Semantic Smart Room

<github_repo_here>

Progettazione, creazione e l'uso delle Smart Things all'interno di una stanza, utilizzando metodologie di descrizione semantica per ricavare conoscenza sul sistema analizzato

Marcin Pabich
0000835166

Gennaio 2021

Università di Bologna
Ingegneria e Scienze informatiche
Anno Accademico 2019/2020

Indice

Elenco delle figure

Elenco dei codici

Capitolo 1

Introduzione

Il crescente interesse verso Internet of Things e un esponenziale crescita nell'utilizzo di apparecchiature elettroniche onnipresenti eterogenee permette l'esplorazione e l'adattamento di diversi approcci per la realizzazione e lo sviluppo di soluzioni Smart. Questo studio si pone come l'obiettivo quello di analizzare le possibilità date e realizzare un progetto che unisca il mondo del Web Semantico e quello di Pervasive Computing, circoscritti nei contesti Smart quotidiani (quali ad esempio ambienti casalinghi o aziendali).

Avendo come caratteristica quella di essere una via di mezzo tra lo studio ed effettiva prototipazione, la seguente relazione si comporrà di diversi punti trattanti in modo dettagliato ogni aspetto che incide sulle decisioni da intraprendere. Le conclusioni tratte da ogni analisi effettuata permetteranno poi la realizzazione di un applicativo dimostrante le capacità di un sistema progettato per sfruttare la conoscenza a priori.

I seguenti punti riassumono la scaletta individuata:

- introduzione all'IoT, Smart Things e standard W3C per la definizione di oggetti e le loro funzionalità all'interno di una Smart Room;
- analisi nel campo del Semantic Web per descrivere la conoscenza relativa agli oggetti e all'ambiente in cui si trovano;

- creazione degli oggetti (virtualmente o fisicamente) aventi caratteristiche rilevate nei precedenti punti;
- focus sulle metodologie a disposizione per sfruttare la conoscenza descritta;
- creazione di un progetto sfruttante tutte le conclusioni pregresse, avente come l'obiettivo la dimostrazione della capacità di un sistema di scoprire dinamicamente oggetti e capirne le potenzialità (anche d'insieme), grazie all'uso della descrizione standardizzata di oggetti.

Ponendo in ordine cronologico le diverse fasi, si ritiene opportuno utilizzare questa relazione anche come un backlog della conoscenza che man mano risulta essere scoperta ed analizzata.

Ove possibile, nell'intera struttura dell'analisi e del progetto, verranno preferiti, come scelta principale, standard emergenti e/o relativamente nuovi, in modo da poter capirne le potenzialità ed eventualmente limiti. Inoltre, trattandosi di un'esplorazione, è più corretto puntare su concetti nuovi, piuttosto che modelli vecchi o "troppo" confermati: motivo per il quale molte delle tecnologie di oggi risultano essere non completamente all'altezza dei requisiti posti rispetto alle funzionalità che dovrebbero offrire.

Capitolo 2

Definizioni e riferimenti

2.1 Internet Of Things

L'ambito di Internet of Things [19] risulta essere particolarmente interessante per diversi motivi, nei quali, tra i principali, si può trovare la possibilità di interazione tra le diverse cose collocate all'interno di un environment. Esistono già numerose applicazioni che sfruttano le potenzialità delle Smart Things, ma le soluzioni proposte riguardano molte volte standard proprietari che non possono interagire direttamente con le diverse soluzioni presentate anche dai relativi competitor.

Già nel 2010 i dispositivi connessi in rete erano più del doppio del numero della popolazione mondiale: la costante esponenziale crescita di questo fenomeno sottolinea l'importanza che esso assume nel tempo, non solo per quanto riguarda l'ambito user, ma anche le aziende e le città.

L'internet delle cose (Internet of Things) è un concetto rappresentate una possibile evoluzione del web: le cose (Things) all'interno di un sistema si rendono scopribili ed accessibili, in modo da garantire l'interoperabilità e il continuo flusso dei dati. Gli oggetti così connessi possono diventare intelligenti e acquisire a loro volta dati provenienti da fonti eterogenee, altri oggetti

compresi. L'insieme delle informazioni percepite diventa quindi conoscenza, e può essere sfruttata sia da calcolatori che da umani.

Per "oggetti" [26] si può intendere una qualsiasi cosa capace di rendersi visibile e comunicante all'interno di una rete: si parla quindi di "Smart Things" quando questi possiedono le capacità di comunicare e di essere scoperti da altri.

La definizione di cosa sia effettivamente l'IoT ha passato diversi stage di revisione, nella quale si è potuto osservare una graduale progressione verso una visione sempre più ampia e complessa. Partendo infatti da semplici oggetti smart, come un condizionatore connesso, si è arrivati ad un sistema in cui è il termostato ad osservare l'ambiente; esso reagisce quindi ai cambiamenti e, comunicando poi le azioni da attuare ad una serie di oggetti connessi, mantiene un certo stato dell'environment in cui si trova. Un ulteriore evoluzione è quella dell'integrazione di questi sistemi con assistenti virtuali, oggetti eterogenei e entità esterne alla propria casa: il tutto in una visione più ampia di un mondo completamente connesso, nel quale l'interazione con i device smart sarà sempre di più facente parte della vita di un singolo e della società nella quale vive.

IoT risulta essere una rete capace di creare un estratto virtuale del mondo reale, nel quale le cose collaborano tra di loro estrapolando ed acquisendo conoscenza, interagendo tra di loro e modificando l'ambiente nel quale si trovano. Attraverso l'uso estensivo di sensori, attuatori e oggetti eterogenei si procederà alla racconta di un enorme quantità di dati: a fronte di una profonda analisi di quest'ultimi si creerà un modello sempre più ottimizzato di ambienti nei quali si vive o si lavora, rendendo le azioni quotidiane più organizzate e ottimizzate da diversi punti di vista, quali costi, tempi e fatica. L'interazione tra le cose avverrà in maniera sempre più invisibile, tale da rendere le Smart Things parte integrante della vita quotidiana, non ac-

corgendosi nemmeno quando e dove esse vengono sfruttate. Un progresso graduale permetterà quindi di ottenere un'unica rete, formata da uomini, cose e intelligenze artificiali, collaboranti tra di loro.

2.1.1 Problematiche

Esistono diversi ambiti di studio che confermano sia le potenzialità che i rischi legati all'applicaione estensiva dell IoT. Il principale problema tecnico è quello relativo all'eterogeneità delle Things che possono essere interconnesse tra di loro e i relativi standard di comunicazione (emergenti e presenti) che vi possono essere. Esistono infatti diversi protocolli di comunicazione già esistenti e definiti, ognuno dei quali possiede i propri vantaggi e svantaggi. Il problema che ne deriva è l'impossibilità di averli implementati tutti all'interno di un singolo dispositivo. Inoltre, per utilizzarne alcuni, come ZigBee, vi è necessità di possedere un hub centrale.

Oltre ai diversi protocolli, ogni produttore di Smart Things implementa uno standard proprietario di comunicazione, il quale generalmente richiede l'utilizzo di un'app o hub apposta, non parlante con dispositivi dei competitor. Si ritrova quindi molto spesso a dover rimanere legati ad un singolo manufacturer, nonostante le varie alternative, a volte nettamente migliori, presenti sul mercato.

I principali problemi di natura umana, invece, risulta essere quello della privacy e della proprietà, ovvero di chi alla fine possiede i dati. Tenendo in considerazione una visione ampia, nella quale tutti i dispositivi parlano tra di loro, inevitabilmente di mezzo vi sono dati riguardanti ogni individuo. Facendo un esempio classico, i dati biometrici di un individuo potrebbero essere benevolmente usati per monitorare lo stato della salute (ad esempio, si pensi agli anziani): un'automazione potrebbe essere fatta quando vi è necessità di chiamare soccorsi o informare urgentemente la famiglia di un accaduto. Questi dati sono sensibili, e dev'essere garantito un loro corretto utilizzo congruo

con l'obiettivo che hanno da raggiungere: ciò che, alla fine, è stato tentato dal GDPR.

Collegandosi poi al tema della proprietà dei dati, si può pensare ad esempio ad una macchina agricola smart, nella quale vi sono diversi sensori che comunicano il loro stato all'utilizzatore. Parlando dunque di privacy e proprietà, vi è un acceso dibattito sul chi effettivamente possiede quei dati, in quanto potrebbe averne accesso anche il produttore della macchina agricola: in questo caso si assisterebbe ad un'analisi comportamentale degli individui, favorendo la prevenzione dei guasti e l'ottimizzazione delle features più richieste a discapito del monitoraggio delle abitudini degli utilizzatori, le quali, dipendentemente dal contesto, potrebbero rendere identificabile un soggetto, anche se in partenza i dati risulterebbero anonimi.

2.2 W3C

Per risolvere (in parte) problematiche relative ai competitor con i diversi metodi di comunicazione, l'ente standardizzante W3C [28], insieme alla collaborazione di alcune marche famose, ha instaurato il **WoT** [30]. Il Web of Things risulta essere un set di standard per quanto riguarda la garanzia dell'interoperabilità tra le Smart Things, che si basa sulla definizione di oggetti con descrizioni, eventi e proprietà, disponibili grazie all'identificazione tramite URI. I livelli di approfondimento di una struttura stabilita dall'ente possono variare e partono dal classico esempio d'uso di un utente semplice, arrivando a gestire casi di Smart Environment in aziende e la definizione dei Digital Twin per l'osservazione, analisi e simulazione dei comportamenti.

Uno dei primi progetti risulta essere quello della Thing Description [27], che permette, tramite l'uso di un file in formato JSON, la definizione del protocollo, delle proprietà osservabili e delle azioni in input e output che può

svolgere un determinato oggetto. Ogni Smart Thing, che può essere raggiunta nella rete, deve includere al suo interno un riferimento a questa descrizione, in modo tale da permettere l'aggiunta di nuovi device all'ambiente smart senza doversi preoccupare della configurazione ad hoc per ogni apparecchio.

La definizione della Thing Description offre numerosissimi vantaggi, non solo dal punto di vista pratico. Essa costituisce infatti un livello di conoscenza su quello che un particolare device è capace di fare ed esporre: grazie a questa è possibile ricavare ulteriore conoscenza con delle analisi più approfondite, entrando più nell'ottica del Web Semantico e dati connessi. Esistono infatti già ontologie standard, come SAREF [23], le quali grazie alle annotazioni, che è possibile includere all'interno della Thing Description, rendono ancora più leggibili e standard le descrizioni, in modo da poter riconoscere particolari gruppi di device e le loro funzionalità.

2.3 Smart Room

Una stanza "Smart" [25] è un ambiente che utilizza al suo interno più oggetti smart, interconnessi tra di loro. Grazie all'Internet of Things, essi sono capaci di fornire features di diverso tipo, partendo da quelle semplici richiedenti l'input dell'utente, arrivando a quelle intelligenti senza che vi sia un'esplicita richiesta per l'attuazione. Nel caso di studio in Pervasive Computing, non si tratta dunque soltanto di una stanza con alcuni oggetti smart all'interno di essa, comandabili tramite lo smartphone, ma Things che riescono a percepire l'ambiente e decidere autonomamente e coordinatamente azioni da svolgere per accomodare il loro utilizzatore nel miglior modo possibile, in modo da creare un ambiente unico dove tecnologia e umani condividono lo stesso environment, interagendo tra di loro.

Una Smart Room fornisce features personalizzabili in base all'utente che ne usufruisce. L'interazione oggetto-utente è generalmente stabilita grazie

all'uso di un terminale, oggetto dedicato o smartphone, e permette in modo semplice, intuitivo e veloce di configurare tutti i parametri richiesti. Le Things al'interno di una Smart Room devono riuscire ad essere collegabili tra di loro e necessitano di parlarsi per scoprire i loro stati attuali, proprietà e possibili azioni da svolgere. Un utente è inoltre libero di aggiungere un numero virtualmente infinito di oggetti all'interno della stanza, senza rompere la stabilità del sistema o comprometterne le funzionalità, anzi, fornendo più possibilità alle cose di interagire e creare nuove funzionalità, anche non previste al momento della creazione della Thing.

2.3.1 Problematiche

Le stesse problematiche che riguardano il mondo dell'IoT possono essere trovate all'interno di una Smart Room, ad esempio considerando classico problema dell'interoperabilità tra oggetti eterogenei e i diversi modi nei quali essi potrebbero voler comunicare.

I problemi che possono essere invece aggiunti in questo ambito riguardano più la personalizzazione da parte di diversi utenti e comportamenti discordanti quando più soggetti con preferenze differenti si trovano all'interno di uno stesso ambiente. Un esempio classico potrebbe essere riferito a due soggetti che hanno una preferenza discordante sull'intensità dell'aria condizionata, dove il primo utilizzatore preferisce un funzionamento moderato, mentre il secondo non lo gradisce affatto. Se entrambi si troveranno all'interno di uno stesso ambiente, il sistema dev'essere in grado di stabilire la situazione ottimale, facendo valere le preferenze e priorità.

2.4 Conoscenza, ontologie e RDF

L'eterogeneità delle informazioni è intrinsecamente presente in ogni ambiente preso in considerazione: vi sono infatti molteplici eventi, entità, pro-

prietà di esse e azioni che possono essere eseguite. Per un umano è quasi immediato riconoscere immagini o situazioni, contestualizzando ogni volta l'evento osservato per poter trarne conclusioni corrette. La cosa non si rende vera per quanto riguarda le macchine o sistemi automatici: i contenuti resi a disposizione sono perlopiù human-readable, ma non sono immediatamente capibili dai calcolatori; essi, infatti, spesso non possiedono un background sufficiente, tale da poter individuare situazioni, oggetti o eventi particolari senza essere prima preaddestrati in un set determinato e limitato di riconoscimenti da svolgere.

Una mancanza di contesto e di informazioni collegabili tra di loro rende il lavoro dei computer, per quanto riguarda l'analisi del mondo che circonda gli umani, complessa e macchinosa. Per questo motivo sin dall'anno 2007 vi sono diversi movimenti per spingere l'adozione sempre più grande dei "Linked Data" [10]: dati sui quali si possono eseguire query semantiche. Quest'ultime non sono nient'altro che un modo per le macchine di ragionare sulla conoscenza che hanno a disposizione, non più scritta in modo da essere capita soltanto da un umano. La rappresentazione dei dati dunque, non riguarda più soltanto quello che si può sapere, ma anche la derivazione di nuovi dati a partire da quelli esistenti, per creare una fonte potenzialmente infinitamente profonda di collegamenti tra tutte le informazioni che si hanno a disposizione, abbattendo i cosiddetti muri virtuali che separano attualmente la conoscenza posseduta.

Le ontologie [12] sono la raccolta dei concetti dato un determinato ambito, mentre il formato più comune per scrivere conoscenza è RDF [13]. L'obiettivo di questi è poter mantenere un modello di dati semplici, che permetta a chiunque la rappresentazione di un qualsiasi concetto. Il tutto è basato sul principio di "*triplette*", le quali possiedono il *"soggetto"*, *"predicato"* e *"l'oggetto"* interessato, in quello che può essere definito come un grafo etichettato che esprime dunque le relazioni che intercorrono tra le cose. La localizzazione

delle risorse deve avvenire tramite l'uso di URI ed è possibile specificare tipi di dati (anche nuovi). Globalmente parlando, ogni risorsa creata può essere referenziata da altre, in modo tale da creare una struttura connessa e unica, con tutta la conoscenza che fino a quel momento si aveva.

Capitolo 3

Web Semantico e Smart Things

In questo capitolo verranno analizzati le possibilità date a disposizione per descrivere le Things, in modo da poter utilizzare in modo interoperabile la conoscenza che presentano.

3.1 Parametri della conoscenza

Per essere considerata valida, una conoscenza dev'essere descritta in modo tale da presentare alcune caratteristiche:

- **efficienza**: il modo in cui descrive gli oggetti è ben definito e non risulta essere complesso nell'utilizzo;
- **completezza**: l'oggetto descritto deve esserlo in ogni suo aspetto, indicante tutte le sue caratteristiche;
- **leggibilità**: la descrizione deve poter essere letta dal soggetto che la voglia leggere (umano o macchina, difficilmente entrambi);
- **compatibilità**: la descrizione deve poter essere letta da diversi soggetti non legati per forza ad un singola famiglia di dispositivi e/o umani;
- **univocità**: la descrizione dev'essere letta e compresa ogni volta allo stesso modo;

- manutenibilità: dev'essere possibile aggiungere, rimuovere e/o modificare la descrizione senza comprometterne completamente la struttura;
- reperibilità: dev'essere possibile ottenere la descrizione dell'oggetto in modo agevole e standard;

È importante sottolineare questi aspetti, in quanto anche una banale descrizione testuale di un oggetto può essere ritenuta inizialmente come una conoscenza che riguarda quell'oggetto. Purtroppo il commento del produttore contiene di solito un linguaggio indirizzato verso il consumatore, per convincerlo delle potenzialità di quel dispositivo rispetto alla concorrenza, ma non descrivente (o nascondente) alcuni importantissimi dettagli riguardanti l'implementazione effettiva delle sue feature.

Di conseguenza, è già possibile definire a priori che linguaggi naturali non possono essere utilizzati per descrivere le things in modo univoco e leggibile da parte di tutti.

3.2 Modelli conosciuti

In letteratura sono noti diversi modi per trascrivere e trasmettere conoscenza. Un esempio banale è semplicemente questa relazione, dentro la quale l'utilizzo di un linguaggio noto permette ad altri umani (capaci di leggere la lingua utilizzata) di esplorare il testo ed estrarre informazioni da esso. Ovviamente, per una macchina è difficile poter analizzare un linguaggio naturale, presentante oltretutto diverse ambiguità e criticità; a volte vi è l'assenza di un esplicito contesto, altre volte vi possono essere diverse interpretazioni dello stesso testo letto da individui diversi e/o traduzioni non esatte in altri linguaggi, a causa di forme sintattiche differenti.

Esistono di conseguenza alcuni modelli definiti e conosciuti per poter ovviare alle problematiche descritte.

- Un modo per poter descrivere la conoscenza rispettando tutti i parametri indicati è l'utilizzo di RDF e RDFS: col primo, si possono esprimere concetti base per la descrizione e/o definizione di *istanze* delle classi, nel secondo i concetti per la descrizione e/o definizione di queste classi.

- Un'altro modo può essere quello di utilizzare OWL, che utilizza una semantica simile a RDFS, ma che aggiunge alcune funzionalità e restrizioni per rendere l'esplorazione e l'inferenza di conoscenza in modo più agevole.

- Infine, è possibile definire un proprio standard per la definizione della conoscenza, in modo da essere facilmente scrivibile e leggibile in un formato standard (come XML-based o JSON-based).

Generalmente parlando, quando si crea della conoscenza nell'ambito del Web Semantico, si definisce un'ontologia [12]: una raccolta di concetti in un determinato ambito. Grazie a questa è possibile ricavare, capire e creare nuova conoscenza, combinando opportunamente attributi noti e collegabili ad altre ontologie. Uno scenario simile si addice perfettamente a ciò che risulta essere

Tra tutte le scelte descritte, solo l'ultima risulta essere molto meno indicata rispetto alle altre; una propria definizione di una descrizione non standard porterebbe presto a diverse difficoltà di lettura, manutenzione e completezza, nonché introdurrebbe un nuovo diverso e incompatibile linguaggio con quelli già esistenti. Di conseguenza, è lecito pensare che le prime due possano essere prese in considerazione, in base a quelli sono i requisiti dell'applicazione che poi si vorrà realizzare.

Nonostante possa sembrare che gli altri due modelli descritti siano la scelta migliore da compiere, nell'ambito di Web Semantico e Smart Things vi è un emergente standard proveniente direttamente da W3C [28]: si tratta infatti della **Thing Description** [27], la quale ha come l'obiettivo principale quello di rendere agevole la descrizione, la lettura e l'inferenza delle informa-

zioni su quello che è l'oggetto preso in considerazione. Anche se coi modelli descritti sarebbe possibile descrivere completamente una Smart Thing scelta, l'utilizzo di uno standard nato esattamente per coprire questo ambito risulta essere altamente raccomandato, onde evitare di ricadere nella ridefinizione di principi già esistenti e confermati dalla comunità.

Essendo questo studio relativo a metodologie nuove ed emergenti (come descritto nel capitolo 1), è più corretto ai fini dello stesso progetto concentrasi quindi sull'analisi di quest'ultima metodologia. È importante però sottolineare sin dal principio che nonostante la Thing Description sia un formato riconducibile (in parte) al terzo elemento dell'elenco, vi è la possibilità (se non la necessità) di includere ontologie esistenti all'interno della descrizione, in modo da poter standardizzare non solo le informazioni riguardanti l'oggetto stesso ma anche di attributi che lo compongono (ad esempio, riferendosi ai nomi delle classi di cui attributi possono esserne le istanze).

3.3 Thing Description

Nella sezione precedente è stato possibile individuare il candidato per questa successiva analisi, che pone in evidenza i punti salienti e criticità del modello selezionato. È stato scelto di non descrivere i modelli rimanenti per via delle scelte intraprese nel capitolo 1, puntando direttamente su uno standard coprente essenzialmente tutto l'ambito di cui si sta trattando in questo capitolo.

Una **Thing Description** è il blocco centrale di costruzione del Web of Things e si compone principalmente di 4 componenti che descrivono:

- la Thing stessa,
- le funzionalità della Thing,
- il set di classi e attributi usati per facilitare il machine-understandability,

- collezione di Web Link che esprimono collegamenti con ulteriori oggetti o descrizioni.

Generalmente una Thing Description viene creata utilizzando il linguaggio JSON. Un esempio di una descrizione può essere trovato nel Codice 3.1

```
{
    "@context": "https://www.w3.org/2019/wot/td/v1",
    "id": "urn:dev:ops:32473-WoTLamp-1234",
    "title": "MyLampThing",
    "securityDefinitions": {
        "basic_sc": {"scheme": "basic", "in":"header"}
    },
    "security": ["basic_sc"],
    "properties": {
        "status" : {
            "type": "string",
            "forms": [{"href": "https://mylamp.example.com/status"}]
        }
    },
    "actions": {
        "toggle" : {
            "forms": [{"href": "https://mylamp.example.com/toggle"}]
        }
    },
    "events":{
        "overheating":{
            "data": {"type": "string"},
            "forms": [{
                "href": "https://mylamp.example.com/oh",
                "subprotocol": "longpoll"
            }]
        }
    }
}
```

Codice 3.1: Esempio di una Thing Description

Anche se non può essere propriamente ritenuta una rappresentazione della conoscenza, come ad esempio lo può essere una qualsiasi ontologia scritta in RDFS/OWL, la Thing Description costituisce un giusto compromesso tra la facilità di implementazione e l'espressività offerta. Utilizzando JSON (un formato ampiamente sfruttato e supportato) il modello può essere letto anche da semplici interpreti per conoscere la thing stessa, senza dover ricorrere all'uso di strumenti sofisticati per il ricavo della conoscenza. Dall'altra parte però, l'importanza di poter includere all'interno di essa ontologie già definite, e di sfruttare le definizioni in esse contenute, amplifica notevolmente il significato intrinseco della conoscenza che questa descrizione porta con se: infatti,

trattandosi di uno standard già definito, è del tutto verosimile aspettarsi una lenta e graduale adozione di questa struttura per un internet sempre più connesso e "cosciente" di ciò che sta rappresentato.

Di conseguenza, anche se di primo impatto essa non sembra legata al mondo del Web Semantico, in un certo senso ne rappresenta l'estensione: si può considerarla come un "utilizzatore" di una conoscenza già posseduta che può essere tranquillamente riutilizzata per Smart Things simili fra loro. La somiglianza può, di conseguenza, essere un buon punto di partenza per ricavare nuova conoscenza, anche in casi di assenza della Thing Description: infatti, si potrebbe ad esempio assumere che, qualsiasi oggetto avente la funzionalità di impostare una luminosità, possa effettivamente essere una fonte di luce che, opportunamente regolata, può portare all'effetto voluto dall'utilizzatore: questo indipendentemente dalla natura dell'oggetto stesso, che potrebbe benissimo essere una semplice radiosveglia con dei LED.

È interessante sottolineare che la documentazione ufficiale [27] non è l'unica fonte di riferimento alla quale vi è possibile attendere: esistono versioni, anche più semplici, dello stesso standard descritti in modi diversi. Una di queste, che potrebbe diventare uno spunto per le successive analisi, risulta essere quella di Mozilla, che tratta la Thing Description come parte integrante della sua visione di Web of Things. L'intero documento, denominato **Web Thing API** [29], descrive non solo lo standard stesso, ma anche il modo nel quale poi l'oggetto potrà comunicare. È da notare però che il documento in considerazione è al momento indicato come non ufficiale, di conseguenza soggetto a modifiche.

Per ulteriori dettagli sul modello, si rimanda alla documentazione ufficiale [27].

3.4 Ontologie note

Considerando l'adozione della Thing Description come entità per avere a disposizione le informazioni riguardanti un oggetto, è utile pensare sin dall'inizio alla conoscenza già posseduta nel campo dell'IoT. Avendo la possibilità di includere ontologie all'interno della stessa TD, è opportuno pensare che esistano delle raccolte di dati già ben strutturate e pronte all'uso, in modo da rendere più agevole l'esplorazione della conoscenza che si va a creare. Inoltre, la sola Thing Description, come definito nella precedente sezione, non è sufficiente per essere ritenuta una conoscenza valida per il mondo del Web Semantico.

Una grande caratteristica delle ontologie è il fatto di poter essere collegate tra di loro: la conoscenza forma così un unico sistema nel quale si possono estrarre o inferire informazioni: ad esempio, sapendo che l'oggetto in questione è una lampadina, anche senza una Thing Description si può assumere che essa abbia modi per accendersi, spegnersi ed eventualmente regolare la propria intensità. Anche una semplice descrizione di cosa sia effettivamente l'oggetto può essere considerata conoscenza utile: sapendo ad esempio di avere una certa tipologia di climatizzatore, si può inferire il modo in cui debba essere eseguita la sua manutenzione, basandosi sulla tecnologia e/o componenti che usa internamente.

Le principali ontologie che possono essere incluse all'interno di una Thing Description sono:

- **SAREF** [23]: si basa sulla definizione di un *Device*, il quale avrà degli obiettivi da raggiungere (ad esempio, lavare i panni o accendere la luce). Per completare questi task ha bisogno di un sottoinsieme di semplici azioni, che sono standardizzate all'interno dell'ontologia stessa. In questo modo, gli oggetti possono compiere azioni anche combinate tra di loro per raggiungere obiettivi prima irraggiungibili, condividendo

le proprie capacità di (inter)agire.

Capitolo 4

Agenti e il mondo WoT

Capitolo 5

Agenti e il mondo WoT

5.1 Definizione di Agenti

In letteratura, si possono trovare diversi esempi di definizioni di un Agente, due di esse citate di seguito:

> *An agent is a computer system that is situated in some environment and that is capable of autonomous action in this environment in order to meet its design objective" [Wooldridge & Jennings, 1995]*

> *An agent is anything that can be viewed as perceiving its environment through sensors and acting upon the environment through effectors." [Russell & Norvig, 1995]*

In poche parole, l'Agente risulta essere un'entità situata all'interno di un ambiente che riesce a percepirlo ed è capace di intraprendere azioni (anche autonome) per modificare il proprio stato (o quello dell'ambiente) attraverso l'uso delle sue capacità o degli oggetti che ha a disposizione, raggiungendo così i propri obiettivi.

Gli agenti possono essere classificati in diversi modi:

- **accessible** (o inaccessible) - stabilisce se l'agente ha l'accesso allo stato completo dell'ambiente,
- **deterministico** (o non deterministico) - quando un cambio dello stato dell'ambiente è determinato univocamente dal suo stato attuale e le azioni eseguite dagli agenti,
- **statico** (o dinamico) - indica se lo stato dell'ambiente può cambiare a fronte di un azione dell'agente,
- **discreto** (o continuo) - specifica se il numero di percezioni e azioni sia limitato
- **centralizzato** (o distribuito) - stabilisce se il sistema degli agenti sè localizzato in una stessa struttura o meno,
- **generalizzato** (o specializzato) - indica se il sistema degli agenti possiede delle azioni ben definite o indipendente dai tipi di azioni che possono essere svolte.

5.1.1 Agenti BDI

Calandosi nel contesto di studio, vengono definiti meglio quelli che sono Agenti BDI.

BDI è l'acronimo di ***b****eliefs*, ***d****esires*, ***i****ntentions* (letteralmente "credenze, desideri, intenzioni") e definisce il modello secondo il quale vengono visti e definiti gli agenti. L'approccio seguito è basato su *practical reasoning*, composto da due diverse fasi:

- **deliberation** - decidere quale stato raggiungere o mantenere,
- **means-end reasoning** - decidere come raggiungere o mantenere quello stato.

L'agente BDI è un'entità **proattiva**, persistendo nelle intenzioni che attualmente possiede e **reattiva**, nel caso in cui debba abbandonarle e/o cambiarle in seguito ad una modifica delle circostanze interne e/o esterne.

Per schematizzare il ciclo di esecuzione di un Agente BDI si può far riferimento alla seguente figura:

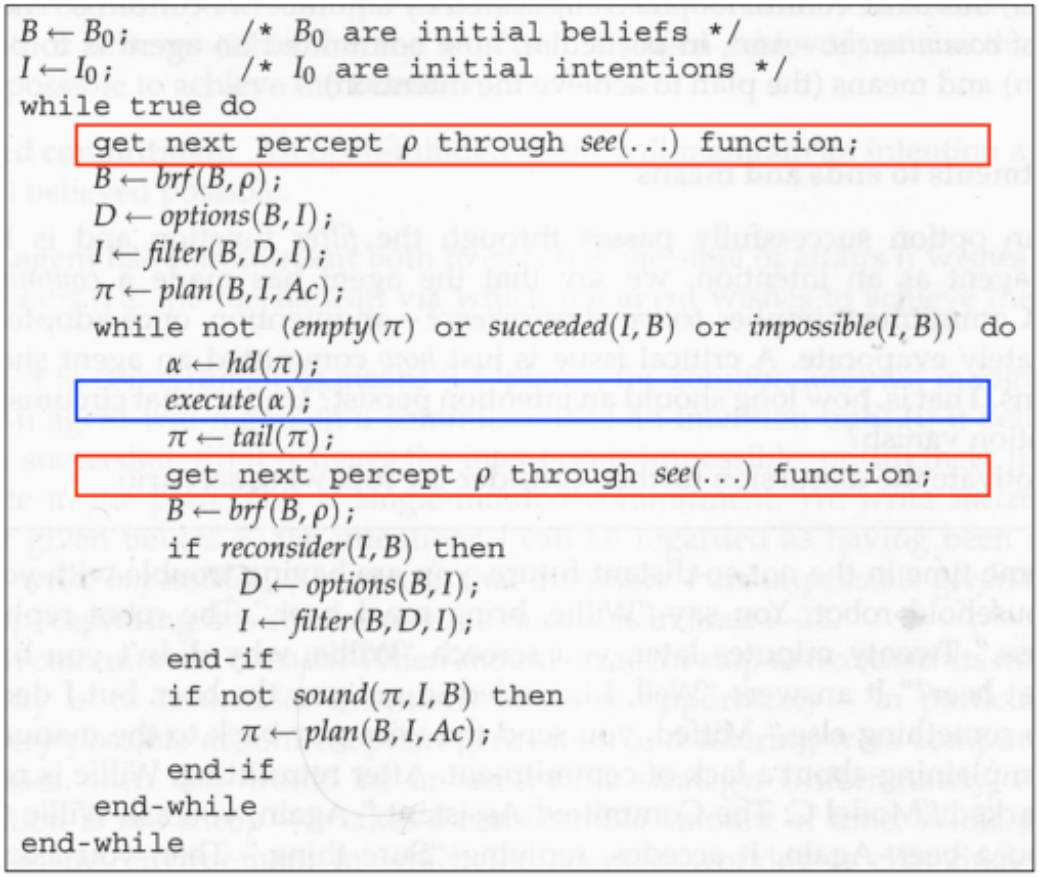

Figura 5.1: Ciclo di esecuzione di un Agente BDI.

L'utilizzo di un Agente BDI ha bisogno di una sua codifica: per poter programmare gli agenti occorre utilizzare linguaggi come AgentSpeak, o meglio ancora Jason e vi è necessità di calarli all'interno di un ambiente definito, in quanto non è possibile utilizzarli esattamente come se fossero una "semplice" libreria: il modo in cui il programmatore deve interfacciarsi dunque

sfrutta non più un classico paradigma, come quello ad oggetti, ma piuttosto reimmagina il sistema in una visione effettivamene ad Agenti.

5.1.2 JaCaMo

Come definito nella sezione precedente, gli Agenti hanno necessità di ritrovarsi all'interno di un ambiente per poter funzionare correttamente e poter parlare tra di loro. Esistono diverse librerie che permettono di istanziare environment pronti ad ospitarli, ma nel caso di studio si preferisce concentrare ad un particolare sistema basato sull'interazione Agenti-Artefatti-Ambiente chiamato JaCaMo.

JaCaMo [9] è un acronimo formato dall'incrocio di tre parole: Jason, CArtAgO [1] e Moise [11]. Si tratta di tre componenti che interagiscono tra di loro per gestire al meglio un sistema ad Agenti. Ogni componente è responsabile di un compito:

- Jason si occupa della definizione di agenti,
- CArtAgO gestisce la creazione degli Artefatti e la gestione della comunicazione Agente-Artefatto, il tutto grazie all'uso di Java
- Moise definisce l'organizzazione degli agenti attraverso l'uso di file *.xml*.

Tutto il pacchetto è strutturato in modo tale da far collaborare sempre un modulo con l'altro: si possono infatti mischiare i concetti ed estendere funzionalità degli agenti definendo alcune features a livello di Java e non solo tramite Jason. Bisogna però porre attenzione a non violare i principi di una buona programmazione ad agenti e trattare sempre le cose non-agent come delle blackbox da utilizzare soltanto in caso di estrema necessità, quando è impossibile scomporre ulteriormente il problema (ad esempio, una GUI non dovrebbe essere utilizzata direttamente all'interno di un Agente, ma necessita di essere un vero Agente/Artefatto con il quale gli altri agenti possono interfacciarsi).

5.2 Agenti nel contesto IoT

Il sistema ad Agenti nasce per diverse esigenze, che comprendono diversi ambiti. Il principale motivo per cui è più utile pensare ad un sistema eterogeneo organizzandolo più in modo Agent-Oriented è la completa indipendenza dell'esecuzione di ogni singolo componente (Agenti incapsulano anche il controllo, il ciclo di vita di uno è indipendente dagli altri) e la possibilità di astrarre a livello ancora più elevato rispetto alla OOP. Inoltre il modello ad Agenti offre la possibilità di rappresentare ancor da più vicino il mondo reale che ci circonda, in quanto questo è effettivamente formato da diverse entità (umani e/o macchine) che interagiscono continuamente tra di loro.

Parlando quindi soltanto in termini di praticità e progettazione, è già molto più facile per un programmatore immaginarsi le Smart Things come delle entità (Agenti/Artefatti) immersi dentro una stanza (Environment) che interagiscono tra di loro ed, eventualmente, hanno una gerarchia tale da permettere un pieno controllo delle azioni che vengono svolte all'interno dell'ambiente. Immaginando dunque di dover realizzare un sistema IoT, si pone davanti ad un problema di organizzazione delle Smart Things e della comunicazione tra di loro. Davanti ad uno studio di una Smart Room, si presuppone di avere alcuni oggetti Smart con le loro caratteristiche, con le quali l'utente può interagire. Dovendo pensare ad un modello ad alto livello, si può a produrre lo schema, visibile nella figura 5.2, come illustrazione di ciò che potrebbe avvenire dentro questo ambiente.

Pensandolo nel modo classico, ognuno dei sistemi rappresentati dovrebbe avere una classe che lo definisce, metodi con i quali posso interagire e caratteristiche specifiche per ogni singolo oggetto, eventualmente derivate. Una progettazione di classi rappresentative però non è l'unico step da eseguire, in quanto, successivamente, ognuna di esser dovrebbe esporre protocolli di comunicazione, dipendenti dall'oggetto in atto. Vi è poi necessità di un sistema che collezioni le istanze delle classi, e richiami opportunamente gli

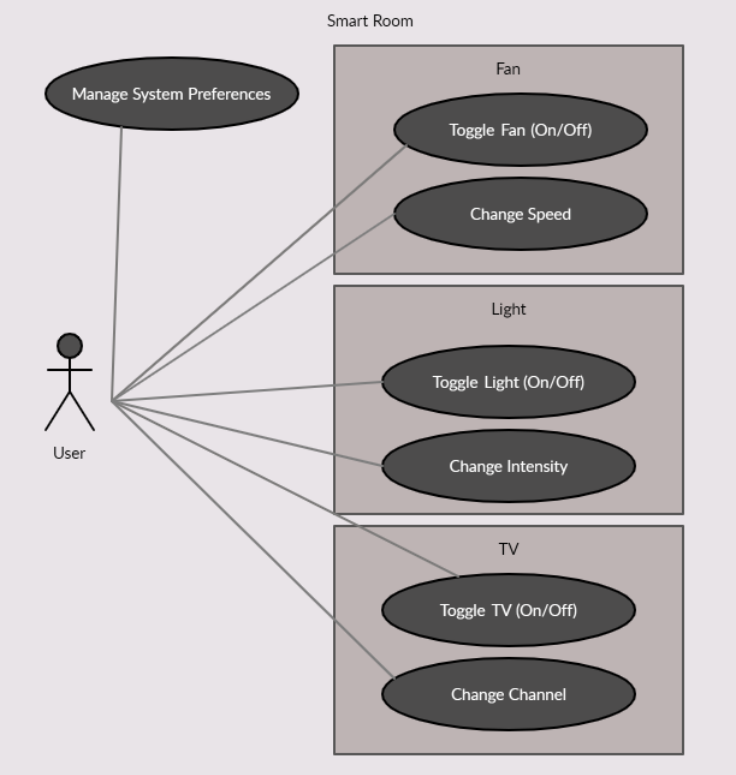

Figura 5.2: Rappresentazione ad alto livello di una Smart Room e suoi oggetti.

oggetti con il quale l'utente vuole interagire. Infine, è necessario poter vedere le preferenze dell'utente, in base alle quali gli oggetti immersi dentro il sistema dovrebbero poter (anche autonomamente) decidere come accontentare al meglio l'utente che le sta sfruttando.

Si può volgere di conseguenza lo sguardo alla descrizione presente sopra e calarla all'interno di un ambiente Agent-Oriented, dove i vari oggetti Smart sono Agenti/Artefatti. In questo modo si può astrarre ancor di più la progettazione, non calandosi direttamente alla rappresentazione tramite classi, ma proprio come entità del mondo vero calate quasi direttamente in quello ad Agenti. Come citato in "*Agent-based modeling: Methods and techniques for simulating human systems*" [17], la modellazione ad agenti possiede tre

benefici:

- cattura meglio i fenomeni che avvengono tra entità,
- fornisce una descrizione più naturale del sistema,
- risulta essere più flessibile.

È quindi di estrema semplicità immaginarsi un sistema Agent-Oriented ad alto livello, incentrato sulle interazioni Utene-Oggetti. Calandosi nel mondo JaCaMo, infatti, per ogni Smart Thing si può definire il suo Artefatto; successivamente, un agente addetto all'osservazione di questi potrà monitorare le preferenze dell'utente e interagire anche in modo autonomo con l'ambiente, per soddisfare eventuali esigenze. L'utente sarà facilitato nell'interazione con il sistema attraverso un'interfaccia grafica, potendo immettere comandi da eseguire.

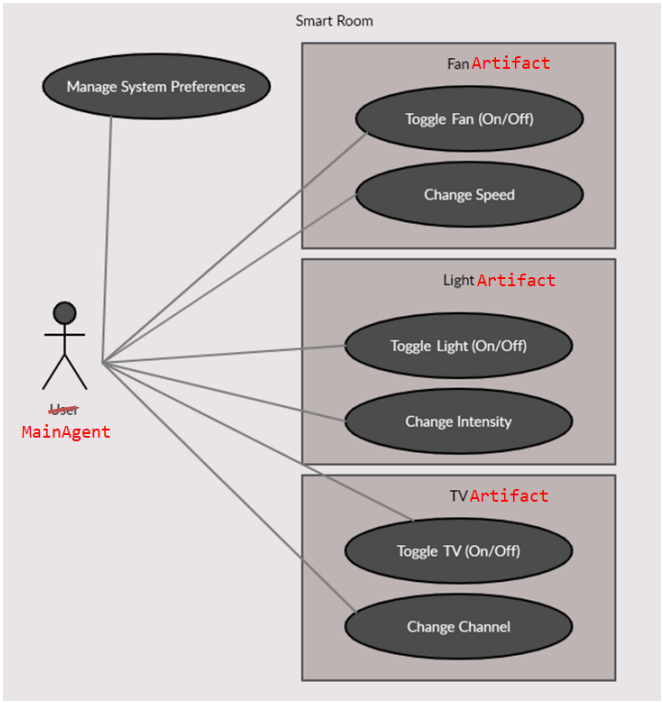

Figura 5.3: Rappresentazione di una Smart Room e suoi oggetti rivisitata secondo il modello Agent-Based di JaCaMo.

Come si può notare dalla figura 5.3, la progettazione di un sistema ad Agenti è direttamente mappata nel mondo reale. Ciò, come sottolineato in precedenza, porta ad un netto vantaggio sull'astrazione del sistema e sulla sua futura implementazione, in quanto nella visione Agent-Oriented tutto viene completamente incapsulato all'interno delle singole entità. Ovviamente, questa non risulta essere una completa progettazione: lo schema vuole dimostrare soltanto il diretto vantaggio che si ha sopra una progettazione classica ad Oggetti.

L'utilizzo del modello ad Agenti, inoltre, non riguarda soltanto semplici contesti come la Smart Room: vi sono infatti presenti studi che spaziano argomenti riguardanti sistemi di evacuazione [21] o addirittura l'osservazione delle api [18].

5.3 BDI e Smart Things

Come da definizione, il modello BDI si basa sui concetti di ***b**eliefs*, ***d**esires* e ***i**ntentions*. Vi è necessità di analisi per quanto riguarda le seguenti affermazioni e quali sono i vantaggi (e/o svantaggi) di utilizzare questa metodologia per lo sviluppo di un sistema IoT.

Come citato da *"MATCH: MultiAgent-based Tactful CooperationScheme for Heterogeneous IoT Devices"* [24], utilizzare agenti può determinare una semplificazione della gestione per quanto riguarda l'eterogeneità delle cose naturalmente presenti in un ambiente, dati diversi contesti nei quali un sistema ad Agenti può essere applicato. Bisogna però saper specificare con criteri rigorosi i comportamenti che le cose debbano avere, in modo da garantire sia la sicurezza delle persone che il risultato atteso, portando ad una collaborazione (e non competizione) tra le cose che si hanno a disposizione.

5.3.1 Ordine ed esecuzione delle azioni

In primo luogo, la capacità degli Agenti ad essere predisposti a interagire tra di loro, anche autonomamente, risulta essere un netto (teorico) vantaggio per quanto riguarda l'implementazione di un sistema Smart: il modello a scambio di messaggi abbraccia completamente la natura asincrona negli eventi che possono capitare all'interno di un ambiente, ma bisogna essere precisi nello specifico sulle priorità dei messaggi in arrivo. Di default, infatti, non vi è implementato un modo per ordinare i messaggi che un Agente si vede arrivare.

Per questo motivo vi sono presenti strumenti per gestire la priorità delle azioni: è necessario però porre attenzione a non disporre i controlli all'interno dell'Agente stesso, ma nella parte relativa alla ricezione e gestione messaggi. L'agente, infatti, deve poter eseguire le proprie intenzioni anche in assenza di alcune conoscenze, se questo vi è permesso: la belief (iniziale e imparata) è esattamente il motivo per il quale ciò è possibile, se ovviamente l'entità è capace di valutare con accuratezza la situazione dell'ambiente in cui si trova.

Non bisogna però intendere questo come una carta bianca per gli Agenti ad eseguire azioni che ritengono giuste: è compito del programmatore a rendere le singole entità sicure, in modo da poter interrompere operazioni, anche critiche, nel caso di uno o più fallimenti. Gli Agenti sono infatti liberi di crearne delle nuovi (o nel caso di JaCaMo, di creare nuovi Artefatti) e di eliminarli a dovere: quest'ultima viene vista come un'operazione importante, in quanto l'eliminazione di una entità provoca automaticamente la distruzione di tutte quelle sottostanti nella sua gerarchia.

L'autoregolazione del sistema, quindi, è frutto di conoscenza degli Agenti ed esperienza del programmatore a renderli autonomi e capaci di imparare. Grazie all'utilizzo di tutti gli strumenti dati a disposizione, gli ambienti possono così diventare più propensi ad essere effettivamente "Smart", non perchè

un'azienda ha deciso di etichettarlo così, ma per definizione di quello che quel determinato sistema può fare e offrire nel suo insieme. Dall'altra parte, è da considerare una visione distopica, dove troppe cose eseguite da macchine rendono uomini meno capaci a svolgere anche banali compiti, diventando quindi completamente dipendenti dalla tecnologia.

5.3.2 Agente come entità centrale

La *belief* di un Agente è quello che egli sa sull'ambiente in cui si trova, sia che si tratti di ulteriori entità, sia che dell'ambiente stesso. Nel caso di una Stanza Smart, questo può essere visto come un vantaggio in termini di completo monitoraggio del sistema implementato; dall'altra parte però utilizzando un'unica figura centrale come Agente, che si interfaccia poi con tutte le Things implementate come Artefatti, potrebbe portare ad un Single Point of Failure, dove se l'entità centrale smette di funzionare, anche tutte quelle ad essa sottostanti non rispondono più ai comandi.

Si potrebbe quindi ragionare piuttosto ad un sistema che, dati gli oggetti Smart, conferisca all'utente un'unica interfaccia nella quale sono disponibili tutti gli oggetti, ma anche una possibilità di controllare direttamente le Smart Thigs, senza l'ausilio di un ente centrale per il quale si debba per forza passare. Il problema di questa soluzione è la difficoltà nell'implementazione: infatti, ogni singolo oggetto non è più solo un Artefatto osservabile, ma di fatto un Agente Intelligente. Un ulteriore problema di questa soluzione risulta essere proprio il trattamento delle Smart Things come entità autonome: non è detto che sia necessario avere una lampadina vista come un'entità così Smart da decidere in autonomia che piano intraprendere in base allo stato dell'environment in cui si trova, anche perchè questo vorrebbe dire che ogni Thing debba essere connessa all'altra, per scoprire perfettamente lo stato del sistema. Tenere un'unità centrale vanificherebbe il tentativo di decentralizzazione, anche perchè si ritornerebbe nel precedente caso nel quale tutte le

informazioni debbano passare da una singola struttura centrale.

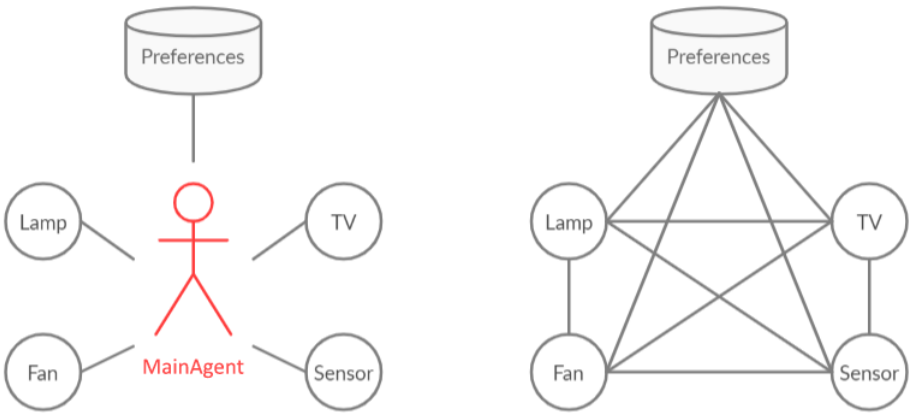

Figura 5.4: Differenza tra un sistema centralizzato e decentralizzato. Si può vedere al colpo d'occhio la semplicità e la criticità del primo, e la complessità del secondo, oltretutto, al crescere dei dispositivi connessi.

Da una parte quindi si ha il problema della molteplicità di comunicazione, che cresce esponenzialmente con l'aggiunta di nuovi dispositivi, che però rispecchia pienamente la decentralizzazione di un sistema ad Agenti; dall'altra un *SPOF* che semplifica nettamente la gestione dell'intero ambiente e permette di concentrare le operazioni importanti all'interno di una singola entità, favorendo l'eventuale collaborazione e il discovery delle nuove funzionalità, non incrementando nello stesso momento le capacità di azione individuale delle Smart Things. Dipendentemente quindi da cosa si vuole ottenere dal sistema, è opportuno scegliere una delle due strade. Essendo però la seconda, quella centralizzata, più facile da implementare e scalabile (in un sistema da 10.000 oggetti è impensabile infatti rendere ogni oggetto responsabile di osservare l'interno ambiente e capire l'azione da intraprendere, mentre con l'entità centrale è lei a doversi preoccupare di tutto), per quanto riguarda la scelta per le successive fasi di analisi, si preferisce pensare al sistema Agent-Based con un Agente centrale che "comanda" gli Artefatti.

5.3.3 Things e Agenti

Non solo la *belief*, ma anche *desires* e *intentions* riguardano da vicino l'analisi che dev'essere affrontata. Mentre è logico infatti pensare all'agente centrale che si pone come desiderio e/o intenzione quella di voler illuminare la stanza, è più difficile calarsi a più basso livello supponendo che una lampadina abbia il desiderio e/o intenzione di accendersi o cambiare la sua intensità.

Vi è però necessità di differenziare quello che idealmente un sistema ad Agenti potrebbe essere e quello che si sta cercando di realizzare. Nel contesto attuale si sta pensando ad un modo per interfacciare le Smart Things esistenti sul mercato in modo da poterle inserire dentro una stanza e utilizzare per questo scopo Agenti BDI. La Thing, di per sè, è stata già progettata, realizzata; ha già feature interne che le permettono di cambiare il loro stato interno: queste però **non** sono state progettate in vista di un'applicazione in un sistema Agent-Based, ma piuttosto in un ambiente Smart classico tramite connessione ad un Hub WiFi (o altri, ad esempio ZigBee). Ponendo caso che si voglia realizzare una Smart Thing da 0, utilizzando sin da subito il paradigma Agent-Based, viene automatico pensare alla lampadina come se fosse effettivamente un agente: ha delle intenzioni, dei desideri e delle credenze su quello che è il sistema "Lampadina". Per essere Smart ha sicuramente bisogno non solo delle parti elettriche che permettono di emanare luce, ma anche quelle di ricevere comunicazioni dall'esterno, di managing energetico interno e simili. Dunque, è logico in questo caso contraddire l'introduzione di questa sezione: infatti, se guardata internamente e realizzata dall'inizio con questa ideologia, la lampadina **può** essere sviluppata seguendo questo paradigma, avendo quindi tutte le cose che caratterizzano un Agente. Di conseguenza, anche l'intero sistema dovrebbe essere rivalutato in modo da inglobare non solo Artefatti che si interfacciano con le Things già esistenti, ma anche l'unità centrale che deve riconoscere ulteriori Agenti come Smart Things.

Come però sottolineato, questo è solo un caso ipotetico. Attualmente non

vi sono particolari sviluppi nell'ambito Smart Things che prevedono l'utilizzo della Programmazione ad Agenti intensiva, e la maggior parte delle Things esistenti non è stata pensata per farne uso. Per questo motivo se si volesse realizzare un sistema completamente ad Agenti, vi è necessità di progettare e realizzare anche Smart Things seguendo lo stesso modello; nel caso di studio, invece, si tratterà di un approccio più limitato, che si baserà sull'adozione di Smart Things correntemente esistenti e funzionanti, senza dover inventare nuovi metodo per la creazione delle entità Smart.

Inoltre, come descritto in *"Building Cyber-Physical Systems - A Smart Building Use Case"* [20], anche una semplice casa possiede diverse sfide, sia per quanto riguardo l'eterogeneità delle cose che la complessità delle azioni da compiere. Vi sono infatti diversi strati di interoperabilità che possono esistere. Dall'altra parte, seppur non citando direttamente, l'articolo fa anche riferimento all'uso di ontologie (o comunque tecnologie standard) per quando riguarda il layer della descrizione degli oggetti e del loro stato, ricadendo perfettamente nell'ambito di studio qui presentato, trattato poi successivamente nel capitolo 5.4.1.

5.3.4 Blackboxes

Nella precedente sezione è emerso un dettaglio molto importante riguardante la natura non-Agent-friendly delle entità che si vorrebbe in qualche modo sfruttare all'interno del sistema. Un Agente, infatti, è autorizzato a comunicare soltanto grazie allo scambio di messaggi, non tramite il richiamo di alcuni metodi, ne tramite l'uso di tecnologie com RPC o altre, come WebSocket.

In JaCaMo questo problema è stato superato grazie all'adozione degli Artefatti: attraverso di essi, infatti, è possibile interagire con del codice legacy (o delle classi/librerie o addirittura programmi interi), rendendo le cose completamente trasparenti dal lato dell'Agente che osserva quell'Artefatto.

Ovviamente, si tratta di un processo di "blackboxing" di quella che è la vera natura dell'oggetto interessato, ma senza di esso la Agent Oriented Programming non potrebbe dichiararsi priva di codice "sporco" di qualche richiamo della classica OOP.

L'artefatto dunque può nascondere l'entità di una qualsiasi parte già implementata: una GUI in JavaFX, una Smart Things con un display e una stampante appaiano tutte come un Artefatto, il quale con le sue azioni interne permette all'Agente di mandargli i messaggi. L'agente risulta essere quindi compliant allo standard di programmazione, e l'artefatto gestisce la richiesta interamente per essere compliant con la tecnologia che viene sfruttata sotto.

Questo problema può essere visto in duplice modo: da una parte vi è una complicazione per i programmatori per rendere le cose più separate e non immediatamente disponibili, dall'altra, come nella programmazione ad Oggetti, è una buona pratica non far fare troppo ad una sola entità; infatti, al crescere di eventuali features è probabile che si assista più ad un caos che ad una semplificazione dei problemi, per cui lo svantaggio descritto risulta essere in realtà più un vantaggio, che permette di mantenere il principio di separazione degli obiettivi, rispettando inoltre gli standard della programmazione ad Agenti.

5.3.5 Sicurezza

Come citato in "*Intelligent Multi-Agent Collaboration Model for Smart Home IoT Security* [6]", gli agenti possono essere utilizzati anche per incrementare la sicurezza dei sistemi IoT. Utilizzando infatti una strutturazione a base di molteplici agenti che creano un layer di trust in base al database sottostante, si può ottenere una organizzazione interna tale da permettere un maggiore controllo su chi esegue i comandi (soprattutto da remoto).

Per motivi di semplicità la soluzione proposta non verrà adottata nel risultato finale, tuttavia si sottolinea l'importanza e l'esistenza di tale studio.

5.4 Disponibilità della conoscenza

In una visione completa di un sistema, nel quale si può immaginare un numero potenzialmente infinito di oggetti Smart, è naturale aver a che fare con una molteplicità di informazioni e operazioni da fare che necessariamente ha bisogno di essere in qualche modo regolamentata e strutturata, in modo da evitare il caos che ne deriverebbe. Le problematiche, infatti, non riguardano soltanto la quasi impossibilità di manutenzione del sistema, dovuta alla scarsa interoperabilità dei sistemi, ma anche la semplice osservazione e raccolta dati dell'ambiente in cui questo fantomatico sistema potrebbe funzionare. In poche parole, l'applicazione caotica e senza regole di un sistema eterogeneo porta inevitabilmente alla lenta distruzione dello stesso, vanificando completamente le soluzioni ai problemi che si erano posti di risolvere.

5.4.1 Thing Description

Da diverso tempo si sta discutendo su come rendere il mondo IoT più versatile, compatibile, universale e aperto. Le attuali tecnologia risultano essere perlopiù proprietarie e non standardizzate, per cui si è spesso costretti a rimanere legati con uno (o pochi) competitor presenti sul mercato per problemi di compatibilità dei sistemi. Infatti, ogni oggetto Smart attualmente presente sul mercato, ha bisogno di un suo centro di comando, che differisce da produttore a produttore: nel caso di oggeti Smart dotati di protocolli WiFi generalmente basta un'applicazione che riesca a sfruttare il protocollo REST; ma utilizzato protocolli come Bluetooth e/o ZigBee ogni manufacturer implementa un suo protocollo di comunicazione che non è noto a priori.

Per far fronte alle diversità e per regolare il caos che si sta cominciando a creare, anche in vista dell'imminente espansione di oggetti Smart, grazie

anche all'introduzione delle nuove infrastrutture, soprattutto quelle di comunicazione, l'ente W3C [28] ha instaurato il Web Of Things [30] e la Thing Description [27], argomenti già trattati nel capitolo 2.2.

L'uso della Thing Description facilita la discovery e l'interoperabilità tra gli oggetti presenti nell'ambiente. Avendo al suo interno attributi, azioni, proprietà osservabili dell'oggetto e l'insieme delle cose human e non-human readable, permette di avere un quadro completo di quello che l'oggetto può essere e di quello che può svolgere.

Calandosi nel mondo degli Agenti, è facile pensare ad una creazione dinamica delle Things per inserirle all'interno del sistema. È necessario quindi definire alcuni passi fondamentali per eseguire completamente il processo di creazione e inserimento di una nuova Thing all'interno di un ambiente:

1. **Utente aziona il comando *Add New Thing***: tramite un'interfaccia grafica presente in un'App o in una Smart Thing l'utente dichiara la volontà al sistema di voler aggiungere una nuova Smart Thing.

2. **Agente cattura il comando**: l'entità centrale che si occupa della gestione capisce che un nuovo oggetto sta per essere inserito. Chiede all'utente gli unici due dettagli che vuole sapere: il nome della cosa e la sua Thing Description.

3. **Agente crea l'artefatto con la TD**: una volta ottenuto i dati, l'agente chiede la creazione di un nuovo artefatto con il nome indicato dall'utente, che rispecchia l'oggetto reale grazie alla Thing Description.

4. **L'Artefatto si dichiara pronto**: esso comunica all'Agente di poter essere testato da parte dell'utente prima di dichiararsi completamente funzionante.

5. **L'Agente vede l'artefatto**: dopo aver ricevuto la sua richiesta, l'agente comunica all'utente che sarebbe meglio se guardasse la Smart Thing appena aggiunta e che ne verificasse le funzionalità.

6. **Verifica delle funzionalità**: l'utente è libero di testare, approvare e/o modificare il comportamento della Thing appena creata.

Questi passaggi risultano essere una proposta di algoritmo che potrebbe servire all'istanziamento di una nuova Thing nell'ambito di una Smart Room Agent-Based, supponendo di voler realizzare il sistema con l'Agente centrale che comanda la Smart Room. Mentre quasi tutti i punti sono realizzabili utilizzando metodologie date allo sviluppatore, sia attraverso le API di JaCaMo, sia attraverso l'uso di librerie Java, vi si possono trovare alcuni punti critici di questo procedimento:

- **Limiti di JaCaMo a Runtime**: anche se esistono (o comunque possono essere implementati) metodi per il parsing della Thing Description, attualmente l'ambiente di JaCaMo non offre nessuna possibilità di aggiungere proprietà e azioni eseguibili a runtime, vanificando completamente l'obiettivo di creare dinamicamente delle Things.

- **Conoscenza delle funzionalità**: non solo l'Artefatto, ma anche l'Agente che lo osserva ha bisogno di sapere quali sono le funzionalità derivate dalla Thing Description letta. Il problema si può risolvere in due modi:

  - L'Agente si occupa di leggere la TD, in modo da capirne le funzionalità. L'Artefatto quindi viene creato dopo che la descrizione dell'oggetto venga letta. In questo modo si fa svolgere tutto il lavoro all'Agente, rompendo il principio della Single Responsibility.
  - L'Artefatto, una volta pronto dopo aver letto la TD, dichiara all'Agente quali sono i metodi che l'Agente può chiamare su di lui. Questo punto però interferisce con i limiti che esistono al momento con JaCaMo sull'esecuzione a Runtime.

- **Necessità dell'intervento umano**: l'approvazione da parte dell'utente potrebbe essere macchinosa, soprattutto se si dovessero aggiungere centinaia di Things. Si potrebbe pensare ad un meccanismo di automazione del test della Thing, che capisce in qualche modo di essere configurata bene (ad esempio, configurando una lampadina si potrebbe pensare di creare, a partire dalle proprietà e azioni comprese, un test automatico che verifica tutte le funzionalità apprese, come accendi/-spegni/regola intensità e simili). Questo processo può essere ritenuto non critico in un ambiente casalingo, mentre nell'ambito industriale è di estrema importanza garantire la sicurezza dei dipendenti e delle macchine.

- **Necessità della configurazione manuale**: l'intero processo potrebbe essere semplificato, se le Smart Things avessero modo di comunicare automaticamente i loro nomi e il percorso per la Thing Description. In alternativa, ogni Smart Thing dovrebbe avere un percorso standard dove trovare il proprio nome e la TD, in modo che, una volta accese, possano essere automaticamente rilevate dall'Agente centrale e configurate senza che l'utente debba per forza dichiarare di voler aggiungere una nuova Thing.

- **Mancanza di standard evoluti/in uso/noti**: anche se si disponesse di tutte le tecnologie che permettono la realizzazione di tutti i punti precedenti, un'ultima nota dolente, per quanto riguarda la situazione attuale, è la completa mancanza delle Thing Description per le Smart Things correntemente esistenti. Essendo uno standard nuovo e non particolarmente usato (anche perchè ogni brand vuole tenersi l'utente stretto alle proprie soluzioni), vi è necessità di creazione manuale delle TD per gli oggetti che si vogliono creare. Il vantaggi ottenuti sono principalmente due: non solo si comincia ad utilizzare uno standard approvato, ma ulteriormente, si può pensare alla realizzazione di un database accessibile a tutti, nel quale vengono raccolte e pubblicate le

TD realizzate da parte di tutti gli utenti da tutto il mondo, facilitando, con il passare del tempo, la configurazione all'interno dei sistemi che sfruttano la Thing Description (non per forza soltanto quelli ad Agenti).

Nel frattempo, JaCaMo si sta evolvendo e metodologie che permettono tali operazioni sono in via di arrivo. Per motivi didattici e di tempo, nell'eventualità di realizzazione un proof of concept, si preferirà predisporre di tutte le cose attualmente implementabili, lasciando spazio alle successive aggiunte e/o modifiche post-aggiornamento, simulando, per il momento, alcuni dei comportamenti non ancora implementati e/o che richiederebbero la realizzazione di un progetto complesso a parte, non rientrante nei temi che si vorrebbero affrontare con questo studio.

5.4.2 Thing Status

Mentre è possibile definire interamente le proprietà e azioni di una Thing attraverso la TD, sarebbe utile se si potesse, allo stesso modo, descriverne lo stato attuale. La Thing Description, infatti, copre soltanto il lato "statico" della Thing, specificando unicamente quelli che possono essere aspetti osservabili ed eseguibili, ma non quelli attualmente attivi.

Non esistono attualmente standard che permettono di definire lo stato attuale di una Thing. Si può immaginare però, ad esempio, che tutte le proprietà osservabili di un Artefatto, che una Thing di conseguenza possiede, possano essere in qualche modo tradotte in un formato standard, ad esempio in RDF. Utilizzando il modello così definito, ogni entità presente all'interno di un ambiente potrebbe fornire non solo la sua descrizione, intesa come il "*cosa so fare*" ma anche il "*come sto ora*".

La questione, inizialmente, risulta essere banale se osservata da solo un punto di vista: le Things hanno già modo di dire al sistema in che situazione attualmente si trovano e di comunicare i cambiamenti in Real Time grazie

ad uno scambio di messaggi. Servire un ulteriore modo per descrivere ciò di cui si ha già disposizione sembra un controsenso, eppure vista dall'alto può essere uno strumento utile per diversi motivi:

- **Utilizzo standard di dati**: avendo a disposizione l'informazione su tutto il sistema in un formato unico e standard, è possibile sfruttare la conoscenza in possesso per eseguire ulteriori analisi, ottimizzazioni, statistiche e quant'altro. In poche parole, l'utilizzo di una rappresentazione comune a tutti facilita quello che è l'osservazione continua del sistema, dal puro punto di vista dei dati e risultati ottenuti.

- **Snapshot**: un fantomatico sistema in funzione potrebbe decidere di eseguire il monitoraggio completo delle risorse che ha a disposizione, salvando man mano lo storico di quello che era lo stato delle varie entità nel tempo. Questo può essere utile per la fase di debug o quando si verificano i guasti: un operatore può andare indietro nello storico e guardare tutti i parametri di tutte le entità connesse tra di loro e individuare eventuali criticità. Nel sistema ad Agenti, questa operazione può essere ulteriormente semplificata, prendendo automaticamente decisioni in base alle circostanze globali dell'ambiente, avvertendo eventualmente gli operatori delle azioni critiche intraprese.

- **Scoperta nuove funzionalità**: un sistema descritto in modo tale da permettere ad una macchina di fare del reasoning sopra, rende possibile la scoperta di ulteriori funzionalità (o proprietà) del sistema, non inizialmente previste. Sapendo, ad esempio, di avere 4 lampadine connesse in fila, si può dedurre che accendendo e spegnendole in un certo ordine si possono creare sequenze di luci. Nel caso di un entità centrale, ad esempio, il reasoning potrebbe essere fatto ogni tal volta che un nuovo oggetto viene aggiunto al sistema, ragionando su quelle che sono le azioni e proprietà di quell'oggetto.

- **Sicurezza**: analizzando i dati a disposizione, eventualmente derivandone nuovi, si possono individuare pattern comportamentali di individui

che utilizzano quotidianamente oggetti Smart. Grazie alla descrizione dello stato, essa è univoca e osservabile sempre: se un malintenzionato volesse procedere in modo molto contrastante con quelle che sono le solite abitudini degli utilizzatori, il sistema potrebbe provvedere all'avviso o addirittura blocco delle azioni malevole. Inoltre, la verifica sui comportamenti dei singoli oggetti, analizzando il loro stato, porta ad un'incremento dell'affidabilità degli stessi, nonché aggiunge uno strato di protezione contro Smart Things malevole, che senza un monitoraggio attivo del loro stato attuale, potrebbero essere immerse nel sistema e risultare invisibili e/o camuffate; ad esempio, conoscendo un classico comportamento di una lampadina, se essa cominciasse ad eseguire azioni non comuni a questa tipologia di oggetto, il sistema potrebbe avvisare l'utente e/o bloccare un tale azione.

- **Individuazione oggetti**: nel caso in cui la Thing Description non fosse disponibile, oppure nel caso in cui non sia completa, avendo a disposizione una quantità di dati sufficiente, è possibile non solo individuare pattern comportamentali di oggetti, ma anche oggetti stessi, permettendo la loro corretta classificazione. Questo aspetto può portare ad una semplificazione di interrogazioni che potrebbero essere fatte all'interno dell'ambiente, chiedendo, ad esempio, di elencare tutti i dispositivi le cui funzionalità corrispondono ad una determinata interessata caratteristica. Questo aspetto è comunque strettamente legato a tutto quello che è descritto già nel punto soprastante, riguardante la sicurezza.

Dare semantica agli oggetti quindi, non solo nel modello degli Agenti BDI, vuol dire aggiungere potenzialità e nuove possibilità all'environment analizzato: rendere le cose ancora più connesse e parlanti, garantendo, oltretutto, un maggior controllo su quello che succede all'interno del sistema incrementandone la sicurezza; il tutto porterebbe ad una riduzione dei costi di manutenzione, prevenzione dei guasti e/o la limitazione dei danni dovuti

ad essi, nonché garantirebbe una vasta disponibilità di dati che possono essere marcati come "smart", in quanto ben definibili, collegabili e capibili da una macchina.

Purtroppo attualmente in campo non esistono standard che definiscono questo tipo di soluzioni, anche se come descritto in *"Building Cyber-Physical Systems - A Smart BuildingUse Case"* [20], si può però pensare di poter usufruire delle ontologie definite, ad esempio, tramite RDF [13]. Per realizzare un test di quello che viene poi definito, si può pensare a SPARQL [14] per quanto riguarda l'interrogazione della conoscenza creata.

Capitolo 6

Proof of Concept - Design

Nel seguente capitolo verrà descritto il procedimento che si è svolto per la realizzazione di un progetto funzionante, basandosi su quanto descritto nei capitoli precedenti.

6.1 Requisiti

6.1.1 Descrizione ad alto livello

Si vuole realizzare un sistema capace di governare su una Stanza Smart, tenendo in considerazione una possibile visione più ampia per estendere le funzionalità al di fuori del contesto della stanza e poterlo gestire all'interno di un'appartamento, oppure, astraendo dall'environment, funzionante generalmente all'interno di un ambiente nel quale si vogliono controllare automaticamente alcuni parametri. La Smart Room dev'essere capace di governare dei device connessi alla rete, i quali dovranno gestire autonomamente, e in base alle preferenze dell'utente, alcune situazioni.

In particolare, per semplicità, si vuole predisporre la stanza di tre oggetti smart: lampadina, televisore e ventilatore. L'utente è libero di scegliere, in base alle proprie esigenze, quale comportamento far eseguire ad ognuno di loro in base alla situazione attualmente presente nella stanza; ad esempio,

si vuole limitare la luce se la TV è accesa, mantenendo il ventilatore attivo soltanto se la temperatura supera 24°C. Non ci dev'essere un processo di configurazione, facendo funzionare il sistema in modalità "Plug and Play", con il risultato di avere immediatamente disponibili tutte le funzionalità offerte dai singoli device in un applicazione di controllo (installata ad esempio su uno Smartphone). In caso di più utenti con preferenze diverse, vi è necessità di stabilire priorità su quale comportamento il sistema debba prendere.

6.1.2 Business Requirements

1. Realizzare un progetto nell'ambito di Pervasive Computing e Web Semantico con l'obiettivo di un sistma Agent-Oriented sfruttante gli standard definiti dall'ente W3C.

2. Mettere alla prova le conoscenze acquisite durante i corsi di Pervasive Computing e Web Semantico relative alle possibilità di realizzare sistemi in ambiti realmente esistenti.

3. Riuscire a mettere in atto un sistema utilizzando le più innovative tecnologie (anche emergenti).

4. Mettere in atto un progetto che incroci conoscenza di due materie differenti all'interno di un'applicazione concreta.

6.1.3 User Requirements

1. Possibilità di funzionamento Plug&Play.

    1.1. Le Smart Things devono essere autoconfiguranti oppure richiedere all'utente l'immissione del solo nome e della Thing Description associata.

    1.2. È possibile offrire all'utente un setup iniziale al primo avvio del sistema.

1.3. L'utente deve poter aggiungere un numero virtualmente infinito di oggetti.

2. L'utente interagisce con gli oggetti tramite un unico terminale.

   2.1. Il terminale potrà essere virtuale (interfaccia grafica su un device come PC o Smartphone) o reale (una Thing).

   2.2. Nuove azioni e proprietà, nonchè nuovi oggetti, devono essere immediatamente visibili e configurabili nel terminale di gestione.

   2.3. L'utente può a piacere decidere cosa e come funzioni, stabilendo per ogni oggetto le sue proprietà e preferenze.

   2.4. Il terminale dev'essere l'unica interfaccia abilitata a gestire gli item nell'ambiente, senza ausilio di nessun'altro programma (proprietario o open).

6.1.4 Functional Requirements

1. Ogni device deve poter essere rilevato dal sistema e funzionare senza interferire nel suo funzionamento.

   1.1. Il device deve fornire una Thing Description.

      1.1.1. L'accesso a quest'ultima dev'essere facile e libero, integrato all'interno della Thing.

      1.1.2. Il nome, le proprietà e le azioni che la Thing può svolgere devono essere tutte descritte all'interno di questa.

      1.1.3. L'analisi della TD deve fornire un quadro completo sull'oggetto, in modo da far capire al sistema automaticamente tutti i parametri necessari al suo utilizzo.

      1.1.4. Se non disponibili, le Thing Description dovranno essere create.

   1.2. Il device deve seguire le linee guida definite dagli standard W3C.

   1.3. Il device deve fornire una Thing Status Description.

2. L'interazione tra le cose deve avvenire in modo seamless.

    2.1. L'utente non deve decidere ogni volta il comportamento delle Smart Things.

    2.2. In base alle preferenze, le Things devono garantire all'utente il massimo comfort che è possibile ottenere in base agli oggetti attualmente connessi.

    2.3. L'utente dev'essere comunque abilitato a poter disattivare alcuni comportamenti automatici e ad avere, in caso di necessità, il pieno controllo della situazione.

    2.4. Se è possibile, le Things devono essere capaci di definire nuovi comportamenti in base alla combinazione di diversi oggetti smart e/o riuscire ad ottenere approssimazioni delle volontà dell'utente sfruttando le funzionalità presenti nel sistema, imparando nuovi comportamenti.

6.1.5 Non Functional Requirements

1. **Rispetto degli standard W3C.** Gli oggetti smart, come il sistema nella sua integrità, dev'essere conforme agli standard definiti dall'ente.

2. **Non reinventarsi la ruota.** Se esistono metodi già definiti in letteratura per la risoluzione di un problema, è necessario utilizzare con tutti i pro e contro questi metodi, per non dover inventare un nuovo standard emergente che nessuno adotterà.

3. **Praticità.** Il sistema dev'essere implementabile con facilità in una casa ed usabile da utenti anche con minori conoscenze tecnologiche.

6.1.6 Implementation Requirements

1. **Utilizzo di librerie standard**. La totalità del sistema deve essere sviluppata sfruttando le possibilità che al giorno di oggi vengono offerte

da diverse aziende. La parte del codice custom deve riguardare solo l'interazione tra di essere e parti che non sono in alcun modo trattate da soluzioni esistenti.

2. **Utilizzo di metodologie innovative**. Il progetto non dev'essere risultare una banale centralina di comando per oggetti Smart, ma un innovativo modo per poter gestire un circoscritto ambiente di IoT, avendo sempre un'ampia visione sulle sue possibili estensioni fuori dall'ambito di attuale studio.

3. **Utilizzo di un sistema di versioning**. Per facilitare lo sviluppo sarà necessario utilizzare un sistema che permette un'agevole sviluppo, considerando come possibilità Git e l'uso della piattaforma GitHub.

4. **Budget**. La realizzazione del sistema deve avvenire senza comportare dispendi economici.

6.2 Metodologia di sviluppo

Il lavoro svolto durante il processo di sviluppo verrà organizzato adottando la meodologia *a fontana*. Nell'accezione di questo framework è previsto uno sviluppo tendente al classico *Waterfall*, con la possibilità di tornare indietro nei vari step che compongono l'intero processo. Questo approccio risulta essere quindi il più adeguato per il lavoro svolto in singolo, offrendo al soggetto più flessibilità: infatti, nonostante la lettura di questa relazione possa sembrare più "a cascata", vi sono state effettuate diverse interazioni tra le fasi di stesura, sia per confronto che per raggiungere un buon livello di esposizione.

6.3 Architettura

Generalmente, quando si tratta di progettare il pattern da utilizzare per la programmazione, si cerca di scegliere uno dei più noti (ad esempio, tra

MVC o MVVM). Nel caso degli agenti questo non è possibile, in quanto essi utilizzano delle modalità di realizzazione e/o comunicazione diversamente strutturate. Vi è necessità dunque di andare più a fondo, descrivendo precisamente di come avviene l'organizzazione in questo particolare modo di programmazione. Non avendo quindi modo di definire, ad esempio, la suddivisione Model-View-Controller, è necessario definire in altri modi la strutturazione che si vorrà intraprendere durante lo sviluppo. Grazie all'adozione del modello ad Agenti, è possibile individuare elementi chiave di un MAS (Multi Agent System), guardandolo immediatamente dal punto di vista di JaCaMo:

- **Agente**: entità intelligente, autonoma, predisposta alla comunicazione. Osserva altri agenti e Artefatti. Interagisce, anche autonomamente, da sola e/o con altre entità per raggiungere i propri obiettivi. Può assumere complessità diverse e può essere reattiva e/o pro-attiva.

- **Artefatto**: entità secondaria, a supporto dell'agente, che funziona sia da tramite nel mapping mondo reale-mondo virtuale, sia da aiuto e/o blackbox per quanto riguarda il sistema multiagente, nel quale esso può costituire un punto di incontro tra codice legacy ed Agenti.

- **Interazione**: è la chiave del sistema che permette agli Agenti di comunicare tra di loro e con l'ambiente.

- **Ambiente**: virtuale o reale, è costituito dall'insieme di Agenti e Artefatti che lo condividono.

- **Organizzazione**: stabilisce una gerarchia e l'ordine all'interno di un sistema ad Agenti.

L'ambiente è quello della Smart Room, che inizialmente sarà virtuale e simulato; per quanto riguarda l'organizzazione, invece, Moise permette un'organizzazione molto dettagliata degli Agenti che vivono all'interno di un sistema. Essendo il caso di studio limitato solo alle eventuali potenzialità degli

Agenti nell'ambito di una Smart Room, piuttosto che alla loro organizzazione, si preferisce non lasciare troppo peso ad una ben definita organizzazione, concentrandosi sugli aspetti di interoperabilità tramite le Thing Description e Thing Status, tenendo conto anche dei requisiti posti.

6.3.1 Agenti

Essendo il modello ad Agenti un modello emergente, vi sono diverse possibilità di organizzare il progetto. La più completa risulta essere quella di **JaCaMo**, il quale oltre alle entità descritte nella sezione precedente, offre la possibilità di definire **Artefatti**. Le caratteristiche però non riguardano solo l'aggiunta di questa ulteriore entità, ma anche nella struttura della soluzione. Essendo quindi un modello molto completo per quanto riguarda l'intera organizzazione, si preferisce puntare su di esso per le successive fasi.

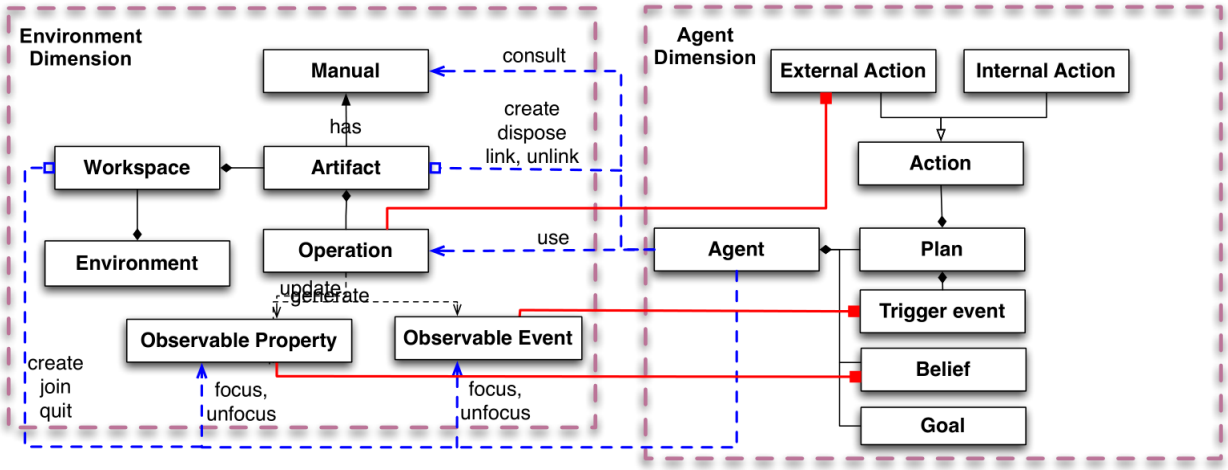

Figura 6.1: Modello di interazione tra Agenti e Environment.

Avendo JaCaMo il supporto su Java per quanto riguarda la definizione di Artefatti e estensioni per Agenti, per le classi definite in questi ambiti si adotterà il principio della separazione dei componenti, dividendo il *model*

e la *view*, ove possibile. È però necessario ricordare che la strutturazione ad Agenti ha una propria visione, in quanto essi possiedono la "conoscenza" *(belief)*: l'insieme delle cose che sanno e che apprendono. Un artefatto, inoltre, può esporre delle proprietà osservabili all'esterno, rendendo però la loro definizione necessaria all'interno dello scope dell'artefatto stesso.

6.3.2 Linguaggi

Principali

Come definito in precedenza, JaCaMo può far uso di Java per la definizione di alcuni suoi elementi. Il linguaggio scelto permette poi la definizione di ulteriori cose, seguendo un classico stile di programmazione.

Java possiede molteplici vantaggi, dove tra quelli principali si possono trovare un vasto supporto della community, un'ampia scelta di librerie e la facilità con la quale le applicazioni al giorno di oggi vengono gestite. Diversi aspetti di questo linguaggio però portano a serie considerazioni sul perchè sia ancora uno dei principali utilizzati nell'ambito dello sviluppo. In primo luogo, è estremamente laborioso gestire i tipi opzionali, vi è solo parziale supporto alla null-safety e ogni classe scritta introduce del non banale boilerplate.

Per poter soddisfare il requisito per l'uso di tecnologie innovative, Java non pare una soluzione adeguata; dall'altra parte, utilizzando JaCaMo non vi è modo di utilizzare un linguaggio diverso. Per questo motivo si decide di puntare su uno dei linguaggi Java-based tra Scala e Kotlin.

Scala risulta essere un linguaggio che punta più sulla programmazione funzionale. Ridefinisce in modo sostanziale quelle che sono le classi e interfacce di Java, ponendole più secondo il suo stile. Per i programmatori non esperti, il linguaggio può sin dall'inizio sembrare più difficile, ma a lungo andare offre funzionalità molto potenti che permettono di abbreviare e sem-

plificare il codice Java. Kotlin risulta essere più una via di mezzo, che non punta ad essere puramente funzionale; esso conserva di più lo stile java-like di classi e interfacce, introducendo soltanto qualche novità. Offre però una gestione di quest'ultime decisamente migliore, nonchè aggiungendo anche la null-safe e l'uso innato di tipi opzionali.

Entrambi i linguaggi rimangono compatibili con Java ed entrambi offrono funzionalità non presenti nativamente: essendo però Kotlin più vicino allo stile di Java, si preferisce di non puntare il tutto sul funzionale e rimanere più al livello base, per non incorrere a problemi implementativi, anche a causa della scarsa esperienza con i linguaggi funzionali. Rimanendo quindi JaCaMo perfettamente funzionante, il riferimento per la programmazione "classica" sarà Kotlin.

Secondari

JaCaMo, oltre a Java, utilizza internamente due linguaggi che permettono:

- la definizione degli agenti - **AgentSpeak** (nella versione estesa), interpretata da Jason;
- la configurazione dell'ambiente di JaCaMo nel file *.jcm* - **markup simil-JSON proprietario**;
- eventualmente, la definizione dell'organizzazione tramite Moise - **XML**.

Oltre all'ambiente di JaCaMo, vi è necessità di utilizzare un linguaggio per la definizione della Thing Description: come definito dallo standard W3C [27], viene utilizzato JSON nella versione *JSON-LD 1.1* [22], mentre per la definizione delle Thing Status si utilizzerà il formato definito dallo standard RDF.

Ulteriori linguaggi verranno presi in considerazione soltanto definendo quali librerie si vorranno utilizzare.

6.3.3 Sistema Operativo & PC

Il progetto verrà svolto interamente su una macchina con Windows 10, configurata con le seguenti caratteristiche:

Component	Name
Motherboard	MSI Z270 M7
CPU	Intel i7-7700k
GPU	Gigabyte GTX 1080Ti
RAM	Ballistix DDR4 @ 2400MHz
Disk	SSD Samsung Evo Pro 750 1TB

Tabella 6.1: Configurazione del PC sul quale verrà realizzato il progetto.

6.3.4 Librerie & Tool

Oltre alla definizione di linguaggi e architettura generale, si vogliono definire in principio alcune delle librerie e tool che verranno utilizzate nel progetto.

Librerie

- **TornadoFX** [15] - facilita la creazione delle interfacce utente, utilizzando una sintassi pesantemente basata sull'uso di Kotlin. Facilita la creazione dei layout e l'applicazione dello stile, che a differenza di JavaFX (su cui è basato) non è scritto in *CSS* custom, ma sempre tramite codice Kotlin. Per il suo utilizzo è comunque necessario includere dipendenze di JavaFX.

Tool

- **Gradle** [5] - facilita la composizione, creazione e la gestione delle dipendenze. È praticamente obbligatorio in qualsiasi progetto con un minimo di complessità non solo per praticità, ma per necessità. Permette la creazione di script per il corretto build delle applicazioni.

- **Eclipse** [3] - IDE per la programmazione, che supporta il plugin per lo sviluppo in JaCaMo [2].
- **IntelliJ** [7] - IDE per la programmazione, funzionante meglio con il linguaggio Kotlin [8].
- **Visual Studio Code** [16] - IDE più leggero e pratico, nel caso di piccole modifiche al volo di parti del codice e/o apertura dei file.
- **Git**, **GitHub**, **GitHub Desktop** [4] - sistema di versioning, il sito per la gestione del repository e il programma in versione desktop.

6.3.5 Casi d'uso

L'architettura deve permettere all'utente di interfacciarsi con tutte le Smart Things tramite un unico terminale. Previa una configurazione, esso deve automaticamente regolarsi in base alle impostazioni dell'utente, che è comunque abilitato a interagire con le singole cose manualmente, sovrascrivendo le eventuali modifiche effettuate dal manager. Seguendo i requisiti, le tre Smart Things e i relativi casi d'uso che si vogliono coprire sono rappresentati nella seguente figura.

6.4 Design di Dettaglio

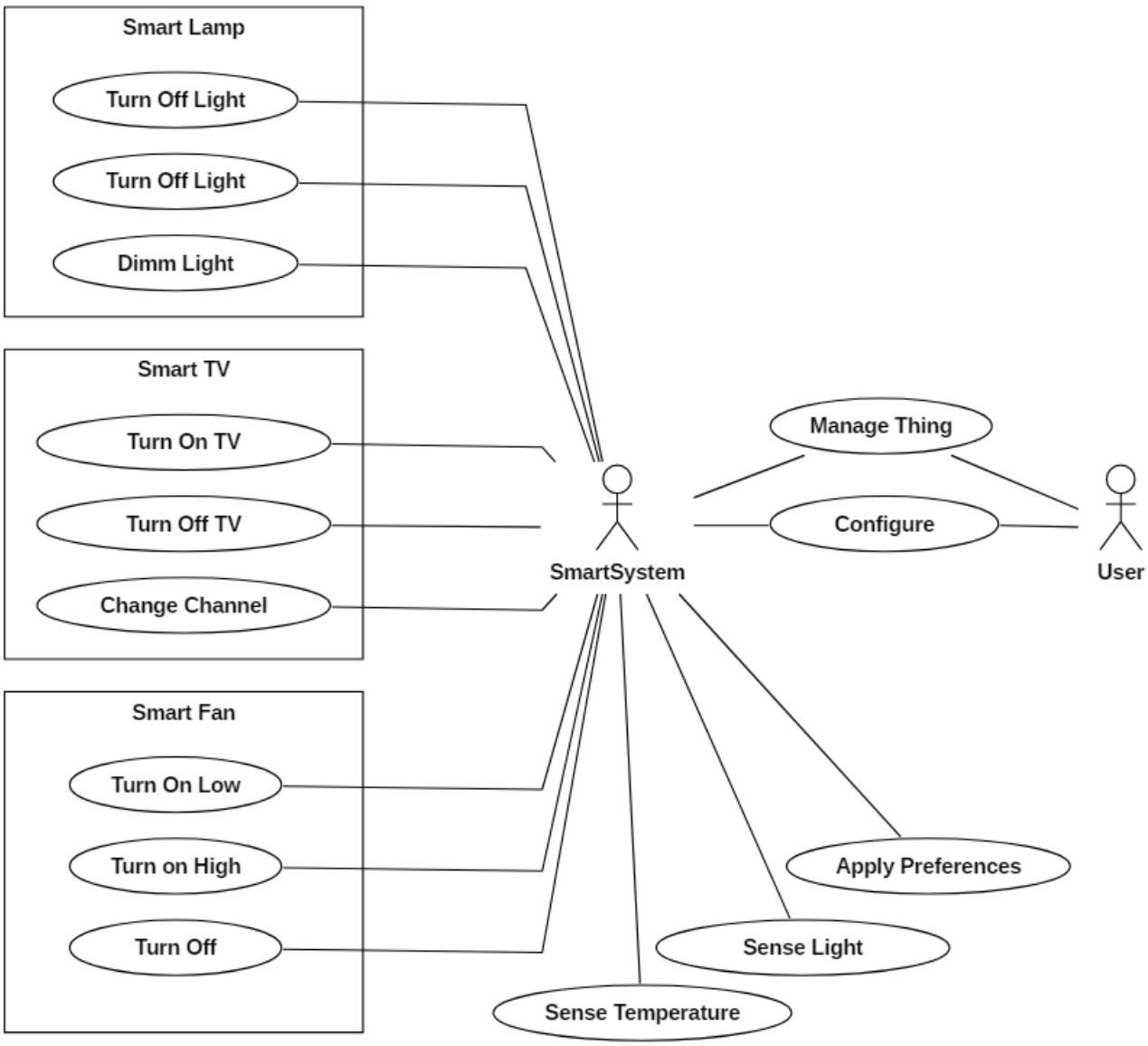

Figura 6.2: Casi d'uso del sistema in progettazione.

Capitolo 7

Proof of Concept - Realizzazione

Capitolo 8

Retrospettiva e commenti

Bibliografia

[1] *Cartago.* http://cartago.sourceforge.net/.

[2] *Configuring eclipse for jacamo.* http://jacamo.sourceforge.net/eclipseplugin/tutorial/.

[3] *Eclipse.* https://www.eclipse.org/downloads/.

[4] *Git, github e github desktop.* https://git-scm.com/, https://github.com/, https://desktop.github.com/.

[5] *Gradle.* https://gradle.org/.

[6] *Intelligent multi-agent collaboration model for smart home iot security.* https://ieeexplore.ieee.org/document/8473441.

[7] *Intellij.* https://www.jetbrains.com/idea/.

[8] *Intellij with kotlin.* https://kotlinlang.org/docs/tutorials/jvm-get-started.html.

[9] *Jacamo.* http://jacamo.sourceforge.net/.

[10] *Linked data.* https://www.w3.org/wiki/LinkedData.

[11] *Moise.* http://moise.sourceforge.net/.

[12] *Ontology.* https://www.w3.org/standards/semanticweb/ontology.

[13] *Rdf.* https://www.w3.org/RDF/.

[14] *Sparql*. https://www.w3.org/TR/rdf-sparql-query/.

[15] *Tornadofx*. https://tornadofx.io/.

[16] *Visual studio code*. https://code.visualstudio.com/.

[17] E. Bonabeau, *Agent-based modeling: Methods and techniques for simulating human systems*. https://www.pnas.org/content/pnas/99/suppl_3/7280.full.pdf.

[18] T. K. Carolin Johannsen, Diren Senger, *A diy sensor kit, gaussian processes and a multi-agent system fused into a smart beekeeping assistant*. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9154974.

[19] *Definitions of IoT*. https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT, https://www.trendmicro.com/vinfo/us/security/definition/internet-of-things.

[20] J. B. M. M. J. Bakakeu, F. Schäfer and J. Franke, *Building cyber-physical systems - a smart building use case*. https://www.researchgate.net/publication/318186249_Building_Cyber-Physical_Systems_-_A_Smart_Building_Use_Case_Foundations_Principles_and_Applications.

[21] H. M. Julie Dugdale, Mahyar T. Moghaddamt, *Agent-based simulation for lot facilitated building evacuation*. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9032909.

[22] G. K. P.-A. C. D. Longley, *W3c candidate recommendation for jsonld 1.1*. https://www.w3.org/TR/json-ld11/, 2020.

[23] *SAREF*. https://ontology.tno.nl/saref/.

[24] T. K. H. T. T. K. Shingo Yokoyama, Ryota Fukutani, *Match: Multiagent-based tactful cooperation scheme for heterogeneous iot devices.* https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8229379.

[25] *Smart Room Definitions.* https://www.revfine.com/smart-hotel-room/.

[26] *Smart Object.* https://en.wikipedia.org/wiki/Smart_object.

[27] *Thing Description.* https://www.w3.org/TR/wot-thing-description/.

[28] *W3C.* https://www.w3.org/.

[29] *Web Thing API.* https://webthings.io/api/.

[30] *WoT.* https://www.w3.org/WoT/.