobile. Tale decisione potrà esser sempre modificata dall'utente se non ritenuta opportuna.

# Capitolo 2

# TuCSoN e TuCSoN in Home Manager

TuCSoN (Tuple Centres Spread over the Network)[1] è un'infrastruttura per la coordinazione di processi distribuiti e di agenti autonomi, intelligenti e mobili [7]. Alla base di esso troviamo delle astrazioni di coordinazione e comunicazione dette centri di tuple, ovvero spazi di tuple il cui comportamento descrive leggi specifiche di coordinazione, programmabili tramite il linguaggio ReSpecT. In questa infrastruttura lo spazio di interazione, ovvero lo spazio dove vengono effettuati il consumo degli eventi di richiesta, la produzione di eventi di risposta e le attività di coordinazione, è rappresentato dai centri di tuple distribuiti sui nodi della rete Internet ed utilizzati dagli agenti per interagire e coordinarsi con altri agenti remoti.
Esso rappresenta una tipologia di infrastruttura adatta per Home Manager poichè permette di mediare le azioni intraprese dai vari Butler col fine di soddisfare le diverse preferenze dell'utente e di perseguire gli obiettivi generali del sistema. TuCSoN inoltre implementa RBAC-MAS [8], una versione di RBAC in cui le questioni di organizzazione e di sicurezza sono gestite in modo uniforme come problemi di coordinamento. Esiste un apposito centro di tuple ($ORG) che contiene le regole dinamiche di RBAC in TuCSoN e viene utilizzato in Home Manager per gestire i ruoli degli utenti presenti nel sistema.

[1] http://apice.unibo.it/xwiki/bin/view/TuCSoN/WebHome

# 1 Modello di coordinazione TuCSoN

In questa sezione si andranno ad analizzare gli aspetti fondamentali che caratterizzano quest'infrastruttura, cercando di evidenziarne i punti di forza.

## 1.1 Caratteristiche

Le entità che caratterizzano un sistema TuCSoN sono [9]:

- *Agenti* TuCSoN - sono entità pro-attive ed intelligenti che interagiscono tra di loro scambiando tuple attraverso i centri di tuple. L'interazione avviene mediante l'uso delle primitive di coordinazione offerte da TuCSoN. Essi hanno anche la caratteristica di essere mobili quindi non sono associati in modo permanente ad un particolare device e sono distribuiti nella rete;
- *Centri di tuple* ReSpecT - forniscono lo spazio condiviso tra gli agenti per la loro comunicazione tuple-based (spazio di tuple), insieme allo spazio di comportamento programmabile per la coordinazione;
- *Nodi* TuCSoN - rappresentano l'astrazione topologica di base che ospita al suo interno i centri di tuple.

Ad ogni nodo, centro di tuple ed agente è associato un identificatore univoco all'interno di un sistema TuCSoN.
Ogni nodo è identificato dalla coppia $\langle NetworkId, PortNo \rangle$, dove $NetworkId$ corrisponde all'indirizzo IP o alla entry DNS del device ospitante il nodo e $PortNo$ è il numero di porta su cui il servizio di coordinazione TuCSoN è in ascolto di una richiesta. In TuCSoN un singolo device può ospitare una molteplicità di nodi e ciascuno di essi sarà definito da `netid:portno`.
Un nome amissibile per un centro di tuple è identificato univocamente da `tname @ netid:portno` dove la coppia `netid:portno` rappresenta il nodo a cui appartiene il centro di tuple.
Ciascun agente, nel momento in cui entra in un sistema TuCSoN, è invece identificato da un nome comune `aname` e da un UUID (*Universally Unique Identifier*) assegnatogli dal middleware in modo da distinguerlo da qualsiasi altro agente del sistema. Il nome completo è quindi formato da `aname:uuid`.

Facendo riferimento ai concetti e alla terminologia introdotti in [10], le principali e più rilevanti caratteristiche di TuCSoN riguardano:

Il *linguaggio di coordinazione* TuCSoN - che permette agli agenti di interagire con i centri di tuple mediante l'esecuzione di operazioni di coordinazione costituite da primitive di comunicazione ben definite;

Lo *spazio di coordinazione* TuCSoN - con la sua duplice interpretazione sia come spazio di interazione globale sia come collezione di spazi di interazione locali;

Il *mezzo di coordinazione* TuCSoN - che rappresenta le astrazioni di comunicazione definite in modo da incorporare le leggi di coordinamento globale e fornire un comportamento ben definito.

Lo spazio di coordinazione sostiene efficacemente il ruolo degli agenti che individuano e accedono ai dati e alle risorse Internet, e possono trasferire la propria esecuzione su un sito in cui essi interagiscono con le risorse locali. Il mezzo di coordinazione arricchisce il modello di coordinazione, fondamentalmente data-oriented, con la flessibilità e il controllo necessari per affrontare la complessità delle applicazioni Internet.

## 1.2 Linguaggio di coordinazione

TuCSoN fornisce un linguaggio di coordinazione, costituito da un insieme di primitive ben definite, per permettere agli agenti di interagire con i centri di tuple. Esso consente agli agenti di leggere, scrivere e consumare tuple nel centro di tuple e di sincronizzarsi con esso. Il linguaggio di comunicazione è costituito da linguaggi di tuple e template di tuple logic-based, TuCSoN infatti utilizza il centro di tuple ReSpecT (anch'esso logic-based) come mezzo di coordinazione.
Ogni operazione di coordinazione è descritta da due principali fasi: una fase di invocazione (*invocation*), dove si esegue l'invio della richiesta da parte dell'agente contenente tutte le informazioni necessarie verso il centro di tuple designato, e da una fase di completamento (*completion*), durante la quale il risultato dell'operazione invocata sul centro di tuple ritorna all'agente richiedente includendo tutte le informazioni sull'operazione eseguita.

La sintassi astratta di un'operazione `op` invocata da un agente su un dato centro di tuple è:

```
tcid ? op
```

dove `tcid` è l'identificatore del centro di tuple. Siccome `tcid` può essere sia un nome assoluto sia un nome relativo del centro di tuple, gli agenti possono adottare due differenti forme di invocazione delle primitive:
La prima `tname @ netid : portno ? op` è utilizzata dagli agenti quando essi agiscono come entità network-aware e necessitano di indicare il centro di tuple attraverso il suo nome assoluto nello spazio di interazione TuCSoN globale.
La seconda invece è rappresentata dalla forma generale `tcid ? op` e viene utilizzata dagli agenti quando si comportano come componenti locali del loro ambiente di hosting.
Di seguito sono elencate le varie primitive di coordinazione presenti nella più recente formalizzazione del modello TuCSoN [9].

### Primitive di base

Il linguaggio di coordinazione TuCSoN fornisce le seguenti primitive di coordinazione per costruire operazioni di coordinazione:

- `out, rd, in`
- `rdp, inp`
- `no, nop`
- `get, set`

`out(`*Tuple*`)` - scrive la tupla *Tuple* nello spazio di tuple specificato, in caso di successo dell'operazione la tupla è ritornata come valore di ritorno.

`rd(`*TupleTemplate*`)` - ricerca se esiste una tupla che ha una corrispondenza col *TupleTemplate*, se una tupla *Tuple* fa il matching quando l'operazione è servita essa ritorna *Tuple*, altrimenti, viene sospesa, per poi esser ripresa e completata con successo quando sarà trovata una tupla con una corrispondenza nello spazio di tuple.

`in`(*TupleTemplate*) - ricerca una tupla che abbia una corrispondenza col *TupleTemplate*, se una tupla *Tuple* fa il matching, il successo dell'esecuzione rimuove *Tuple* dallo spazio di tuple e poi la ritorna come risultato, in caso contrario, l'esecuzione dell'operazione viene sospesa per poi esser ripresa e completata con successo quando sarà trovata una tupla con una corrispondenza nello spazio di tuple, la tupla anche in questo caso verrà rimossa.

`rdp`(*TupleTemplate*) - ricerca se una tupla ha una corrispondenza col *TupleTemplate*, se una tupla *Tuple* fa il matching quando l'operazione è servita essa ritorna *Tuple*, altrimenti, l'esecuzione dell'operazione fallisce e viene inviato come valore di ritorno il *TupleTemplate*.

`inp`(*TupleTemplate*) - ricerca una tupla che abbia una corrispondenza col *TupleTemplate*, se una tupla *Tuple* fa il matching, il successo dell'esecuzione rimuove *Tuple* dallo spazio di tuple e poi la invia come valore di ritorno, in caso contrario, l'esecuzione dell'operazione fallisce e viene ritornato il *TupleTemplate*.

`no`(*TupleTemplate*) - ricerca una tupla che abbia una corrispondenza col *TupleTemplate* nello spazio di tuple. Nel caso in cui non vi sia alcuna corrispondenza, viene inviato il *TupleTemplate* come valore di ritorno, altrimenti, l'esecuzione viene sospesa, per poi esser ripresa e terminata con successo nel momento in cui il matching non sarà trovato per nessuna tupla dello spazio di tuple, il *TupleTemplate* verrà inviato poi come valore di ritorno.

`nop`(*TupleTemplate*) - controlla se una tupla ha una corrispondenza col *TupleTemplate*. Nel caso in cui non vi sia alcuna corrispondenza l'operazione ha successo e il *TupleTemplate* viene ritornato, in caso contrario se una tupla *Tuple* fa il matching, l'esecuzione dell'operazione fallisce e verrà ritornata la *Tuple* che ha fatto fallire l'operazione.

`get` - legge tutte le *Tuple* dallo spazio di tuple specificato e le ritorna come una lista, se nello spazio di tuple non è presente nessuna tupla nel momento dell'esecuzione, l'operazione si considera comunque terminata con successo e viene ritornata una lista vuota;

`set(`*Tuples*`)` - riscrive lo spazio di tuple con la lista di *Tuples* specificata come parametro e quando l'esecuzione dell'operazione è terminata, la lista di *Tuples* viene inviata come valore di ritorno.

## 1.3 Architettura

L'architettura di un sistema TuCSoN è caratterizzata dall'insieme dei nodi (possibilmente distribuiti) su quali è presente un servizio TuCSoN [9]. Un servizio TuCSoN si mette in ascolto di possibili richieste in arrivo su una porta dell'host collegato alla rete che lo ospita. Diversi nodi TuCSoN possono funzionare sullo stesso host, mettendosi ciascuno di essi in ascolto su una differente porta di rete. In TuCSoN il numero di porta di default è 20504 quindi se un agente invoca un'operazione del tipo

```
tname @ netid ? op
```

senza specificare il numero di porta `portno` l'operazione `op` sul centro di tuple `tname` verrà invocata sul nodo di default `node netid : 20504` ospitato dall'host `netid`.
All'interno di ogni nodo TuCSoN è presente una collezione di centri di tuple resi disponibili ed interrogabili attraverso un nome `tname` valido. Ogni operazione di coordinazione può essere invocata su ogni centro di tuple appartenente ad un qualsiasi nodo TuCSoN, quindi agli agenti è fornito un completo spazio di coordinazione. Ogni nodo definisce inoltre un centro di tuple di default, chiamato `default`, il quale risponderà qualsiasi operazione richiesta e ricevuta dal nodo senza che non sia specificato il centro di tuple destinatario.

TuCSoN sfrutta i centri tuple come mezzi di coordinazione. Il comportamento di ogni singolo centro di tuple può essere definito separatamente e indipendentemente da qualsiasi altro centro di tuple in base agli specifici compiti di coordinazione scelti. Il singolo comportamento è naturalmente definito come la transizione di stato osservabile dopo un evento di comunicazione quindi per definire un nuovo comportamento è necessario specificare una nuova transizione di stato in risposta ad un evento di comunicazione standard. Ciò

si ottiene consentendo la definizione di reazioni (*reaction*) di comunicazione attraverso un linguaggio di specifica di reazione [11].
A qualsiasi delle primitive di TuCSoN è associabile un linguaggio di reazione che permette di specificare le attività computazionali relative a tale primitiva ed esse saranno chiamate *reaction*. Le *reaction* vengono definite come un insieme di operazioni non bloccanti il cui successo può atomicamente produrre effetti sullo stato del centro di tuple, mentre un loro fallimento non produce alcun risultato. Ogni *reaction* può liberamente accedere e modificare le informazioni raccolte in una tupla di un centro di tuple, e può accedere a tutte le informazioni relative all'evento di comunicazione invocato tramite la primitiva. Ogni evento di comunicazione può innescare una molteplicità di reazioni, tuttavia, siccome esse sono conseguenza di un singolo evento di comunicazione vengono tutte eseguite in una sola transizione di stato del centro di tuple.
Dal punto di vista degli agenti il risultato della chiamata di una primitiva è la somma degli effetti della stessa primitiva e di tutte le reazioni che essa ha innescato, e che complessivamente viene percepita come un'unica transizione di stato del centro di tuple su cui è stata invocata la primitiva. Tali nozioni sono adottate nei centri di tuple di TuCSoN attraverso il modello ReSpecT[2].

### Agent Coordination Context

Nel modello di coordinazione TuCSoN un *Agent Coordination Context* fornisce agli agenti una visione del loro spazio di coordinazione come collezione dei centri di tuple del sistema. Gli ACC, assieme ai centri di tuple, consistono nelle astrazioni che permettono a TuCSoN di gestire in modo uniforme i problemi di coordinazione, organizzazione e sicurezza. Esso ricopre il ruolo di mediatore dell'interazione in quanto un agente per poter invocare un'operazione su un determinato nodo dovà utilizzare "l'interfaccia" delle operazioni disponibili fornitagli dall'ACC di quello specifico nodo. Dualmente il sistema dei nodi interagisce con gli agenti solo tramite l'ACC e nel momento in cui un agente rilascia il suo contesto, dal punto di vista del sistema esso smette di esistere. Tramite la sua configurazione un Agent Coordination Context

[2]http://apice.unibo.it/xwiki/bin/view/ReSpecT/

permette di specificare un insieme di regole di base per definire le possibili operazioni effettuabili sul sistema. Infatti un ACC non è vincolato a fornire ad un agente una visione dell'intero spazio di interazione costituito da tutto l'insieme dei nodi disponibili e dei rispettivi centri di tuple, anzi può, in certi casi, anche bloccare totalmente l'accesso verso alcuni centri di tuple, impedire solo certe operazioni, limitare l'accesso o la semplice lettura delle informazioni appartenenti a certi centri di tuple. L'Agent Coordination Context rappresenta in questo senso il contratto tra l'agente e il sistema TuCSoN [9].

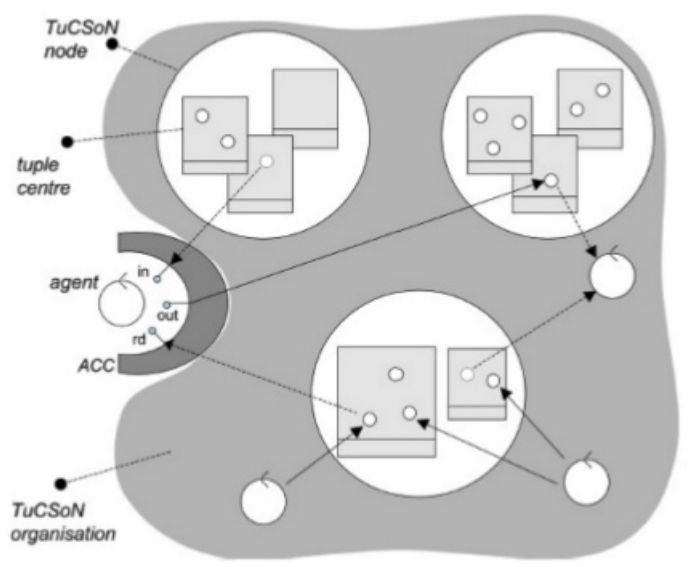


Figura 2.1: Architettura di un nodo TuCSoN in cui è presente anche l'ACC

### Transducer

I sistemi distribuiti sono situati in un ambiente e reattivi agli eventi di qualsiasi tipo e ciò comporta che i media di coordinamento sono tenuti a mediare qualsiasi attività verso l'ambiente consentendo un'interazione fruttuosa con esso. Gli attuali centri di tuple ReSpecT sono in grado di catturare gli eventi generati dall'ambiente, e di mediare la generale interazione processo-ambiente permettendo al sistema TuCSoN di esser anch'esso situato, renderlo in grado di catturare gli eventi ambientali ed esprimere le interazioni generali MAS-ambiente [12]. Il middleware Tucson ed il linguaggio ReSpecT possono quindi

catturare, reagire ed osservare gli eventi generati dall'ambiente in cui il sistema è situato, ma anche interagire esplicitamente ed influenzare l'ambiente ed in questo modo un evento di un centro di tuple può avere come origine o destinazione una qualsiasi risorsa esterna.
In TuCSoN è stato introdotto uno schema di identificazione adatto sia a livello di sintassi che di infrastruttura per rappresentare le risorse ambientali presenti nel sistema. Il linguaggio di coordinazione è stato esteso col fine di esprimere esplicitamente la manipolazione delle risorse ambientali e nel quale sono stati inseriti i seguenti nuovi predicati dei centri di tuple:

$$\langle EResId \rangle \; ? \; getEnv(\langle Key \rangle , \langle Value \rangle)$$

che consente ad un centro di tuple di ottenere le proprietà delle risorse ambientali specificate e

$$\langle EResId \rangle \; ? \; setEnv(\langle Key \rangle , \langle Value \rangle)$$

che consente ad un centro di tuple di configurare le proprietà delle risorse ambientali specificate.

A livello di infrastruttura, gli eventi dell'ambiente sono stati tradotti in eventi ReSpecT dei centri di tuple attraverso uno schema generale che potrebbe essere specializzato secondo la natura di una qualsiasi risorsa specifica ed il *Transducer* è il componente che ha il compito di portare gli eventi ambientali generati ad un centro ReSpecT di tuple (e viceversa) secondo il modello generale di eventi ReSpecT [9]. Ogni *Transducer* è specializzato in base alla porzione specifica dell'ambiente di cui è responsabile, o più in genere della risorsa specifica che deve poter manipolare, come un sensore di temperatura ecc.
Queste risorse all'interno dell'infrastruttura TuCSoN sono modellate attraverso un *Probe*. Un semplice *Probe* può essere un sensore, un attuatore o un qualsiasi altro dispositivo che si voglia inserire nel sistema. A ciascun *Probe* viene assegnato un *Transducer*, permettendoci di avere *Transducer* di tipo attuatore e di tipo sensore, che ha il compito di gestire gli eventi da/verso questa specifica risorsa e di tradurre i suoi cambiamenti di stato in eventi gestiti successivamente dai centri di tuple, ed eventi interni al sistema in

cambiamenti di stato da notificare ai *Probe*. I *Transducer* TuCSoN svolgono quindi un ruolo centrale nel sostenere la distribuzione ed il disaccoppiamento tra gli agenti e le risorse all'interno del MAS, mentre i centri tuple ed il linguaggio ReSpecT sono fondamentali per supportare sia la Situatedness sia la coordinazione.

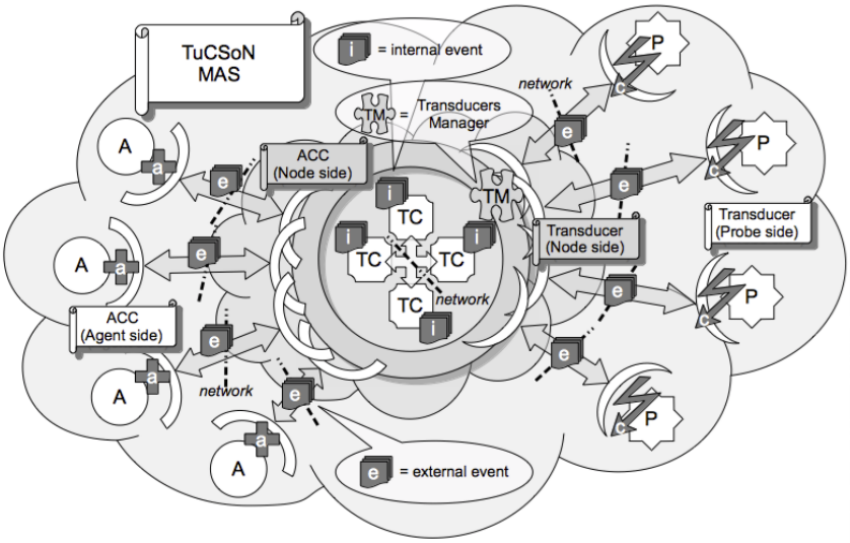


Figura 2.2: Architettura di un nodo TuCSoN in cui sono presenti Transducer e Probe

Ciascun *Transducer* può operare attraverso operazioni di coordinazione sia sincrone sia asincrone, mostrate rispettivamente in Fig. 2.3 e in Fig. 2.4 [13]. Dopo l'invio di un evento, il centro di tuple target dell'operazione richiesta reagisce innescando la reazione ReSpecT dell'annotazione 1.1.1 (2.1.1), che genera un evento situato (rispettivamente step 1.1.2 e 2.1.2) al fine di eseguire una operazione situata ($getenv(temp, T)/getenv(temp, T)$) sullo specifico *Probe* (sensore o attuatore). Il *Transducer* associato al centro di tuple e responsabile per il *Probe* di destinazione intercetta tale evento e si prende cura di eseguire effettivamente l'operazione su di esso (messaggio 1.1.2.1 / 2.1.2.1). La risposta del *Probe* (messaggio 1.1.2.2 / 2.1.2.2) genera una propagazione di eventi in risposta alla operazione di coordinazione originale invocata dall'agente (messaggio 1.1.2.3.2.1 / 2.1.2.3.2.1).

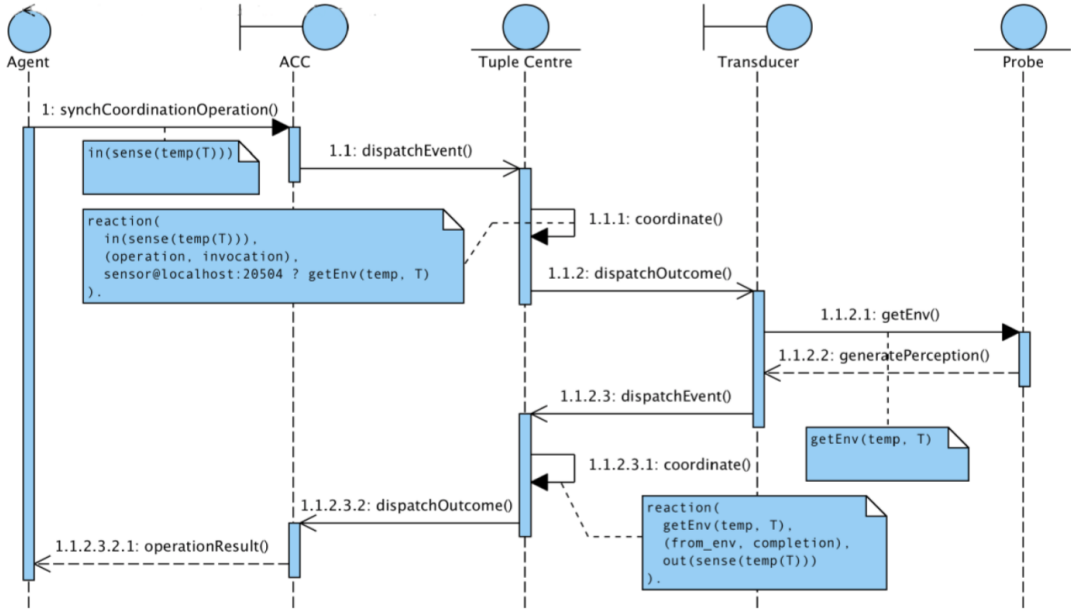


Figura 2.3: Interrogazione sincrona di un sensore.

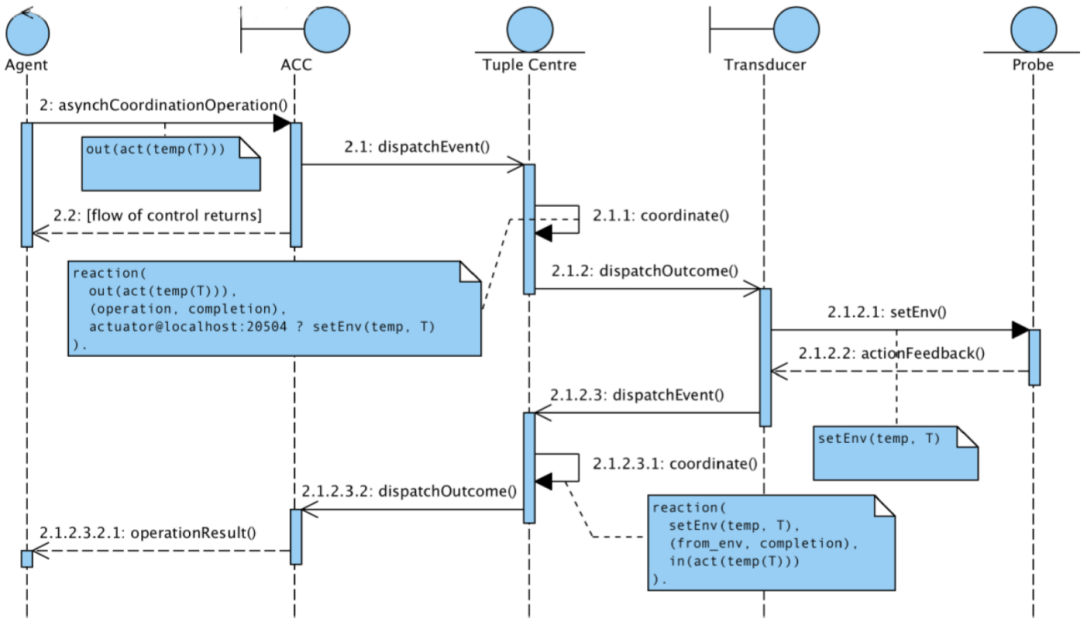


Figura 2.4: Operazione asincrona su un attuatore.

## 1.4 Centri di tuple spaziali

Nel corso degli anni TuCSoN ed il linguaggio di coordinazione ReSpecT sono stati estesi per poter affrontare tutte le problematiche di coordinazione legate ai concetti di tempo (non trattato in questa tesi), ambiente (affrontato la sezione precedente) e spazio, per i quali risulta necessaria un'opportuna gestione all'interno di un sistema situato e i cui componenti devono coordinarsi a fronte del passare del tempo e delle eventuali variazioni della posizione del sistema stesso. In particolare per quanto riguarda la coordinazione spaziale (ovvero legata allo spazio), essa richiede situatedness spaziale e che i mezzi di coordinazione nella loro ontologia contengano qualche nozione di spazio, ovvero che siano spatial aware [14].
I centri di tuple spaziali, a tal proposito, hanno esteso i centri di tuple permettendo di affrontare tali problematiche spaziali enunciate precedentemente. Nei centri di tuple è stata introdotta la nozione di *current place* per definirne la loro posizione, la quale può essere riferita alla posizione assoluta nello spazio del dispositivo nel quale il centro di tuple è in esecuzione, al domain name del nodo TuCSoN che lo ospita, oppure ad una locazione geografica. In questo modo è possibile rappresentare il movimento attraverso due tipi di eventi spaziali, il primo consiste nello spostamento da una posizione di partenza ed il secondo nella fermata ad una posizione di arrivo [14]. Ogni volta che si verifica un movimento sia nello spazio fisico sia in quello virtuale, viene generato un evento spaziale che, tramite l'estensione del linguaggio ReSpecT con le specifiche *reazioni spaziali*, innesca reazioni permettendo al centro di tuple di reagire a tale movimento. Oltre alla nozione di *current place* è presente quella di *località*, infatti quando viene invocata una primitiva di coordinazione senza la specifica del nodo, essa viene gestita implicitamente come riferita allo spazio di interazione locale che il nodo ospita.

Allo scopo di consentire la definizione di leggi di coordinazione che tengano conto della nozione di spazio, sono stati definiti nuovi eventi ammissibili generati in risposta agli eventi spaziali:

- `from(?S, ?P)` - corrisponde ad un evento spaziale generato quando il dispositivo che ospita il centro di tuple comincia a muoversi dalla posizione di partenza `P`, specificata in accordo con `S`;

- `to(?S, ?P)` - corrisponde ad un evento spaziale generato quando il dispositivo che ospita il centro di tuple raggiunge la posizione di arrivo `P`, specificata in accordo con `S` e termina il movimento.

Assieme a questi eventi sono definiti nella forma ⟨*EventView*⟩ dei *predicati di osservazione* utili per accedere alle proprietà spaziali e al recupero delle informazioni di un evento che si verifica all'interno di una reazione. La sintassi di tali predicati è:

- `current_place(@S, ?P)` - ha successo se `P` unifica con la posizione del nodo al quale in centro di tuple appartiene;
- `event_place(@S, ?P)` - ha successo se `P` unifica con la posizione del nodo nel quale è stato originato l'evento scatenante la computazione corrente;
- `start_place(@S, ?P)` - ha successo se `P` unifica con la posizione del nodo nel quale è stata originata la serie di eventi che hanno portato all'evento scatenante la computazione corrente.

dove la posizione del nodo può essere specificata sia come la sua posizione assoluta fisica (`S = ph`), il suo indirizzo IP (`S = ip`), il suo domain name (`S = dns`), la sua posizione geografica (`S = map`) tipicamente definita da servizi di mappatura come Google Maps, o la sua posizione organizzativa (`S = org`), cioè una posizione all'interno di una topologia organizzativa virtuale.

Sono definiti inoltre anche *predicati di guardia* che possono essere utilizzati per selezionare le reazioni da innescare a seconda delle proprietà degli eventi spaziali e la loro sintassi è la seguente:

- `at(@S, @P)` - ha successo quando il centro di tuple è presente nella posizione `P`, specificata in accordo con `S`;
- `near(@S, @P, @R)` - ha successo quando il centro di tuple è in esecuzione in una posizione contenuta nella regione con raggio `R` e centro `P`, specificata in accordo con `S`.

## 1.5 Geolocalizzazione

TuCSoN presenta un livello logico chiamato *Geolocation* atto a garantire l'interfacciamento tra i componenti precedentemente presentati e qualsiasi

piattaforma di geolocalizzazione e formato da tre principali entità generiche che fungono da ponte tra il servizio di geolocalizzazione ed il centro di tuple. Tali entità sono:

- `GeolocationService` - entità astratta che rappresenta il generico servizio di geolocalizzazione. Ad essa vengono associati un identificatore `GeoServiceId`, un centro di tuple a cui riferirsi per gli aggiornamenti di movimento e posizione e una piattaforma di esecuzione, inoltre comprende una lista di ascoltatori alla quale vengono aggiunti i listener `GeolocationServiceListener` che sono incaricati di ricevere le notifiche sulla posizione. Permette inoltre di generare eventi spaziali di tipo `from` e `to` nel caso in cui siano presenti reazioni spaziali nella specifica di comportamento del centro di tuple assegnatogli;
- `GeolocationServiceListener` - entità ascoltatore che ha il compito di rimanere in attesa dell'arrivo di notifiche sugli aggiornamenti di posizione e la generazione di eventi di movimento provenienti dal servizio di geolocalizzazione a cui è associata. Una volta ricevuti tali aggiornamenti essa ha il compito di aggiornare la posizione della RespecVM oppure di generare e notificare ad essa gli eventi spaziali eventualmente richiesti, tramite la specifica di comportamento, dal centro di tuple assegnatogli;
- `GeolocationServiceManager` - entità, definita mediante il patter Singleton, responsabile della creazione, registrazione e rimozione dei servizi di geolocalizzazione. Il compito della creazione del servizio di geolocalizzazione gli viene delegato dal nodo TuCSoN su cui se ne richiede la configurazione.

Poichè un agente potrebbe esser in esecuzione su un dispositivo in cui la RespectVM non è presente o in esecuzione TuCSoN mediante l'estensione dell'ACC, in particolare della classe `ACCProxyAgentSide`, permette di affiancare servizi di geolocalizzazione anche agli agenti. Tali aspetti però, dato l'obiettivo della tesi, non andranno approfonditi ma si rimanda alla tesi del collega Bombardi [15] per ulteriori dettagli.

#### Configurazione del servizio

La configurazione iniziale del livello di *Geolocation* è affidata all'agente *GeolocationConfigAgent*, il quale è avviato contestualmente all'avvio di un nodo TuCSoN e rimane in attesa di richieste di configurazione del servizio sul centro di tuple *geolocationConfigTC*. Tale centro di tuple è programmato, tramite le specifiche di comportamento contenute nel file `geolocation_spec.rsp`, in modo tale che tramite l'invocazione di due primitive specifiche di `out` su di esso è possibile creare e rimuovere il servizio. Le due operazioni che permettono la configurazione sono:

- `out(createGeolocationService(Sid, Sclass, Stcid))` - utilizzata per la creazione del servizio. Il servizio viene identificato da `Sid`, identificatore di tipo `GeoServiceId`, la cui classe di implementazione è la classe `Sclass` definita dall'utente e al quale viene associato il centro di tuple `Stcid`;
- `out(destroyGeolocationService(Sid))` - utilizzata per la rimozione del servizio specificato dall'identificatore `Sid`.

# 2 TuCSoN in Home Manager

All'interno di Home Manager troviamo tre principali elementi, le risorse, gli agenti e i centri di tuple, tutti strettamente correlati con l'infrastruttura di coordinazione TuCSoN.

## 2.1 Risorse

Nel prototipo attuale le risorse sono modellate come entità denominate "device" e rappresentate da tuple all'interno dei centri di tuple. Le entità device sono implementate dalla classe Java `Device`, all'interno del package `it.unibo.homemanager.detection`, che fornisce tutti i metodi necessari per gestire e salvare le proprietà del dispositivo, il suo stato verrà anche salvato all'interno di un centro di tuple dedicato sottoforma di tupla.

In origine il prototipo di Home Manager prevedeva solamente la simulazione di alcune possibili risorse presenti all'interno dell'abitazione, anch'essa simulata, ma recenti sviluppi hanno permesso di interfacciare anche dispositivi embedded fisici con semplici sensori o di utilizzare quest'ultimi per riprodurre dispositivi casalinghi più complessi. Parallelamente a questa funzione è stato anche sviluppato un meccanismo di rilevazione automatico di nuovi dispositivi esterni/risorse aggiunti a run-time, che tramite una negozazione del nome dei dispositivi attraverso uno scambio di tuple, permette di inserire una nuova risorsa senza andare ad agire sul codice o sui file di configurazione delle stanze e della abitazione. Al momento dell'inserimento di una risorsa fisica viene comunque creata un entità virtuale all'interno del sistema che la rappresenta.

Detto ciò possiamo notare come nel prototipo non sia presente il concetto di *Transducer* e che le risorse e le relative astrazioni sono realizzate e gestite a livello applicativo e non a quello di infrastruttura, che risulterebbe più corretto e funzionale a livello ingegneristico e fornirebbe un migliore disaccoppiamento tra risorse, infrastruttura e sistema applicativo. Nella sua tesi [16], il collega Pometto aveva fornito uno "studio di fattibilità" dell'utilizzo dei *Transducer* all'interno di una versione meno recente di Home Manager, mirato a capire come fosse tecnicamente fattibile realizzare l'utilizzo e nel quale era riuscito a fornire un semplice caso di studio.
Obiettivo della mia tesi sarà quello di estendere questo precedente studio al sistema di rilevazione automatico di nuovi dispositivi/risorse.

## 2.2 Agenti

In Home Manager gli agenti presenti nel sistema possono essere classificati secondo la seguente convenzione(informale): vi sono agenti "di tracciamento" predisposti al tracciamento dei movimenti degli utenti, agenti "pianificatori" che si occupano di realizzare un piano di azione, scelto in base ai dati ottenuti dagli agenti "di tracciamento", agenti "esecutori" che realizzano concretamente il piano proposto dagli agenti "pianificatori", ed infine gli agenti di gestione delle nuove risorse. Sono inoltre presenti agenti generici che hanno il compito di coordinare gli altri agenti. Questa società di agenti, attraverso

l'interazione e la coordinazione delle singole parti, ha il fine di gestire il comportamento complessivo dell'intera abitazione.

Di seguito sono elencati alcuni dei principali agenti presenti nel sistema.

Agenti "di tracciamento"
(`package it.unibo.homemanager.home_agents.check_people`)

- `DetectorAgent` - mediante l'interrogazione dei sensori presenti in ciascuna stanza esso controlla i movimenti delle persone fra le varie stanze;
- `ListManager` - esso è incaricato di mantenere aggiornata una lista contenente le informazioni sulle presenze in ciascuna stanza.

Agenti "pianificatori"
(`package it.unibo.homemanager.home_agents.elab_plan`)

- `ActControllerAgent` - determina la presenza di eventuali attività in sospeso e che devono essere eseguite;
- `CmdControllerAgent` - è incaricato di gestire i comandi degli utenti dell'abitazione inviati tramite il terminale;
- `ConflictsManagerAgent` - è incaricato di valutare le informazioni ricevute dagli altri agenti e di generare un nuovo piano di azione in caso di conflitti tra le preferenze degli utenti e i relativi piani di azione;
- `PrefControllerAgent` - ha il ruolo di raccogliere e gestire le preferenze di ciascun utente in una stanza in cui si deve eseguire il piano scelto.

Agenti "esecutori"
(`package it.unibo.homemanager.home_agents.elab_plan`)

- `BlindAgent` - controlla le tapparelle automatizzate;
- `BrightnessAgent` - incaricato di amministrare i sensori di luminosità e regolare le luci e le tapparelle automatizzate della stanza. In caso di luce esterna ne ottimizza anche l'utilizzo per avere un minor consumo di energia elettrica e conseguentemente un maggior risparmio energetico;
- `DeviceAgent` - gestisce un dispositivo o un elettrodomestico all'interno della stanza, interfacciandosi al sistema mediante il rispettivo ACC specificato durante la fase di inizializzazione;
- `LampAgent` - ha il ruolo di regolare lo stato delle lampade all'interno dell'abitazione;
- `PlanDistributorAgent` - è colui che durante l'esecuzione di un piano distribuisce i comandi agli agenti e alle entità coinvolte;
- `WindowAgent` - gestisce gli infissi automatizzati.

Agente di gestione nuove risorse

(`package it.unibo.homemanager.detection`)

- `DeviceManagerAgent` - il suo ruolo consiste nel controllare costantemente se nel centro di tuple destinato ai nuovi dispositivi aggiunti a run-time è presente una richiesta di nome da parte di un nuovo dispositivo appena collegato. In caso di nuova richiesta esso è incaricato di assegnare un nome, associarlo al tipo di dispositivo e di notificarlo a quest'ultimo. Il dispositivo poi viene aggiunto anche al centro di tuple della specifica stanza che tiene traccia dei dispositivi presenti nell'abitazione e tutti gli agenti coinvolti vengono notificati.

## 2.3 Centri di tuple

In Home Manager vi sono numerosi centri di tuple nei quali sono racchiuse tutte le informazione del sistema abitativo. Essi sono distinguibili in:

- *Stanze* - ogni stanza ha un suo centro di tuple(sono compresi anche i corridoi, eventuali zone ingresso e garage);
- *Casa* - oltre le stanze è presente un centro di tuple per rappresentare la casa;
- *DB* - centro di tuple in cui sono memorizzate le informazioni sugli utenti e le loro preferenze, gli URL dei web services utilizzati dal servizio meteo, le credenziali di Twitter. Contiene inoltre anche tutte le tuple che rappresentano i sensori e i dispositivi presenti nella casa;
- *Device manager* - corrisponde al centro di tuple che raccoglie tutte le richieste di naming da parte dei nuovi dispositivi collegati a run-time;
- *RBAC* - contiene tutte le informazioni per il funzionamento del sistema Role-Based Access Control, ovvero tutte le credenziali di accesso alle risorse e le azioni eseguibili su tutti i dispositivi per ciascun utente presente nel sistema;
- *Risorse* - ciascuna risorsa o dispositivo hanno un proprio centro di tuple nel quale salvare i dati;
- *Meteo* - i dati meteo raccolti dal web service utilizzato vengono inseriti in questo centro di tuple;
- *Twitter* - raccoglie le informazioni social degli utenti;

- *UsageManager* - contiene le informazioni e le politiche di gestione delle risorse focalizzate al risparmio energetico.

# Capitolo 3

# Transducer in Home Manager

Nella sezione 2.2.1 abbiamo evidenziato che nell'attuale prototipo di Home Manager non è ancora presente il concetto di *Transducer*. Attualmente infatti è prevista per dispositivi simulati e reali solamente una coordinazione effettuata tramite scambio di tuple e al sistema non è permesso catturare gli eventi generati dall'ambiente o mediarne l'interazione. L'integrazione del *Transducer* permette a dispositivi e risorse simulate ma anche reali di interagire con il sistema in modo del tutto trasparente e di spostare la gestione delle risorse e dei dispositivi non più a livello applicativo ma a livello di infrastruttura, consentendo inoltre di esprimere le interazioni sistema-ambiente e i rispettivi eventi. In termini di livelli dell'architettura Butlers, l'introduzione dei *Transducer* permette di avere uno sviluppo ancora più dettagliato di quanto già presente dei livelli di *Information* e *Control*.
In questo capitolo in particolare prenderemo in esame l'elettrodomestico *"Frigorifero"*, di come esso è gestito all'interno del sistema e come sono realizzate le interazioni e lo scambio di informazioni che lo interessano.

## 1 Analisi

Nell'attuale prototipo di Home Manager, come accennato nelle sezione 2 del secondo capitolo, esiste un sistema di autodetect dinamico dei dispositivi che rende il sistema sensibile all'introduzione di nuove risorse collegate a runtime, le quali attraverso l'inserimento di una tupla specifica all'interno del centro di tuple *device_manager_tc* richiedono il nome identificativo che li con-

traddistinguerà dagli altri dispositivi all'interno della casa. Una volta inserita la tupla essa sarà l'agente DeviceManagerAgent a gestire la prima interazione con la risorsa, esso infatti tramite un'operazione di coordinazione `in` verifica che sia presente tale tupla all'interno del centro di tuple e in caso di match si incarica di creare il nome univoco per la risorsa e di inviargli una risposta contenente quest'ultimo. Successivamente alla risposta esso salva inoltre la presenza del nuovo dispositivo sia in versione fisica sia in quella simulata, quest'ultima è stata decisa come necessaria per permettere all'utente di continuare a lavorare anche in caso di guasto di quello reale.
La creazione degli agenti che monitorano i dispositivi è delegata ad un componente chiamato `AgentManager` per soddisfare il principio delle singole resposabilità, il quale si avvale del metodo `createAgent` della classe astratta `AgentFactory` per istanziare tutti gli agenti interessati ed inizializzarne correttamente tutti i parametri e le interfacce. Ciascun agente viene poi inserito in una lista allo scopo di tener traccia di tutte le entità presenti nel sistema e che sarà utilizzata per lanciare poi l'esecuzione attraverso il metodo (della classe `AgentManager`) `goAgents()` o `goAgent(String name)` rispettivamente di tutti gli agenti o di un singolo agente specificato.
Da questo momento inizia la vera e propria interazione con scambio di dati attraverso tuple tra il sistema Home Manager e il dispositivo appena collegato, andando a sfruttare i centri di tuple creati ad hoc per la singola risorsa.
In particolare, per il dispositivo *"Frigorifero"* le entità coinvolte sono `FridgeAgent`, l'agente che ha il compito di simulare l'ettrodomestico, e il centro di tuple *fridge_tc*, il quale ha il compito di raccogliere le informazioni sui cibi e bevande presenti al suo interno. Tali informazioni sono rappresentate sotto forma di tuple del tipo

```
content(fridge_#, item, quantity)
```

dove # indica il numero del dispositivo, *item* e *quantity* rispettivamente l'oggetto contenuto all'interno del frigorifero e la sua quantità. In questa tesi si è scelto di non usare un dispositivo fisico tenendo in considerazione il solo `FridgeAgent` il quale possiede il compito di tenere aggiornata la lista degli oggetti presenti richiedendo il contenuto del centro di tuple *fridge_tc* e andando a verificare che non vi sia una richiesta di prodotti da parte dell'agente `MixerAgent` all'interno del centro di tuple *mixer_tc*, visualizzare la

lista nell'interfaccia grafica presente su Home Manager e simulare l'inserimento e l'eliminazione di prodotti all'interno dell'elettrodomestico simulato nel momento in cui un utente autenticato utilizzi gli strumenti di inserimento o cancellazione forniti dall'interfaccia stessa. Possiamo quindi affermare che nel suo comportamento è racchiusa tutta l'interazione che avviene tra sistema ed i vari centri di tuple coinvolti nelle operazioni di coordinazione da lui invocate, la quale è rappresentata dalla Fig. 3.1 (per semplificare le prossime due immagine non sono state inserite le eventuali reazioni ReSpecT).

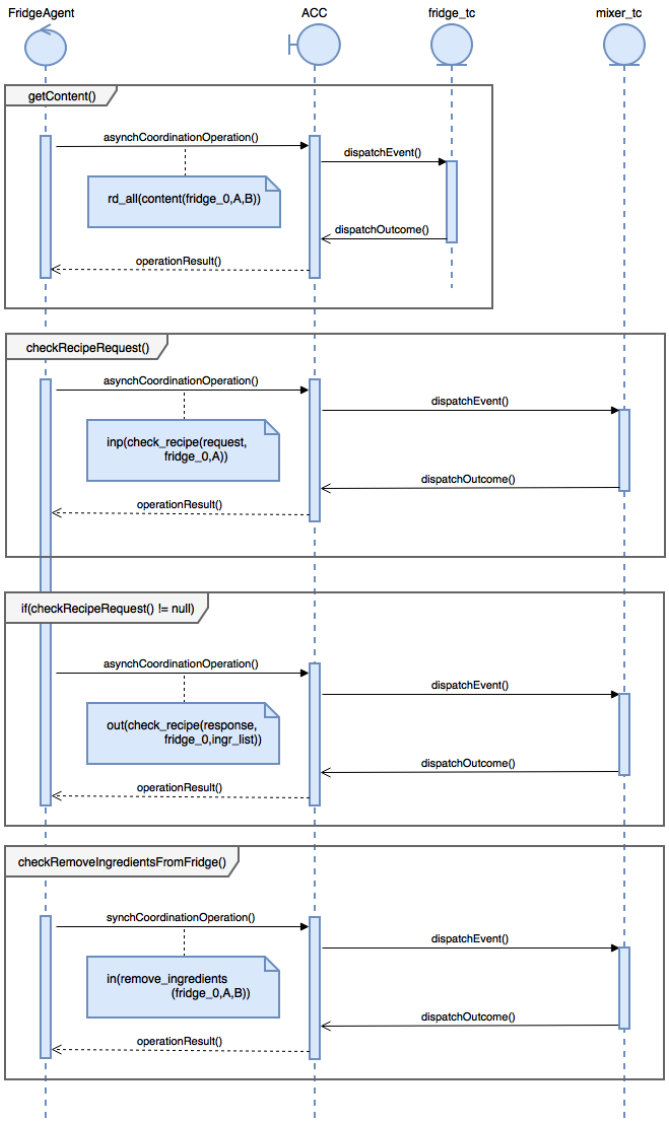


Figura 3.1: Iterazione tra `FridgeAgent` e centri di tuple.

Come detto precedentemente però un utente loggato correttamente attraverso l'interfaccia utente resa disponibile da Home Manager nella sezione Butles, può far eseguire all'agente `FridgeAgent` azioni di inserimento (`addIngredient`) o aggiornamento/eliminazione (`updateContent`), entrambi simulate, di oggetti all'interno del frigorifero. La figura seguente ne rappresenta lo schema.

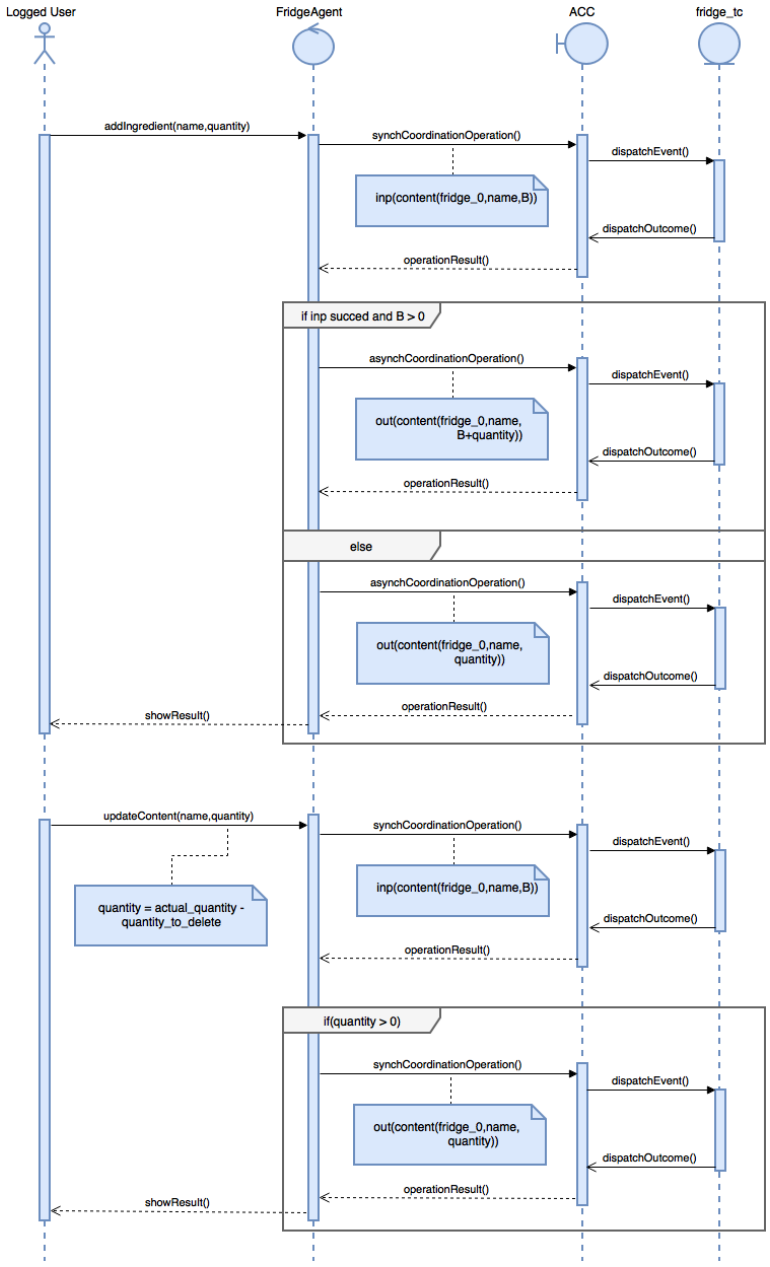


Figura 3.2: Iterazione tra utente, `FridgeAgent` e centri di tuple.

L'introduzione del *Tranducer* per il dispositivo "Frigorifero" però andrà ad interessare solamente le interazioni che coinvolgono il centro di tuple *fridge_tc*.

# 2 Progetto

Come già anticipato nella sezione 2.1.3 un *Transducer* comprende tre principali componenti: il *Transducer* vero e proprio, le *reaction* da inserire nel centro di tuple, il centro *fridge_tc* nel nostro caso, per ottenere il comportamento desiderato a fronte di eventi ambientali e di invocazioni specifiche di operazioni di coordinazione ed infine il *Probe*, un componente che rappresenta il modello della risorsa all'interno del sistema TuCSoN e che consiste in una sorta di ponte/driver con il dispositivo che vogliamo andare a gestire.
Data la complessità del prototipo Home Manager e data la moltitudine di entità che sono strettamente correlate tra loro al fine di ottenere i meccanismi di automazione previsti dall'architettura Butler, inizialmente l'agente `FridgeAgent` sarà sostituito da un semplice agente, denominato `FridgeSituatedAgent`, realizzato ad hoc attraverso il quale si andrà a configurare opportunamente il *Transducer* ed il *Probe*, a caricare le reazioni ReSpecT e a verificare la correttezza delle funzionalità previste per *Transducer*, *Probe* e *reaction*. Tale scelta ci permette inoltre di verificare quanto l'introduzione del *Transducer* all'interno di Home Manager ci permetta di realizzare una forma di modularità per quanto riguarda i dispositivi collegati ad esso. Una volta implementato il *Transducer* , qualora si decidesse di utilizzare un altro dispositivo le uniche modifiche da effettuare sul codice saranno quelle sul *Probe* e sulle specifiche *reaction* .

## 2.1 Transducer

Per quanto riguarda il *Transducer* si è deciso che, dato il comportamento complessivo dell'attuale prototipo, esso dovrà possedere un comportamento sia da attuatore che da sensore. Nonostante si sia deciso di tralasciare in un primo momento il `FridgeAgent` e di utilizzare un semplice agente, il *Transducer* deve poter permettere al sistema Home Manager di interrogare il dispositivo "Frigorifero" per apprenderne il contenuto e di poter modificar-

ne lo stato a fronte di un comando di aggiunta/modifica/rimozione inviato tramite l'interfaccia grafica da parte di un utente. Detto ciò si può notare come il comportamento attuale del sistema non verrà per nulla modificato, ma esso verrà esteso in quanto le operazioni che l'agente `FridgeSituatedAgent` (o anche `FridgeAgent`) invocherà sul centro di tuple *fridge_tc* dovranno essere propagate al *Probe* e gli eventi generati da quest'ultimo potranno essere catturati dal *Transducer* e poi gestiti secondo le *reaction* programmate sul centro di tuple. Nella Fig. 3.3 è rappresentata l'interazione tra le entità coinvolte nello scenario in cui il sistema Home Manager, attraverso l'agente `FridgeSituatedAgent`, modifica lo stato del *Probe*, invocando una primitiva di `out` sul centro di tuple *fridge_tc*, mentre nella Fig. 3.4 è raffigurata l'interazione nel momento in cui è il sistema tramite l'agente a richiedere lo stato del dispositivo.

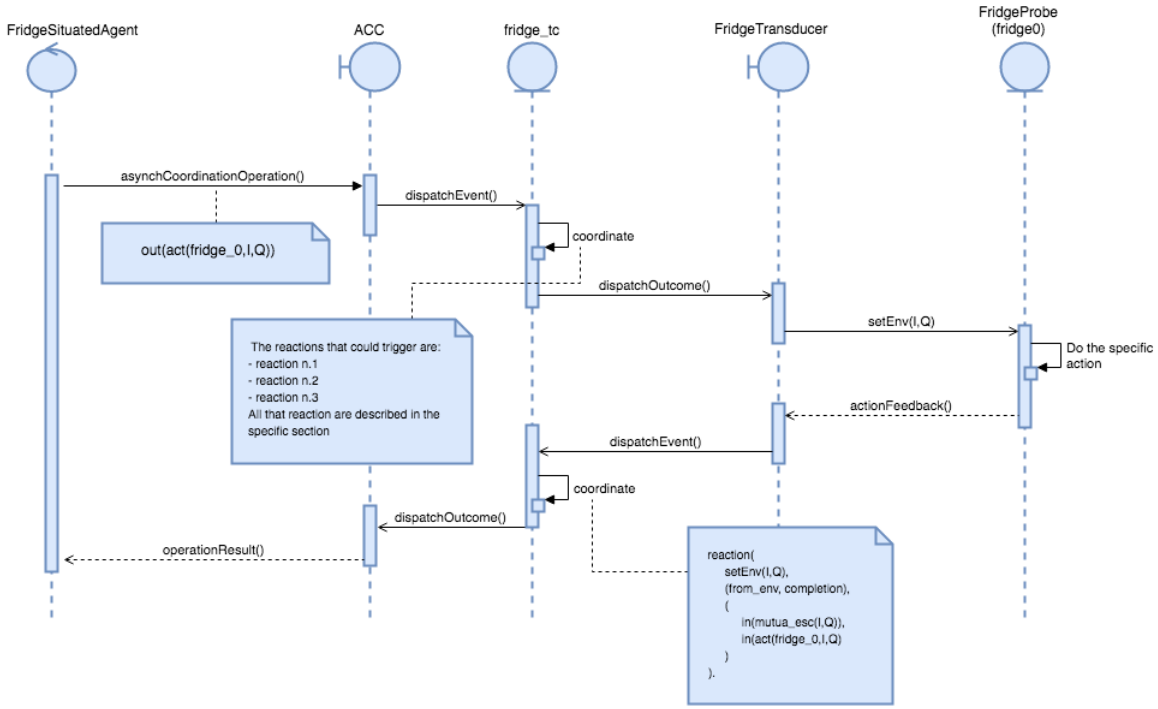


Figura 3.3: Operazione di aggiunta/modifica/cancellazione di un ingrediente nel *Probe*.

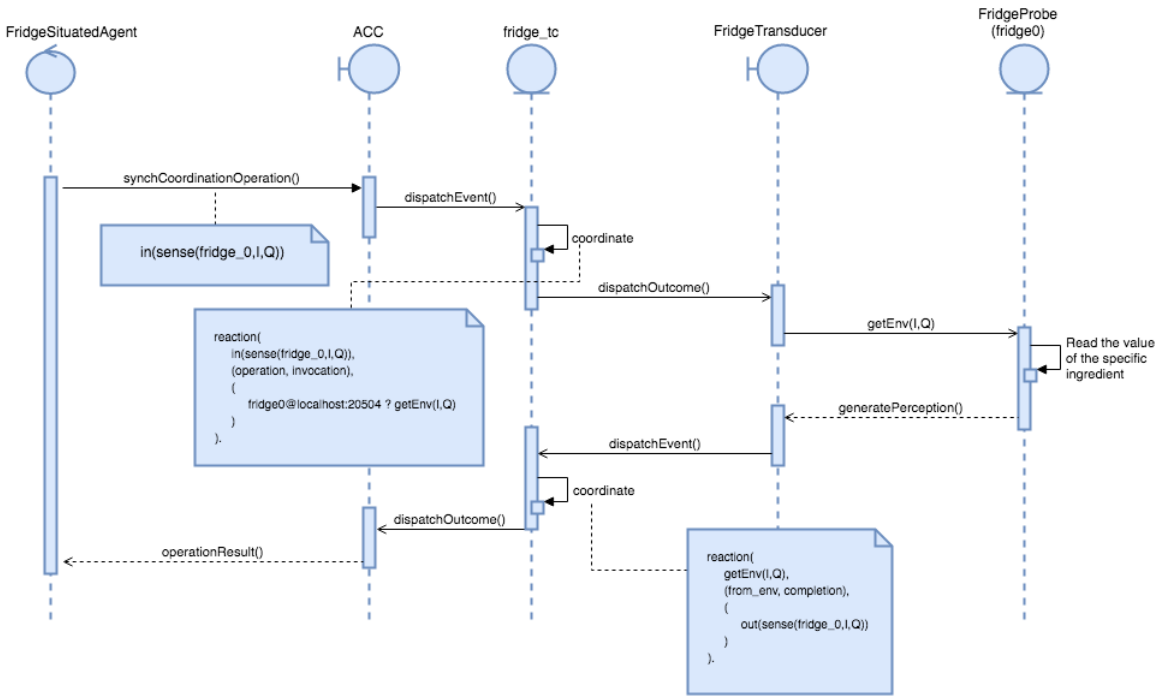


Figura 3.4: Interrogazione sullo stato di un ingrediente nel *Probe*.

## 2.2 Probe

Il probe utilizzato inizialmente in questo prototipo non sarà collegato ad un dispositivo fisico, ma rappresenterà un frigorifero virtuale e i relativi dati saranno memorizzati in un apposito centro di tuple denominato *probeFridgeTc*. Data la possibilità di avere un *Transducer* attuatore e sensore e l'esistenza dei due soli predicati *setEnv* e *getEnv* si utilizzerà una convenzione per poter realizzare le operazioni di aggiunta, modifica, cancellazione e interrogazione di stato di un ingrediente, in particolare essa verrà utilizzata dal nuovo agente `FridgeSituatedAgent` nelle operazioni di coordinazione eseguite per verificare il corretto funzionamento del prototipo e nella parte da attuatore del *Probe*. Quest'ultima verrà implementata dal metodo `writeValue(String key, int state)`, richiamata dalla `setEnv` del *Transducer* e nel quale avremo un'operazione di aggiunta o incremento di un ingrediente per valori positivi del parametro `state`, per valori negativi un'operazione di modifica con riduzione della quantità e per il valore nullo "0" la cancellazione dell'ingrediente specificato dal parametro `key`. La funzionalità di sensore sarà implementata normalmente sviluppando il metodo `readValue(String`

`key)` richiamato dalla getEnv del *Transducer*. Tutte le aggiunte, modifiche e cancellazioni dei vari ingredienti saranno memorizzati di volta in volta nel centro di tuple *probeFridgeTc* mediante semplici operazioni di coordinazione, `in` e `out`, mentre la lettura dello stato di un determinato ingrediente avverrà per mezzo di una `rd`.
Come specificato in fase di progetto, l'utilizzo dei *Transducer* permette di avere una forma di modularità all'interno del sistema per quanto riguarda i dispositivi esterni e quelli simulati. Ciò sarà provato attraverso l'implementazione di un secondo *Probe* dotato di una semplicissima interfaccia grafica che però sarà testato assieme all'attuale `FridgeAgent`. Quest'ultimo sarà leggermente esteso per realizzare la configurazione del nuovo *Probe*, del relativo *Transducer* (che rimarrà lo stesso realizzato per il primo *Probe*) e delle reazione ReSpecT necessarie, ma non subirà nessun'altra modifica e il suo comportamento, ovvero le operazioni di coordinazione invocate e le operazioni di gestione dei dati, rimarrà intatto.

## 2.3 Reaction

Qui si riportano già le *reaction* utilizzate per questo prototipo così da facilitare il lettore nella comprensione del funzionamento del prototipo e delle Fig. 3.3 e 3.4. Tali reazioni fanno riferimento al file `fridgeSpec.rsp`. Inoltre, come menzionato precedentemente, il simbolo # verrà sostituito dal numero assegnato al dispositivo nel momento in cui l'agente `FridgeSituatedAgent` configurerà *Transducer* e *Probe*.

```
1  % Reaction n.1
2  reaction(
3     out(act(fridge_#, I, Q)),
4     (operation, completion),
5     (
6        no(mutex(A,B)),
7        out(mutex(I,Q)),
8        no(content(fridge_#, I, A)),
9        out(content(fridge_#, I, Q)),
10       fridge#@localhost:20504 ? setEnv(I, Q)
11    )
12 ).
13
14 %Reaction n.2
15 reaction(
16    out(act(fridge_#, I, Q)),
```

```
17    (operation, completion),
18    (
19       no(mutex(A,B)),
20       out(mutex(I,Q)),
21       in(content(fridge_#, I, A)),
22       Y is A+Q,
23       Y > 0,
24       out(content(fridge_#, I, Y)),
25       fridge#@localhost:20504 ? setEnv(I, Q)
26    )
27 ).
28
29 %Reaction n.3
30 reaction(
31    out(act(fridge_#, I, Q)),
32    (operation, completion),
33    (
34       no(mutex(A,B)),
35       out(mutex(I,Q)),
36       in(content(fridge_#, I, A)),
37       Y is A+Q,
38       Y == 0,
39       fridge#@localhost:20504 ? setEnv(I, Q)
40    )
41 ).
42
43 reaction(
44    setEnv(I, Q),
45    (from_env, completion),
46    (
47       in(mutex(I,Q)),
48       in(act(fridge_#, I, Q))
49    )
50 ).
51
52 reaction(
53    in(sense(fridge_#, I, Q)),
54    (operation, invocation),
55    (
56       fridge#@localhost:20504 ? getEnv(I,Q)
57    )
58 ).
59
60 reaction(
61    getEnv(I,Q),
62    (from_env, completion),
63    (
64       out(sense(fridge_#, I, Q))
65    )
66 ).
```

Come si può vedere dal codice presentato nelle prime tre *reaction* è presente una tupla denominata `mutex(I,Q)`, essa viene utilizzata per realizzare, come il nome suggerisce, una forma di mutua esclusione nell'esecuzione delle reaction per evitare di avere comportamenti inattesi ed indesiderati dovuti alla eleborazione non deterministica delle *reaction* che triggerano a fronte dell'evento `out(act(fridge_#, I, Q)` e all'esecuzione con successo delle operazioni contenute nel corpo della *reaction* che potrebbero influenzare l'esecuzione delle *reaction* processate sequenzialmente a quella appena terminata.

# 3 Implementazione

Durante la fase di progettazione si è deciso di realizzare i tre principali componenti che compongono un *Transducer* e un semplice agente `FridgeSituatedAgent` con il quale interagirerà. Andranno quindi implementati:

- Un *Transducer*, attraverso la classe Java `FridgeTransducer` che estende la classe astratta `AbstractTransducer` fornita dalle API TuCSoN;
- un *Probe*, mediante la classe Java `FridgeProbe` che implementa l'interfaccia `ISimpleProbe`;
- le relative reazioni ReSpecT già presentate nella sezione precedente;
- l'agente `FridgeSituatedAgent` mediante la classe Java ononima che estende la classe astratta `AbstractTucsonAgent`.

La realizzazione di questi componenti come già espresso nel corso della tesi permetterà in termini di ingegneria del software di realizzare una forma di modularità del sistema e nel contesto dell'architettura Butlers di realizzare e adottare quasi completamente i due livelli *Control* e *Information* all'interno dell'infrastruttura di coordinazione e non più a livello applicativo, andandosi così a posizionare al giusto livello di astrazione.

## 3.1 FridgeTransducer

L'implementazione di un *Transducer* risulta piuttosto standard tra i vari scenari applicativi pensabili. In caso di *Transducer* sensore si dovrà implementare solamente il metodo `getEnv`, nel caso invece di *Transducer* attuatore

la funzione `setEnv` ed infine entrambi se si vuole permettere al *Transducer* di essere sia sensore, sia attuatore.
Come deciso in fase di progettazione il componente `FridgeTransducer` è stato realizzato secondo quest'ultima opzione ed entrambi i metodi sono stati implementati per eseguire le funzionalità richieste. Il metodo `getEnv` permetterà di conoscere lo stato dell'ingrediente richiesto e passato come parametro `key`.
L'implementazione del metodo `getEnv` è presentata qui sotto.

```
@Override
public boolean getEnv(final String key) {
 this.speak("[" + this.id + "]: Reading...");
      boolean success = true;
      final Object[] keySet = this.probes.keySet().toArray();
      /*
       * for each probe this transducer models, stimulate it to sense its
       * environment
       */
      for (final Object element : keySet) {
         if (!((ISimpleProbe) this.probes.get(element)).readValue(key)) {
            this.speakErr("[" + this.id + "]: Read failure!");
            success = false;
            break;
         }
      }
      return success;
 }
```

Il metodo `setEnv` invece permette di innescare le funzioni da attuatore del *Probe* consentendo l'aggiunta, la cancellazione di un ingrediente e la modifica della sua quantità nel centro di tuple *fridge_tc* e sarà il valore del parametro `state`, che seguirà la convenzione presentata in fase di progettazione, a stabilire quale azione di quelle precedentemente citate verrà realmente eseguita.

Di seguito ne è presentata l'implementazione:

```
@Override
 public boolean setEnv(String key, int value) {
      this.speak("[" + this.id + "]: Writing...");
      boolean success = true;
      final Object[] keySet = this.probes.keySet().toArray();
      /*
       * for each probe this transducer models, stimulate it to act on its
       * environment
```

```
9         */
10      for (final Object element : keySet) {
11         if(!((ISimpleProbe)this.probes.get(element)).writeValue(key,value)){
12            this.speakErr("[" + this.id + "]: Write failure!");
13            success = false;
14            break;
15         }
16      }
17      return success;
18   }
```

## 3.2 FridgeProbe

Questa classe rappresenta il dispositivo virtuale i cui dati (gli ingredienti) verranno memorizzati sotto forma di tuple nel centro di tuple `probeFridgeTc` e che si limiterà a compiere le operazioni richiestegli dall'agente `FridgeSituatedAgent`. Per evitare ulteriori operazioni di coordinazione sul centro di tuple `probeFridgeTc`, si è deciso di memorizzare i dati degli ingredienti anche in una lista di oggetti `Ingredients` che opportunamente verrà aggiornata ad ogni operazione per mantenere la consistenza dei dati e tramite la quale andremo a recuperare le informazioni sugli ingredienti, per poi confrontarle con quelle ricevute e decidere quali azioni intraprendere realmente. Tali operazioni si limiteranno all'aggiunta di un ingrediente, alla modifica della quantità di un ingrediente già presente all'interno del frigorifero, alla sua completa eliminazione o alla semplice interrogazione di stato ed esse sono implementate rispettivamente dai metodi `writeValue(String key, int state)` e `readValue(String key)` di seguito riportati:

```
1  @Override
2  public boolean writeValue(String key, int state) {
3
4   [...]
5   boolean new_ingredient = true;
6   int index = -1;
7   for(Ingredient ingredient : ingredients){
8      String correct_key = ingredient.getName();
9       if (key.equals(correct_key)){
10       index = ingredients.indexOf(ingredient);
11       new_ingredient = false;
12       break;
13      }
14  }
```

```
15 if(!new_ingredient){
16  // I've already had that ingredient in the fridge
17  // I have to check the state value
18     int ing_quantity = ingredients.get(index).getQuantity();
19     int new_quantity = ing_quantity + state;
20     if(new_quantity > 0){
21      // The value has to be updated
22      try {
23        final LogicTuple template = LogicTuple.parse("content(" + key + " , Q)");
24        final ITucsonOperation op = this.acc.in(this.probeFridgeTc, template,null);
25        if (op.isResultSuccess()) {
26           ingredients.get(index).setQuantity(new_quantity);
27           final LogicTuple tempTuple =
28                 LogicTuple.parse("content("+key+","+new_quantity+")");
29           this.acc.out(this.probeFridgeTc, tempTuple, null);
30           System.err.println("["+this.pid+"]Now we have" +new_quantity+" of "+key);
31           this.transducer.notifyEnvEvent(key, state, AbstractTransducer.SET_MODE);
32           return true;
33        }
34      } catch (TucsonOperationNotPossibleException | UnreachableNodeException |
35            OperationTimeOutException | InvalidLogicTupleException e) {
36       e.printStackTrace();
37      }
38     }else if(new_quantity == 0){
39      // The ingredient has to be deleted
40         try {
41           Ingredient ing_deleted = ingredients.remove(index);
42        final LogicTuple template = LogicTuple.parse("content(" + key + " , Q)");
43        final ITucsonOperation op = this.acc.in(this.probeFridgeTc, template,null);
44        if (op.isResultSuccess()) {
45           System.err.println("["+this.pid+"]Ingredient "+key+" deleted");
46            this.transducer.notifyEnvEvent(key, state, AbstractTransducer.SET_MODE);
47            return true;
48        }
49      } catch (TucsonOperationNotPossibleException | UnreachableNodeException |
50             OperationTimeOutException | InvalidLogicTupleException e) {
51        e.printStackTrace();
52      }
53     }
54 }else{
55  // A new ingredient
56  try{
57   ingredients.add(new Ingredient(key, state));
58   LogicTuple tempTuple;
59   tempTuple = LogicTuple.parse("content(" + key + ", "+ state + ")");
60   final ITucsonOperation op = this.acc.out(this.probeFridgeTc, tempTuple, null);
61   if (op.isResultSuccess()) {
62       System.err.println("["+this.pid+"]New ingredients: "+state+" of "+key);
63       this.transducer.notifyEnvEvent(key, state, AbstractTransducer.SET_MODE);
64       return true;
65    }
66      } catch (TucsonOperationNotPossibleException | UnreachableNodeException |
```

```
67      OperationTimeOutException | InvalidLogicTupleException e) {
68      e.printStackTrace();
69        }
70   }
71   System.err.println("["+this.pid+"]Unknown property " + key);
72   return false;
73  }
```

```
1  @Override
2  public boolean readValue(String key) {
3
4   [...]
5   boolean new_ingredient = true;
6   for(Ingredient ingredient : ingredients){
7     String correct_key = ingredient.getName();
8     if (key.equals(correct_key)){
9       // Tuple template to match
10      try{
11       final LogicTuple template = LogicTuple.parse("content("+ key +", Q)");
12       final ITucsonOperation op = this.acc.rd(this.probeFridgeTc, template, null);
13       if (op.isResultSuccess()) {
14         final String quantity = op.getLogicTupleResult().getArg(1).toString();
15         System.err.println("["+this.pid+"]My content is: "+"num."+quantity+" of "+key);
16         // Notify the Transducer
17         this.transducer.notifyEnvEvent(key,Integer.parseInt(quantity),
18                     AbstractTransducer.GET_MODE);
19         return true;
20       }
21     } catch (TucsonOperationNotPossibleException | UnreachableNodeException |
22          OperationTimeOutException | InvalidLogicTupleException e) {
23         e.printStackTrace();
24     }
25    }
26   }
27   System.err.println("["+this.pid+"]Unknown property "+key);
28   return false;
29  }
```

## 3.3 FridgeSituatedAgent

L'implementazione di questo agente si è focalizzata sulla realizzazione delle funzionalità di base che un agente TuCSoN deve avere per poter configurare ed interagire con un dispositivo mediante l'utilizzo di un *Transducer* così da poter verificare nel modo più semplice possibile tutte le funzionalità implementate nelle classi precedenti sapendo inoltre che, una volta appurata la correttezza di tutte le funzioni implementate, le modifiche da apportare in

caso di inserimento del *Transducer* nel sistema complessivo di Home Manager sarebbero state limitate al solo *Probe* e a come verrebbero caricate le configurazioni.
I compiti svolti dall'agente `FridgeSituatedAgent` saranno quindi i seguenti:

- Generare la tupla di configurazione del *Transducer* e del relativo *Probe* ed inserirla all'interno del centro di tuple di configurazione del sistema;
- Recuperare le specifiche reazioni ReSpecT da file e configurarle sul centro di tuple interessato. Prima di fare ciò esso deve preventivamente controllare che all'interno di tale centro di tuple non siano già presenti altre *reaction*, in caso positivo anch'esse devono essere salvate, aggiunte a quelle specifiche per il *Transducer* e poi configurate nuovamente nel centro di tuple;
- Verificare le funzionalità sviluppate attraverso l'invocazione di semplici operazioni di coordinazione.

### Generazione della tupla di confgurazione

Di seguito è riportato il codice in cui è implementata la generazione della tupla di configurazione ed il suo inserimento nel centro di tuple di sistema.

```
1  EnhancedSynchACC acc = this.getContext();
2  TucsonTupleCentreId configTc =
3              new TucsonTupleCentreId("'$ENV'", "localhost", "20504");
4  [...]
5  final LogicTuple actuatorTuple =
6       new LogicTuple("createTransducerActuator",
7          new TupleArgument(fridgeTc.toTerm()),
8             new Value("it.unibo.homemanager.situatedness.FridgeTransducer"),
9                new Value("fridgeTransducer_" + device.getDeviceId()),
10                  new Value("it.unibo.homemanager.situatedness.FridgeProbe"),
11                     new Value("fridge" + device.getDeviceId()));
12 acc.out(configTc, actuatorTuple, null)
```

dove sono presenti 'ID del centro di tuple, la classe Java che implementa il *Transducer*, l'ID del *Transducer*, la classe Java che implementa il *Probe* ed infine l'ID del *Probe*.

### Caricamento delle reazioni

Il caricamento e la programmazione delle reazioni ReSpecT sono descritti dalla seguente procedura:

- *Recupero delle reazioni da file* - le specifiche delle reazioni sono state memorizzate in un file `.rsp`;
- *Sostituzione del marcatore #* - all'interno delle *reaction* è presente il marker # che sarà sostituito con il numero assegnato al dispositivo al momento del collegamento con il sistema;
- *Corretta formattazione delle reazioni* - per comodità e semplicità di lettura le reazioni all'interno del file sono indentate in una certa struttura, ma devono essere raggruppate e riformattate per poterle caricare sul centro di tuple;
- *Recupero delle eventuali reazioni già all'interno del centro di tuple* - data la complessità del sistema Home Manager e l'alto livello di automazione dato dai vari centri di tuple e Butlers è altamente possibile che il centro di tuple del dispositivo sia stato precedentemente programmato con alcune reazioni ReSpecT;
- *Unione delle reaction e caricamento sul centro di tuple* - tutte le reazioni ricavate dal file e dal centro di tuple vengono unite e poi caricate in blocco sul centro di tuple.

Tutta questa procedura è racchiusa da questo codice:

```
1 [...]
2 String path = "it/unibo/homemanager/situatedness"
3 String config = Utils.fileToString(path+"/fridgeSpec.rsp");
4 this.loadConfig(fridgeTc, config);
5 [...]
```

## 3.4 FridgeWithGuiProbe

Come anticipato in fase di progettazione si è deciso di sviluppare un secondo *Probe* con una propria interfaccia grafica (una semplice tabella) nel quale sarà possibile visualizzare tutti gli ingredienti e le relative quantità presenti nel frigorifero simulato. Tale *Probe* è stato introdotto direttamente nel sistema Home Manager e messo in comunicazione con l'agente `FridgeAgent`

mediante il centro di tuple *fridge_tc* già presente nel prototipo, questo per provare l'effettiva modularità introdotta dal *Transuducer*.
Per realizzare questo nuovo prototipo infatti le uniche modifiche eseguite sull'attuale sistema sono state quelle di inserire all'interno di `FridgeAgent` la procedura di generazione della tupla di configurazione del *Transducer* e del `FridgeWithGuiProbe` e quella del caricamento delle *reaction*, mentre il nuovo *Probe* e le reazioni che determinano il comportamento del centro di tuple a fronte delle operazioni invocate dal `FridgeAgent` sono state implementate da zero.
Come per il *Probe* precedente, anche in questo caso si è mantenuta la stessa scelta sulle funzioni `setEnv` e `getEnv`, ma in questo caso le operazioni sul centro di tuple non sono invocabili direttamente dal `FridgeWithGuiProbe`, ma dal `FridgeAgent` i cui compiti sono già stati studiati in fase di analisi e le cui possibili azioni sono riportate in Fig. 3.1 e Fig. 3.2, inoltre il centro di tuple *fridge_tc* è stato poi riprogrammato e sono state aggiunte nuove reazioni ReSpecT che permettono di mantenere aggiornato il `FridgeWithGuiProbe`.

Qui di seguito invece sono riportati il codice di implementazione dei due metoti `readValue(String key)` e `writeValue(String key, int state)` del nuovo *Probe* realizzato anch'esso come classe Java che implementa l'interfaccia `ISimpleProbe` e le reazioni ReSpecT utilizzate per programmare il centro di tuple contenute nel file `fridgeSpec2.rsp`.

Metodi `readValue` e `writeValue`:

```
@Override
public boolean readValue(String key) {
   [...]
   boolean new_ingredient = true;
   int index = 0;
   for(;index<ingredients.size();index++){
      String correct_key = ingredients.get(index).getName();
      if (key.equals(correct_key)){
         new_ingredient = false;
         break;
      }
    }
    if(!new_ingredient){
      // Remove the ingredient from the list
      Ingredient ingredient_removed = ingredients.remove(index);
```

```
16        f_frame.deleteIngredient(key);
17        //System.out.println("["+this.pid+"]-Ingredient: "+key+" removed");
18        return true;
19      }else{
20        System.err.println("[" + this.pid + "]: Can't found the ingredient!");
21        return false;
22      }
23  }
24
25  [...]
26
27  @Override
28  public boolean writeValue(String key, int state) {
29     [...]
30     boolean new_ingredient = true;
31     int index = 0;
32     for(;index<ingredients.size();index++){
33        String correct_key = ingredients.get(index).getName();
34        if (key.equals(correct_key)){
35            new_ingredient = false;
36            break;
37        }
38     }
39     if(!new_ingredient){
40        // This ingredient is already been in the fridge
41        Ingredient ing = ingredients.get(index);
42        int quantity = ing.getQuantity();
43        int new_quantity = quantity + state;
44        ingredients.get(index).setQuantity(new_quantity);
45        f_frame.addIngredient(key,new_quantity);
46        //System.out.println("["+this.pid+"]-Ingredient upgraded");
47        return true;
48     }else{
49        // New ingredient
50        ingredients.add(new Ingredient(key, state));
51        f_frame.addIngredient(key,state);
52        System.out.println("["+this.pid+"]-New ingredient: "+key+"added");
53        return true;
54     }
55  }
```

Reazioni ReSpecT:

```
1  reaction(
2     out(content(fridge_#, I, Q)),
3     (operation, completion),
4     (
5        fridge#@localhost:20504 ? setEnv(I, Q)
6     )
7  ).
8
```

```
reaction(
   in(content(fridge_#, I, Q)),
   (operation, invocation),
   (
      fridge#@localhost:20504 ? getEnv(I,Q)
   )
).

reaction(
   inp(content(fridge_#, I, Q)),
   (operation, invocation),
   (
      fridge#@localhost:20504 ? getEnv(I,Q)
   )
).

reaction(
   setEnv(I, Q),
   (from_env, completion),
   (
      out(content(fridge_#, I, Q))
   )
).

reaction(
   getEnv(I,Q),
   (from_env, completion),
   (
      in(content(fridge_#, I, Q))
   )
).
```

Di seguito è fornita anche l'interfaccia grafica completa a disposizione dell'utente dopo essersi loggato nel sistema Home Manager e quella del `FridgeWithGuiProbe`.

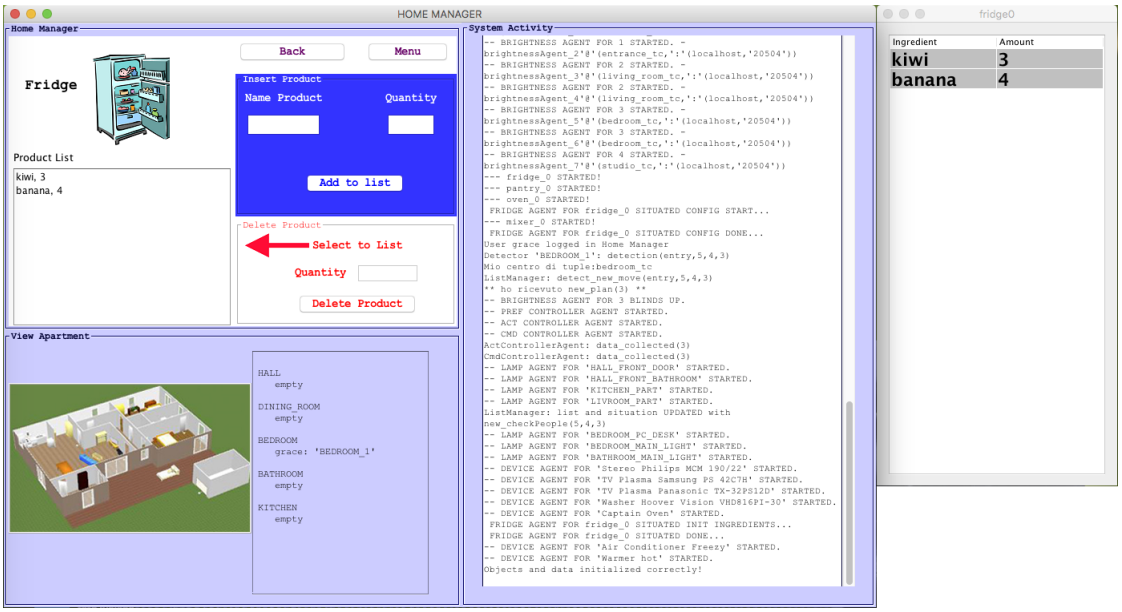


Figura 3.5: Interfaccia grafica completa

# 4 Proprietà aggiunte al sistema

L'introduzione della nozione di *Transducer* all'interno dell'attuale prototipo del sistema Home Manager ha portato al soddisfacimento di diversi requisiti non funzionali sotto il punto di vista ingegneristico, esso infatti ha aggiunto/migliorato diverse proprietà al/del sistema. Alcune di queste proprietà sono già state anticipate nel corso della tesi ma in questa sezione saranno presentate in modo più approfondito e verranno introdotte le nuove.
Tali proprietà sono:

- *Modularità* - l'introduzione del *Transducer* permette al sistema di essere modulare in quanto qualora si volesse usare un dispositivo differente da quello utilizzato gli interventi da effettuare sul codice e sul sistema sarebbero minimi. Si può addirittura pensare che, prendendo in esame il pattern Model-View-Controller, progettando opportunatamente la parte di *Controller* del *Probe* rimarranno da implementare solamente quelle di *Model* e *View* se richiesta;
- *Disaccoppiamento* - il *Transducer* fornisce una sorta di interfaccia utilizzabile per configurare i dispositivi riducendo al minimo la dipendenza del sistema dal dispositivo. Tale proprietà è strettamente correla-

ta con la precedente e quella della *Separation of concerns* presentata successivamente;

- *Separation of concerns* - date le due proprietà precedentemente descritte è possibile affermare che l'introduzione del *Transducer* permette di avere nel modello del sistema la distinzione di una sezione dedicata ai dispositivi separata da quella della logica del sistema, inoltre prendendo in considerazione il modello di sistema MAS le cui astrazioni tipiche sono agenti, società ed ambiente, i *Transducer* possono essere usati per fornire il giusto livello di astrazione per i device e per collocarli nell'astrazione di ambiente;
- *Riusabilità del codice* - diretta conseguenza della proprietà di *Modularità*
- *Configurabilità* - andando ad agire solamente sulle componenti del *Transducer* o modificando solamente le reazioni ReSpecT è possibile fornire un alto livello di configurabilità del dispositivo;
- *Semplicità* - è stata aumentata la semplicità con cui un nuovo dispositivo può interfacciarsi al sistema, facilitando inoltre l'utilizzo e la configurazione di dispositivi anche reali;
- *Usabilità* - attraverso il *Transducer* il sistema può continuare a supportare le attività di un utente anche attraverso interfacce grafiche per i dispositivi collegati.

*Modularità*, *Disaccoppiamento*, *Separation of concerns* erano già proprietà non funzionali presenti all'interno del sistema, ma attraverso l'introduzione del *Transducer* queste proprietà sono state enfatizzate e migliorate permettendo una separazione ancora più distinta delle entità del sistema coinvolte, della loro interazione e della rappresentazione delle informazioni scambiate.

# 5 Transducer e Architettura Butlers

Come più volte ripetuto all'interno della tesi, il *Transducer* ha permesso di "spostare" la gestione dei dispositivi collegati al sistema dal livello applicativo a quello di infrastruttura fornendo una implementazione ancora più completa e dettagliata dei livelli di *Information* e *Control* dell'architettura

Butlers e portando l'architettura di Home Manager a rispecchiare i livelli proposti e a basarsi in modo ancora più fedele su quella Butlers. I *Transducers* inoltre permettono di rappresentando in maniera uniforme sia gli eventi ambientali che quelli che coinvolgono gli agenti, fornendo così un'interfaccia uniforme al livello superiore, ovvero il livello di *Coordination*.

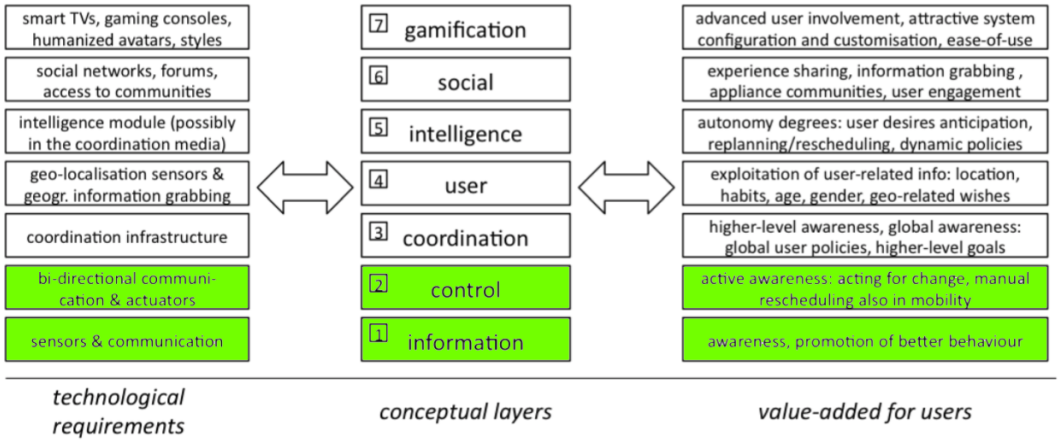


Figura 3.6: I livelli di *Information* e *Control* evidenziati.

# Capitolo 4

# Situatedness spaziale in Home Manager mobile

Nella sezione 1.3.2 si è introdotto il fatto che il prototipo attuale di Home Manager ha la capacità di esser (in parte) gestito da remoto attraverso una sua versione mobile la quale, comunicando con il sistema Home Manager fisso su pc/server consente all'utente di monitorare e gestire i dispositivi e, attivando la funzionalità di geolocalizzazione ha la capacità di rilevare la posizione dell'utente e di inviarle al sistema fisso. Quest'ultimo, una volta ricevuti i dati sulla locazione dell'utente attraverso i propri Butlers attivi nell'abitazione analizza la posizione ricevuta e al verificarsi di determinate situazioni e in base alle preferenze preimpostate dell'utente può decidere, in modo autonomo, sulla gestione dei dispositivi e notificare l'utente della decisione presa mediante notifica nell'app mobile, il quale potrà sempre modificarla se necessario.
Lo scenario rappresentato da questo insieme di azioni, quali la raccolta dei dati di locazione, il recupero delle preferenze di un utente, la valutazione dei primi in base a quest'ultimi e l'attuazione di azioni sui dispositivi in modo del tutto autonomo da parte del sistema indica ciò che nell'architettura Butlers è definito dalla combinazione dei livelli di *User* e *Intelligence*.
In tale scenario è interessante però pensare a come i concetti di centri di tuple e reazioni spaziali ed il servizio di geolocalizzazione offerto nativamente da TuCSoN possano integrarsi con quelli già esistenti in Home Manager e in alcuni casi addirittura sostituirli, portando, come già avvenuto con i *Transdu-*

*cer* per i dispositivi collegabili al sistema, la gestione delle geolocalizzazione e di certe forme di autonomia a livello di infrastruttura e non più a quello applicativo.
In questo capitolo in particolare prenderemo in esame la parte mobile del sistema, di come esso gestisce tutte le Activity e gli AsyncTask legati alla geolocalizzazione e l'invio della posizione al sistema "server" e di come esso possa esser esteso/sostituito dal supporto nativo di TuCSoN verso questo tipo di servizio e di come le reazioni spaziali possano esser d'aiuto per aumentare gli automatismi e conseguentemente l'autonomia del sistema.

# 1 Analisi

L'attuale prototipo Home Manager dell'app mobile sfrutta le API offerte da Google per poter accedere ai dati raccolti dai servizi di localizzazione, come GPS, presenti nella quasi totalità degli smartphone (Android) presenti in commercio. Questi servizi consentono alle applicazioni di ottenere aggiornamenti periodici sulla posizione geografica del dispositivo, oppure di avviare o eliminare un Intent specifio quando il dispositivo entra nella vicinanza di una determinata posizione geografica. Nella app dopo una fase di login, si accede alla Activity `MenuScelta` nella quale sono proposti all'utente differenti strumenti per la gestione dei dispositivi dell'abitazione e lo strumento per attivare il servizio di localizzazione. Quest'ultimo permette all'app mobile di recuperare la posizione dell'utente e di inviarla al sistema "server" che la analizzerà ed eventualmente elaborerà piani di azione che dipendono dalla locazione dell'utente. La posizione dell'utente all'interno del centro di tuple sarà rappresentata dalla tupla:

```
geo_position(nomeUtente, latitudine, longitudine)
```

La procedura di recupero della posizione dell'utente e relativo invio sul centro di tuple in esecuzione sulla versione di Home Manager presente nell'abitazione è la seguente:

- All'apertura della Activity `MenuScelta` vengono istanziati un oggetto di tipo `Location`, classe che rappresenta una posizione geografica e

nel quale verrà salvata la posizione dell'utente, un oggetto di tipo `LocationManager`, classe attraverso la quale fornisce l'accesso ai servizi di localizzazione del sistema e infine viene salvata l'ultima posizione conosciuta dell'utente;

```
[...]
private Location location;
private LocationManager lManager;
[...]
   @Override
   protected void onCreate(Bundle savedInstanceState){
      [...]
      lManager =
         (LocationManager)this.getSystemService(Context.LOCATION_SERVICE);
      location =
         lManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
      [...]
   }
[...]
```

- Alla scelta dell'attivazione dei servizi di localizzazione, effettuata mediante il bottone dedicato nella finestra del menù di scelta, viene richiamato il metodo privato `getCurrentLocation()`, nel quale viene inizializzato il listener, di tipo `LocationListener`, incaricato di ricevere gli aggiornamenti sulla posizione inviatigli dal `LocationManager` e viene avviato il servizio vero e proprio di localizzazione attraverso il metodo `lManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,0,0,lListener)`. Una volta avviato il servizio si esegue il controllo sulla posizione che esso ha ritornato, in caso negativo si avvisa l'utente mediante un Toast, un widget che permette di notificare all'utente in modo estremamente semplice e con una durata di visualizzazione simile ad una notifica un messaggio in seguito a qualche operazione compiuta dall'utente, mentre in caso di successo vengono estratte le coordinate e vengono passate come parametri all'AsyncTask `LocalizzazioneTask`;

```
   [...]
   private NotificationManager notificationManager;
   [...]
   private LocationListener lListener;
   Context saved = this;
   String nome;
```

```
7   [...]
8   @Override
9   protected void onCreate(Bundle savedInstanceState) {
10     [...]
11     final Activity att = this;
12     [...]
13
14     ImageButton bottLoc= (ImageButton) findViewById(R.id.localizza);
15     bottLoc.setOnClickListener(new OnClickListener() {
16        @Override
17        public void onClick(View v) {
18           getCurrentLocation();
19           if(location==null) {
20              Toast.makeText
21                 (saved,"No coordinate",Toast.LENGTH_SHORT).show();
22           }else{
23              String s=""+location.getLatitude();
24              String str= s.substring(0,s.length()-7);
25              float lat = Float.parseFloat(str);
26              s =""+location.getLongitude();
27              str= s.substring(0,s.length()-1);
28              float lon = Float.parseFloat(str);
29              LocalizzazioneTask loc =
30                new LocalizzazioneTask(att,saved,notificationManager,nome);
31              loc.execute(lat, lon);
32           }
33        }
34     });
35  }
36
37  [...]
38
39  private void getCurrentLocation() {
40     lListener = new LocationListener() {
41        [...]
42     };
43     lManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,0,0,lListener);
44  }
45  [...]
```

- E' nell'AsyncTask che viene creato l'agente TuCSoN incaricato di depositare la tupla indicatrice la posizione dell'utente nel centro di tuple dedicato (*infoTc*) in esecuzione sul sistema "server".
  Nell'attuale prototipo poi si è implementata la funzionalità che recupera le tuple che rappresentano i servizi presenti nelle vicinanze dell'utente. Da tali tuple poi verranno estrapolate tutte le informazioni attraverso il metodo `getFromLocation` offerto dalla classe `Geocoding`, il quale permette attraverso un processo di geocoding inverso di tra-

sformare le coordinate (latitudine, longitudine) di un punto in un indirizzo (parziale). Tali informazioni verranno poi visualizzate all'utente all'interno dell'Activity specifica.

```
[...]
@Override
protected void onPreExecute() {
   super.onPreExecute();
   geo= new Geocoder(c, Locale.getDefault());
}

@Override
protected String[] doInBackground(Float... params) {

   lat=params[0] ;
   lon= params[1];
   String[] resultList=null;
   try {
      this.tc =
       new TucsonTupleCentreId("infoTc",Costanti.getIndirizzo(),"20504");
      this.agent = new TucsonAgentId("locAg");
      this.acc =
       TucsonMetaACC.getContext(this.agent,Costanti.getIndirizzo(),20504);
      // Store the geo_position tuple to the right tuple center
      Value longVal= new Value(lon);
      Value latVal= new Value(lat);
      final LogicTuple arr =
       LogicTuple.parse("geo_position("+this.userName+",'"
                  +latVal+"','"+longVal+"')");
      final ITucsonOperation op = this.acc.out(this.tc, arr,Long.MAX_VALUE);
      if (op.isResultSuccess()) {
         Log.i("Result: ", "Succes");
      } else {
         Log.i("Result: ", "Something gone wrong");
      }
      [...]
      // Recover the services found by HM ``server''
      Thread.sleep(10000);
      tcList =
       new TucsonTupleCentreId("serviceTc",Costanti.getIndirizzo(),"20504");
      LogicTuple arr1 = LogicTuple.parse("service("+ new Var("Z")+")");
      final ITucsonOperation op1 =
         this.acc.rdAll(tcList, arr1,Long.MAX_VALUE);
      if (op1.isResultSuccess()){
        List<LogicTuple> list = op1.getLogicTupleListResult();
         resultList = new String[list.size()+2];
         List<Address> addresses = null;
         try{
            addresses = geo.getFromLocation(lat, lon, 1);
         } catch (IOException e){}
         [...]
```

```
48          }
49       }
50    }
51    [...]
```

# 2 Progetto

Come abbiamo già visto nelle sezioni 2.1.4 e 2.1.5 l'infrastruttura di coordinazione TuCSoN mette a disposizione nativamente diversi supporti per la coordinazione situata e servizi di geolocalizzazione, pertanto viene spontaneo domandarsi come essi possano essere sfruttati all'interno dell'attuale prototipo di Home Manager, in particolare per quanto riguarda il prototipo dell'app mobile su cui è principalmente focalizzata la percezione di geolocalizzazione. Dall'analisi precedentemente eseguita abbiamo appurato che attualmente il sistema mobile utilizza esplicitamente le API di Google il GPS dello smartphone come servizio di localizzazione per ricavare la posizione dell'utente e successivamente si incarica sfruttando un agente TuCSoN di inviare le coordinate ricevute sottoforma di tupla al centro di tuple *infoTc*. Tramite TuCSoN ciò potrebbe essere realizzato in maniera del tutto automatica, le uniche operazioni da eseguire sarebbero quelle della creazione di un centro di tuple all'interno del dispositivo mobile, dell'avvio del servizio di geolocalizzazione mediante l'invocazione dell'operazione `out(createGeolocationService(Sid, Sclass, Stcid))` sul centro di tuple *geolocationConfigTC* associando al servizio il centro di tuple precedentemente creato e programmandolo secondo delle specifiche di comportamento realizzate tramite le *spatial reaction*. In questo modo sarà possibile implementare comportamenti specifici del sistema a fronte degli spostamenti dell'utente eseguibili in modo del tutto automatico e trasparente all'utente portando così un incremento della autonomia del prototipo attuale.

A tale scopo si è deciso di avviare il nodo TuCSoN all'interno dell'app mobile una volta effettuata la procedura di login e selezionata l'opzione localizzazione attraverso il bottone già presente all'interno del prototipo, mediante l'utilizzo delle librerie TuCSoN `alice.tucson.service.TucsonNodeService`, creare un centro di tuple `Stcid` che sarà utilizzato per raccogliere gli eventi

spaziali generati (`Stcid` verrà poi sostituito dall'id scelto in fase di implementazione) e riutilizzare l'attuale agente TuCSoN presente nell'AsyncTask `LocalizzazioneTask` per avviare il servizio di geolocalizzazione tramite l'invocazione della primitiva `out(createGeolocationService(Sid, Sclass, Stcid))`. Sarà poi sempre compito di quest'agente, analogamente a come già visto per i *Transducer*, caricare le specifiche di coordinazione per configurare il comportamento del centro di tuple in cui verranno salvate le posizioni ricevute dal servizio di geolocalizzazione e successivamente "inizializzare" quest'ultimo invocando la primitiva specifica `out(init(geo_service))` che attraverso una reaction precedentemente caricata permette di salvare la posizione corrente dell'utente. Le specifiche di comportamento caricate inoltre permetterebbero di reagire agli eventi spaziali generati alla partenza e all'arrivo da/in un luogo da parte dell'utente e consentirebbero di andare a salvare in automatico tali dati sul centro di tuple nel centro di tuple *infoTc* in esecuzione sul sistema Home Manager "server", il quale poi le elaborerà ed eseguirà eventualmente un piano di azione sulla gestione dei dispositivi dell'abitazione.

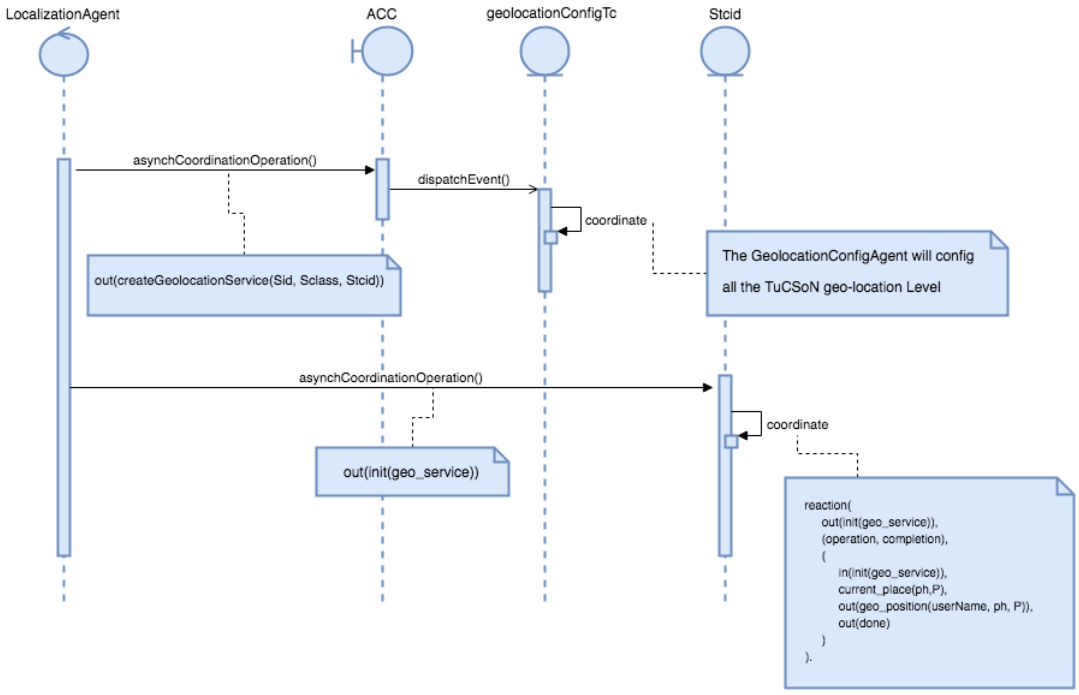


Figura 4.1: Operazione di configurazione del servizio di geolocalizzazione

Nella figura seguente invece sono rappresentati i due comportamenti del sistema a fronte degli eventi spaziali `from` e `to` provenienti dal livello di geolocalizzazione di TuCSoN.

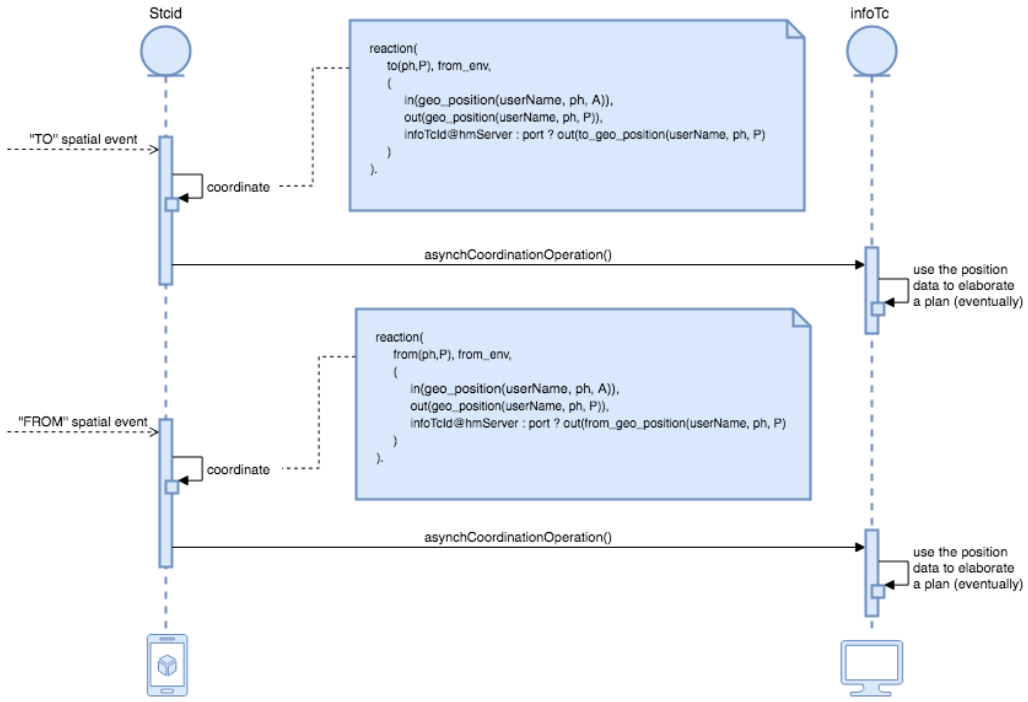


Figura 4.2: Comportamento del prototipo a fronte degli eventi spaziali *from* e *to*

## 2.1 Reaction

Qui si riportano già le *reaction* utilizzate così da facilitare il lettore nella comprensione del funzionamento del prototipo e delle Fig. 4.1. Tali reazioni fanno riferimento al file `geoSpec.rsp` e comprendono anche quelle utilizzate nelle dinamiche successive alla fase di configurazione e inizializzazione del servizio di geolocalizzazione. Inoltre, la stringa `userName` verrà sostituita dal nome dell'utente col quale si è loggato al sistema, `infoTcId` rappresenterà il `TucsonTupleCentreId` del centro di tuple *infoTc*, `hmServer` e `port` rispettivamente l'indirizzo del nodo su cui è in esecuzione il centro di tuple e la porta di ascolto.

```
reaction(
   out(init(geo_service)),
   (operation, completion),
   (
      in(init(geo_service)),
      current_place(ph, P)
      out(geo_position(userName, ph, P),
      out(done)
   )
).

reaction(
   out(done),
   (internal),
   (
      rd(geo_position(userName, ph, P),
      infoTcId@hmServer : port ? out(geo_position(userName, ph, P),
      in(done)
   )
).

reaction(
   to(ph, P),
   from_env,
   (
      in(geo_position(userName, ph, A),
      out(geo_position(userName, ph, P),
      infoTcId@hmServer : port ? out(to_geo_position(userName, ph, P)
   )
).

reaction(
   from(ph, P),
   from_env,
   (
      in(geo_position(userName, ph, A),
      out(geo_position(userName, ph, P),
      infoTcId@hmServer : port ? out(from_geo_position(userName, ph, P)
   )
).
```

Nelle *reaction* che rappresentano reazioni sensibili agli eventi spaziali `from` e `to` è facile notare che le tuple che saranno salvate sul centro di tuple *infoTc* all'interno del sistema Home Manager "server" saranno differenziate da un prefisso di appartenenza, questa decisione è stata presa per distinguere ulteriormente il possibile comportamento che potrebbe conseguirne una volta eleborati i dati ricevuti ed elaborato un piano di gestione dei dispositivi da parte del sistema, separando concettualmente le possibili azioni che possono

essere eventualmente eseguite in quelle che prevedono come scenario la partenza da un luogo da parte dell'utente (vedi partenza da casa o dalla sede di lavoro ecc.) e quelle eseguibili invece all'arrivo di un luogo (vedi arrivo al lavoro o al ristorante preferito ecc.).

# 3 Implementazione

Durante la fase di progettazione si è deciso di reimplementare ed estendere l'AsyncTask `LocalizzazioneTaskin` questo modo:

- Creazione ed installazione del nodo TuCSoN;
- Creazione del centro di tuple positionTc;
- Creazione dell'agente locAg;
- Caricamento delle specifiche di comportamento;
- Avvio del servizio di geolocalizzazione;
- Inizializzazione di positionTc;

Questa implementazione permetterà di aumentare il livello di autonomia del sistema Home Manager mobile, di fornire uno strumento di interfacciamento agli strumenti di geolocalizzazione, disaccoppiando il servizio stesso di basso livello che sfrutta il GPS dello smartphone dal livello applicativo del prototipo. Inoltre, attraverso l'utilizzo di questo livello e delle *spatial reaction* tutti gli aggiornamenti sulla locazione dell'utente vengono interpretati come eventi provenienti dall'ambiente in cui gli agenti ed il sistema Home Manager è situato. Di seguito verranno presentati le porzioni di codice che implementano le azioni appena descritte.

## 3.1 Creazione ed installazione del nodo TuCSoN

Per poter sfruttare il livello di geolocalizzazione ènecessario installare sul dispositivo un nodo TuCSoN e tale livello, come già descritto precedentemente nella sezione 2.1.5 permette di interfacciarsi ad un servizio che ricava la posizione dell'utente mediante l'uso del GPS dello smartphone. L'installazione del nodo è implementata come segue:

```
[...]
// Create and install TuCSoN node
this.tns = new TucsonNodeService(portn);
tns.install();
[...]
```

## 3.2 Creazione del centro di tuple positionTc

Il servizio di localizzazione necessita di un centro di tuple a cui essere associato, per tanto per tenere traccia della posizione dell'utente si è deciso di creare un centro di tuple sul dispositivo mobile denominandolo *positionTc*, il quale permetterà anche di catturare gli eventi spaziali generati sempre dal livello di geolocalizzazione e, in base alla specifica di comportamento caricata, reagire ad essi col fine di eseguire azioni in modo del tutto autonomo.

```
[...]
// Create positionTc where to save position
this.tc = new TucsonTupleCentreId("positionTc", localIp, portn);
Log.i("Tuple center: ", this.tc.toString());
Log.i("Ip address: ", this.tc.getNode());
Log.i("Port number: ", String.valueOf(this.tc.getPort()));
[...]
```

## 3.3 Creazione dell'agente locAg

Per poter attivare la funzione di geolocalizzazione offerta da TuCSoN è necessario che un agente invoca la primitiva `out(createGeolocationService(Sid, Sclass, Stcid))` sul centro di tuple textitgeolocationConfigTC, dove `Sid` rappresenta l'identificatore, del tipo `GeoServiceId`, con cui identificarlo, `Sclasse` la classe che implementa il servizio e `Stcid` il centro di tuple, nel nostro caso `positionTc` su cui verranno salvate le posizioni dell'utente e generati gli eventi spaziali. L'agente inoltre dovrà "inizializzare" il centro di tuple andando a salvare la posizione corrente dell'utente invocando la primitiva `out(init(geo_service))` e sfruttando le *reaction* settate sul centro di tuple.

```
[...]
// TuCSoN agent
this.agent = new TucsonAgentId("locAg");
Log.i("Agent: ", this.agent.toString());
this.acc = TucsonMetaACC.getContext(this.agent, localIp, portn);
[...]
```

## 3.4 Caricamento delle specifiche di comportamento

Il centro di tuple *positionTc* dovrà essere programmato mediante il caricamento delle specifiche di comportamento rappresentate dalle *reazioni spaziali* contenute all'interno del file `geoSpec.rsp`, nel quale, come descritto precedentemente verranno sostituite le stringhe `userName`, `infoTcId`, `hmServer` e `port` assegnandogli i valori specifici definiti dall'implementazione.

```
[...]
String config =
     Utils.fileToString("com/example/myapphomemanager/MyTask/geoSpec.rsp");
loadConfig(tc, config);
[...]
```

## 3.5 Avvio del servizio di geolocalizzazione

Il servizio di geolocalizzazione deve essere avviato mediante l'invocazione della specifica tupla presentata precedentemene e sfrutta l'agente `GeolocationConfigAgent`, il quale viene avviato contestualmente all'avvio del nodo TuCSoN ed è definito per gestire le richieste di creazione e rimozione dei servizi di localizzazione.

```
[...]
// Launch the geolocation service
GeoServiceId sid = new GeoServiceId("geo_service"));
String sclass = "it.unibo.tucson.android.geolocation.AgentGeolocationService";
final LogicTuple init =
     LogicTuple.parse("createGeolocationService("+sid+","+sclass+","+this.tc+")");
final ITucsonOperation op =
      this.acc.out( geolocationConfigTC, init ,Long.MAX_VALUE);
if (op.isResultSuccess()) {
   Log.i("Geo Service = ", "Started");
} else {
   Log.i("Geo Service = ", "Init failed");
}
[...]
```

## 3.6 Inizializzazione di positionTc

Una volta avviato il servizio di geolocalizzazione si è pensato fosse necessario salvare la posizione corrente dell'utente e per realizzare ciò si è deciso di

sfruttare le *reaction* caricabili su centro di tuple *positionTc*. Alla ricezione della primitiva `out(init(geo_service))` triggera la prima reazione presentata nella sezione 2.1 e attraverso il predicato di osservazione `current_place(ph, P)` viene ricavata la posizione corrente dell'utente che verrà successivamente salvata nel centro di tuple nella forma `geo_position(userName, ph, P)`.

```
[...]
// Initialize positionTc
final LogicTuple init_pos = LogicTuple.parse("init(geo_service");
final ITucsonOperation op = this.acc.out( this.tc, init_pos ,Long.MAX_VALUE);
if (op.isResultSuccess()) {
   Log.i("Position tc = ", "initiated");
} else {
   Log.i("Position tc = ", "Init failed");
}
[...]
```

# 4 Caso di studio

Nel seguente caso di studio si vuole affrontare uno scenario applicativo per inquadrare ciò che è stato presentato nella sezione di implementazione precedente. In questo scenario prevediamo che l'utente decida di fermarsi nella sua pizzeria preferita, il sistema mobile sul quale sarà già attivo il sistema di geolocalizzazione segnalerà la fermata attraverso l'evento spaziale `to` indicando la posizione in cui si è fermato l'utente, le reazioni spaziali che programmano il centro di tuple *positionTc* poi salveranno la posizione sul dispositivo e invocheranno la primitiva specifica sul centro di tuple *infoTc* presente sul sistema "server" di Home Manager. Qui, attraverso altre reazioni e *reazioni di linking* sarà possibile recuperare tutti i servizi preferiti dell'utente salvati precedentemente sul centro di tuple *favouriteTc*. Infatti le *reazioni di linking* sono utilizzate quando sono i centri di tuple ad invocare le operazioni e è necessario gestire le tuple di ritorno nel flusso delle operazioni innescate da una reazione del centro di tuple che ha invocato l'operazione. Una volta recuperati i servizi preferiti dell'utente infatti essi verranno gestiti da una seconda reazione che verificherà che la posizione ricevuta sia nell'immediata vicinanza di una tra quelle preferite e nel caso di successo verrà posta una tupla `service(`*`serviceName`*`)` che sarà analizzata da un Butlers. Quest'ul-

timo riconosciuto il servizio come la pizzeria preferita dell'utente agirà in modo del tutto autonomo, anticipando il bisogno dell'utente di accendere il forno per scaldare la pizza una volta rientrato in casa, andando ad accendere il forno e notificando l'azione all'utente che riceverà una notifica sul proprio dispositivo, che avrà sempre la possibilità di modificare tale scelta. Di seguito è rappresentato il caso di studio nello scenario precedente in cui Home Manager non possedeva ancora la nozione di coordinazione spaziale.

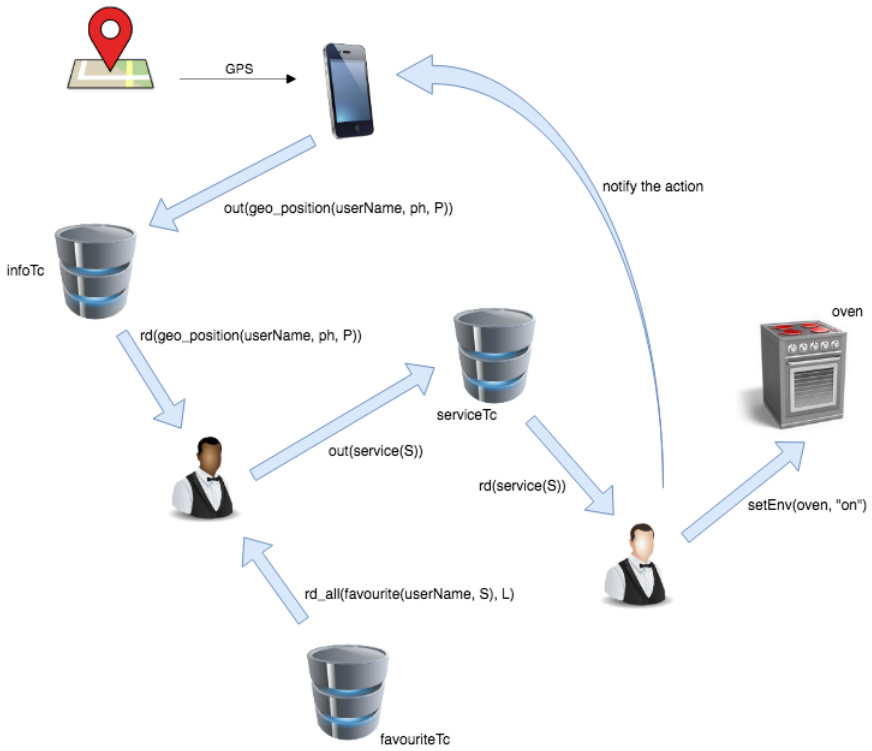


Figura 4.3: Scenario senza coordinazione situata

In figura si nota come senza l'introduzione della coordinazione spaziale siano i Butler a dover interrogare i centri di tuple e solo successivamente agire in base ai risultati recuperati, invece nella figura sottostante si comprende appieno come l'inserimento dei livello di *Geolocation* e l'utilizzo di centri e reazioni spaziali aumentino il grado di autonomia. Infatti l'elaborazione dei possibili piani a fronte di spostamenti di locazione dell'utente sarà eseguita a fronte di un evento e sarà composta quasi interamente da operazioni eseguite a fronte di reazioni stimolate.

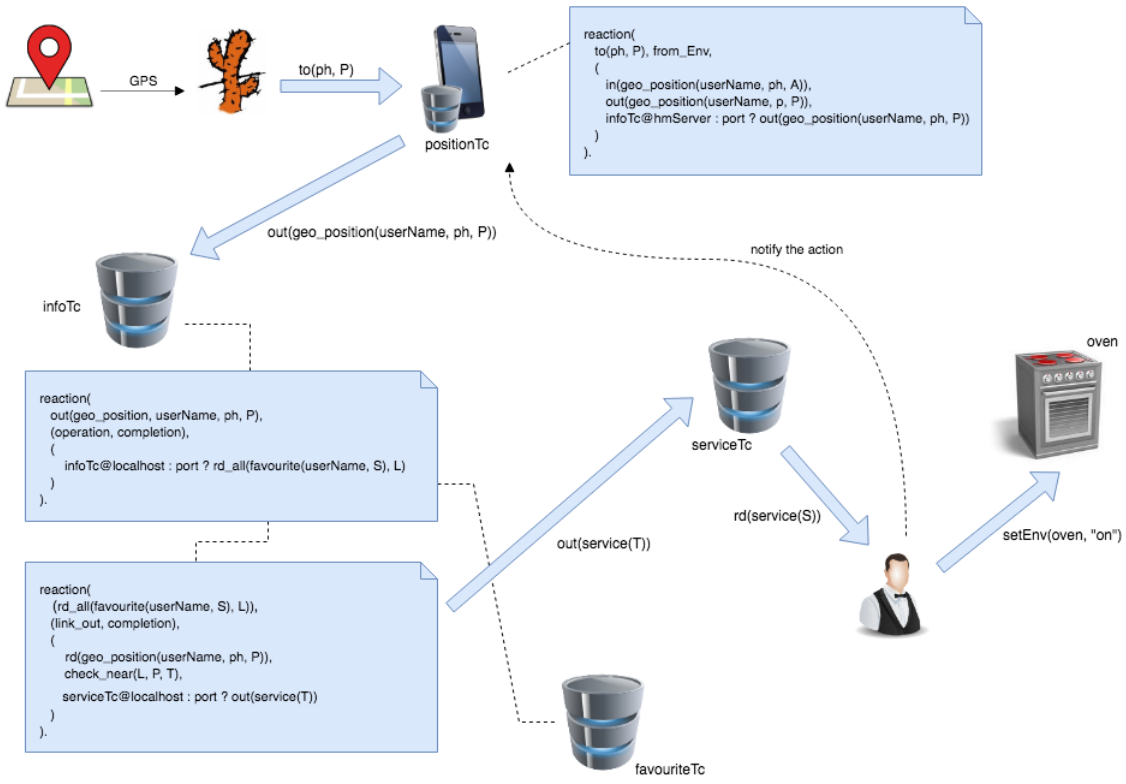


Figura 4.4: Scenario con implementazione della coordinazione situata

Riportiamo inoltre le *reaction* pensate per lo sviluppo del caso di studio caricate sul centro di tuple *infoTc*.

```
1  reaction(
2    out(geo_position, userName, ph, P),
3    (operation, completion),
4    (
5       infoTc@localhost : port ? rd_all(favourite(userName, S), L)
6    )
7  ).
8
9  reaction(
10   (rd_all(favourite(userName, S), L)),
11   (link_out, completion),
12   (
13      rd(geo_position(userName, ph, P)),
14      check_near(L, P, T),
15      serviceTc@localhost : port ? out(service(T))
16   )
17 ).
```

# 5 Proprietà aggiunte al sistema

Con l'utilizzo del supporto offerto da TuCSoN per la geolocalizzazione e l'introduzione dei centri di tuple e reazioni spaziali il prototipo Home Manager Mobile ha potuto godere di alcuni benefici. Questi benefici sono già stati anticipati nel corso di questo capitolo e rappresentano proprietà che portano al soddisfacimento di diversi requisiti non funzionali che un sistema può avere, tali proprietà sono:

- *Disaccoppiamento* - il livello *Geolocation* di TuCSoN, presentato nella sezione 2.1.5, ed utilizzato per l'implementazione del concetto di coordinazione situata all'interno dell'app mobile del sistema Home Manager, ha permesso di fornire una interfaccia che garantisce un collegamento tra la parte applicativa e qualsiasi piattaforma di geolocalizzazione;
- *Configurabilità* - come già affermato nella proprietà precedente il livello di *Geolocation* permette di interfacciarsi con qualsiasi servizio di geolocalizzazione e ciò rende il sistema estremamente configurabile sotto il punto di vista della sensoristica utilizzata per recuperare le informazioni sulla posizione dell'utente;
- *Autonomia* - l'utilizzo delle *reazioni spaziali* ha permesso di implementari meccanismi che esulano dall'utilizzo degli agenti stessi e permettono di realizzare comportamenti complessi in modo del tutto autonomo;
- *Trasparenza* - il livello di *Geolocation* inoltre permette di percepire tutti gli aggiornamenti inviati dalla piattaforma di geolocalizzazione come eventi ambientali fornendo quindi un meccanismo di trasformazione e rappresentazione dei dati ricevuti che dal punto di vista della società degli agenti che costituisce il sistema Home Manager risulta del tutto trasparente.

# Conclusioni e sviluppi futuri

In questa tesi si è cercato di introdurre il concetto di coordinazione situata nell'ambito dell'IoT prendendo in considerazione come riferimento il prototipo Home Manager, un sistema di gestione di una smart home basato sull'architettura Butlers.
Per iniziare si sono introdotti i concetti di Internet of Things e AmI, presentando ed analizzando tutte le loro caratteristiche e punti chiave, in modo da apprendere le nozioni di base che mi avrebbero guidato nell'analisi del sistema Home Manager e nella progettazione delle soluzioni proposte in seguito.
Successivamente si è analizzata l'architettura Butlers andando a descriverne i livelli concettuali di riferimento con i quali consente di mettere in relazione le caratteristiche e le tecnologie presenti in un'applicazione con il valore aggiunto percepito dall'utente. In tale architettura l'astrazione principale è quella del *butler*, un componente specializzato in una certa attività con un alto livello di intelligenza che, attraverso l'analisi delle preferenze dell'utente e delle sue necessità, permette al sistema di agire in modo autonomo col fine di soddisfare tali esigenze ed anticiparne i bisogni. In particolare si è analizzato come tale astrazione sia implementata nel sistema Home Manager attraverso l'utilizzo della metodologia SODA (Societies in Open Distributed Agent environments) e il meta-modello A & A (Agents & Artifacts) e come essa sfrutti i meccanismi di coordinazione offerti dall'infrastruttura di coordinazione TuCSoN.
Si è proseguito con lo studio dell'infrastruttura TuCSoN nel quale si è cercato di sviluppare i punti ritenuti fondamentali al fine della tesi, soffermandosi quindi su caratteristiche, linguaggio di coordinazione alla sua base, architettura dell'infrastruttura e i concetti alla base della coordinazione situata, ovvero *centri di tuple spaziali* e gestione della *geolocalizzazione*.

Una volta terminato lo studio di TuCSoN e di tutte le sue nozioni fondamentali si è preso in considerazione il prototipo Home Manager analizzando i punti in cui le nozioni e gli strumenti precedentemente acquisiti potessero essere introdotti per fornire una forma di coordinazione situata al prototipo stesso e portare a miglioramenti generali e dal punto di vista ingegneristico al sistema complessivo.
Il primo aspetto su cui ci si è focalizzati è stato quello della gestione dei dispositivi collegabili al sistema i quali sono stati trattati attraverso il componente *Transducer* offerto da TuCSoN. Esso ha permesso di avere una interfaccia di configurazione per qualsiasi dispositivo collegabile, di rappresentare le interazioni da e verso di essi mediante le due primitive di coordinazione *setEnv* e *getEnv* e di programmare comportamenti complessi attraverso *reaction* specifiche.
Il secondo aspetto è stato quello della geolocalizzazione che nel prototipo Home Manager risiede nell'app mobile per smartphone Android. In essa si è inserito il livello di *Geolocation* offerto da TuCSoN che ha permesso di offrire all'app un layer di interfacciamento alle piattaforme di geolocalizzazione che sfruttano il GPS del dispositivo per inviare aggiornamenti sulla posizione dell'utente, e sono state aggiunte *reazioni spaziali* che hanno fornito meccanismi di autonomia più avanzati di quelli già presenti.
L'introduzione del concetto di coordinazione situata all'interno del sistema Home Manager ha quindi portato a quest'ultimo miglioramenti in termini di *disaccoppiamento*, *separation of concerns*, *configurabilità*, *autonomia* e *trasparenza*, e nel caso della gestione dei dispositivi anche di *riusabilità del codice*, *semplicità* e *usabilità*. L'introduzione dei *Transducer* ha inoltre permesso di "spostare" la gestione dei dispositivi dal livello applicativo a quello della infrastruttura fornendo un'implementazione ancora più dettaglia dei livelli di *Information* e *Control* dell'architettura Butler su cui Home Manager si basa.
Per entrambi gli aspetti presi in considerazione sono state presentate soluzioni implementative con relativi casi di studio, i quali potrebbero essere presi come punto di partenza per sviluppi futuri che riguardino l'integrazione totale in Home Manager del concetto di coordinazione situata.

# Bibliografia


[1] Atzori L., Iera A., Morabito G. - *The internet of things: A survey.* Computer Networks, 54(15), 2787-2805. doi:10.1016/j.comnet.2010.05.01, October 2010;

[2] INFSO D.4 Networked Enterprise, RFID INFSO G.2 Micro, Nanosystems with the Working Group RFID of the ETP EPOSS - *Internet of Things in 2020, Roadmap for the Future.* Version 1.1, May 2008;

[3] Tschofenig H. et. al. - *Architectural Considerations in Smart Object Networking.* Tech. no. RFC 7452. Internet Architecture Board, March. 2015; `https://www.rfc-editor.org/rfc/rfc7452.txt`

[4] Ferrari L. , Zambonelli F. - *Agents and Ambient Intelligence: the LAICA Experience.* Università di Modena e Reggio Emilia, Italy

[5] Denti E. - *Novel pervasive scenarios for home management: the Butlers architecture.* SpringerPlus number 52, pages 1-30, ISSN 2193-1801, 25 January 2014
`http://springerplus.springeropen.com/articles/10.1186/2193-1801-3-52`

[6] Denti E. , Calegari R. - *Butler-ising HomeManager: A Pervasive Multi-Agent System for Home Intelligence.* 7th International Conference on Agents and Artificial Intelligence 2015 (ICAART 2015), pages 249-256, January 2015;

[7] Omicini A. , Zambonelli F. - *"Coordination for Internet Application Development".* Autonomous Agents and Multi-Agent Systems 2(3), September 1999;

[8] Omicini A. , Ricci A. , Viroli M. (2005b). - *RBAC for organisation and security in an agent coordination infrastructure*. Electronic Notes in Theoretical Computer Science, 128(5), pages 65-85, 2nd International Workshop on Security Issues in Coordination Models, Languages and Systems (SecCo'04), August 2004;

[9] Omicini A. , Mariani S. - *"The TuCSoN Coordination Model and Technology. A Guide"*. TuCSoN v. 1.12.0.0301, Guide v. 1.3.1, June 2015;
`http://www.slideshare.net/andreaomicini/the-tucson-coordination-model-technology-a-guide`

[10] Omicini A. - *"On the Semantics of Tuple-based Coordination Models"*, ACM Symposium on Applied Computing (SAC '99), San Antonio (TX), 28 February - 2 March 1999, ISBN 1-58113-086-4

[11] Denti E. , Natali A. , Omicini A. - *"On the expressive power of a language for programming coordination media"*, in: Proceedings of the 1998 ACM Symposium on Applied Computing (SAC '98), Atlanta (USA) 27 February - 1 March 1998;

[12] Casadei M. , Omicini A. - *Situated tuple centres in ReSpecT*. In Shin, S. Y., Ossowski, S., Menezes, R., and Viroli, M., editors, 24th Annual ACM Symposium on Applied Computing (SAC 2009), volume III, pages 1361-1368, Honolulu, Hawaii, USA. ACM

[13] Mariani S. , Omicini A. - *Advanced Coordination Techniques: Experiments with TuCSoN and ReSpecT*. Dipartimento di Informatica - Scienza e Ingegneria (DISI) Alma Mater Studiorum - Università di Bologna & Faculté d'informatique - Université de Namur, 28 April 2016;

[14] Mariani S. , Omicini A. - *Space-aware Coordination in ReSpecT*. Dipartimento di Informatica - Scienza e Ingegneria (DISI) Alma Mater Studiorum - Università di Bologna, Proceedings of the 14th Workshop "From Objects to Agents", co-located with the 13th Conference of the Italian Association for Artificial Intelligence (AI*IA 2013), CEUR Workshop Proceedings 1099, pp. 1-7, 2-3 December 2013;

[15] Bombardi M. - *Coordinazione space-aware per dispositivi mobili in TuCSoN*;
`http://amslaurea.unibo.it/6350/`

[16] Pometto A. V. - *Coordinazione situata per la domotica: Butlers in TuCSoN*;
`http://amslaurea.unibo.it/9564/`