Alma Mater Studiorum
Università di Bologna

CAMPUS DI CESENA
SCUOLA DI INGEGNERIA E ARCHITETTURA

Corso di Laurea in Magistrale in
INGEGNERIA E SCIENZE INFORMATICHE

Chat e Oggetti Intelligenti

Un'Interfaccia Conversazionale per l'Internet Of Things

Tesi in Sistemi Autonomi

Relatore:
Prof. Alessandro Ricci

Presentata da:
Greta Sasso

Correlatore:
Dott. Tommaso Dionigi

Sessione III · Anno accademico 2016 - 2017

Parole Chiave

ChatBot

Conversational Agents

Internet Of Things

Web Of Things

Agent Programming

Philips Hue

Dedico questo lavoro a chi ha sempre creduto in me, supportandomi e condividendo le mie passioni e le mie battaglie.

Introduzione

Le applicazioni di messaggistica e i social network sono oggigiorno i software maggiormente utilizzati nei dispositivi mobile, in quanto accorciano le distanze e semplificano notevolmente la comunicazione e la coordinazione fra persone, amici, famiglie. Il loro vasto utilizzo è favorito anche dall'ampia diffusione degli smartphone che rappresentano ormai lo strumento tecnologico must di cui ormai le persone non riescono a fare a meno.

La forte crescita dell'utilizzo dei servizi di messaggistica ha portato inoltre ad un forte sviluppo dei Chatbot, assistenti virtuali disponibili 24 ore su 24, con cui gli utenti finali possono messaggiare istantaneamente utilizzando il linguaggio naturale con l'obiettivo di richiedere e ottenere supporto relativo a vari ambiti, come ad esempio l'acquisto di beni, la richiesta di informazioni, la risoluzione di problemi.

In parallelo in questi ultimi anni si stanno diffondendo a livello esponenziale altre tecnologie: gli oggetti e i sensori dell'Internet of Things (IoT).

E' previsto per i prossimi anni un superamente anche del quantitativo degli smartphone nel mercato. In commercio sono presenti numerosi esempi in diversi settori come la domotica, la cura della persona, la medicina, i trasporti. Per quanto riguarda gli oggetti per gli utenti finali, la gestione di questi avviene normalmente con applicazioni mobile con diverse grafiche e interazioni in base all'azienda e al dispositivo in uso. Ne sono un esempio i braccialetti fitness tracker, le macchine da caffè, le lampadine smart, o ancora le prese intelligenti. Sono tutti dispositivi dell'Internet of Things accedibili tramite apposite applicazioni rilasciate dalle aziende produttrici.

Dati questi presupposti il **progetto** sviluppato si è posto l'obiettivo di fornire un diverso tipo di interfaccia per l'interazione con i dispositivi dell'Internet of Thing utilizzando non più le diverse applicazioni proprietarie mobile e desktop ma attraverso un'unica modalità, senza perdita alcuna di funzionalità, sfruttando il canale più utilizzato in assoluto: le applicazioni di messaggistica, anche dette chat. Gli oggetti dovranno quindi essere in grado di comunicare, rispondere e reagire alle domande e ai comandi degli utenti come se si comunicasse con una vera e propria entità umana. Si creerà quindi un ChatBot rappresentante non un assistente virtuale, ma il device stesso, l'oggetto smart. L'interazione avverrà per mezzo di dispositivi di uso quotidiano, come uno smartphone o un computer, e di una chat come canale di comunicazione.

L'utilizzo delle chat come interfacce degli oggetti smart semplifica notevolmente l'interazione. Questo perché nella maggior parte dei casi, l'Internet of Things richiede che gli oggetti siano accedibili con applicazioni mobile e/o desktop diverse e poco intuitive. In questo modo si può quindi ottenere una maggiore propensione all'acquisto dei device IoT e un loro utilizzo più prolungato nel tempo.

In particolare si vuole sviluppare un sistema composto dal kit contenente le lampadine Philips Hue in cui l'utente abbia la possibilità di interagirvi non tramite l'utilizzo dell'applicazione fornita dall'azienda, ma tramite una chat di messaggistica mobile, come ad esempio Telegram o Facebook Messanger, utilizzando il linguaggio naturale per esprimere i propri desideri, intenzioni, come se stesse dialogando con un'altra persona.

La tesi si sviluppa nella seguente modalità: In prima battuta si presenta il contesto tecnologico di riferimento partendo dall'Internet of Things e procedendo con le applicazioni di messaggistica e i Chatbot. Si prosegue analizzando i limiti attuali delle tecnologie considerate e le opportunità da

cui deriva il progetto, nonchè le motivazioni che possono supportare ulteriormente gli obiettivi che si vogliono ottenere nei confronti degli utenti finali e del mercato. Dopo queste due fasi si dettagliano gli obiettivi realizzativi, affrontando quindi l'analisi dei requisiti e l'individuazione delle entità principali del sistema di cui gli oggetti IoT e gli utenti fanno parte.

Data la novità del settore dell'IoT e il forte sviluppo dei Chatbot si affronta nella sezione 2.6 lo stato dell'arte con l'analisi degli studi, delle architetture e dei sistemi simili presenti in letteratura, che affrontano e approfondiscono il discorso delle interfacce conversazionali applicate agli oggetti intelligenti.

Con il capitolo 3 si entra nel concreto del progetto affrontando l'analisi del problema per individuare gli elementi e le interazioni necessari alla progettazione e allo sviluppo del sistema. Da questo deriva l'architettura del sistema. Una volta individuata l'architettura del sistema, si prosegue con la progettazione, preceduta da recap delle tecnologie abilitanti scelte, con le relative motivazioni. Questo perché alcune tecnologie influiranno sulla progettazione dettagliata del sistema. Con il capitolo 5 si continua per livelli seguendo l'architettura emersa nel capitolo 3, tenendo in considerazione gli eventuai punti critici emergenti dalle tecnologie scelte.

Successivamente si descrive il testing effettuato e si termina con la fase conclusiva per verificare il soddisfacimento dei requisiti preposti in analisi e per individuare le opportunità di uno sviluppo futuro.

Indice

Capitolo 1

Contesto di Riferimento

1.1 Internet of Things

> *"In the next century, planet earth will don an electronic skin. It will use the Internet as a scaffold to support and transmit its sensations"*
>
> - Neil Gross, *Business Week, 1999*

Con Internet Of Things, termine introdotto da Kevin Ashton nel 1999, si identifica una rete composta da oggetti concreti di uso comunem integrati da componenti elettronici in grado di comunicare l'uno con l'altro scambiandosi dati, cooperando e acquisendo così una propria identità digitale tramite gli indirizzi di rete IP.

Tali oggetti sono caratterizzati dall'essere connessi alla rete e dalla presenza di sensori e di attuatori. Permettono inoltre l'interazione degli oggetti con l'ambiente in cui sono situati, con gli utenti utilizzatori, e con la rete che include la presenza degli altri oggetti connessi. I sensori producono grosse quantità di dati, stream, con informazioni relative all'ambiente. A partire da questi si possono ottenere comportamenti più o meno smart elaborando i dati, con la possibilità eventuale di integrare algoritmi di Intelligenza Artificiale per derivare comportamenti e situazioni di alto livello. Ad esempio

quando un utente spegne il televisore e le luci della cucina e del salotto, un sistema potrebbe capire che l'utente ha l'intenzione di dormire e avvisarlo nel caso in cui altri sistemi o device siano attivi, come il condizionatore.

Nascono così gli smart objects, anche detti oggetti intelligenti. Oggigiorno sono molti gli esempi presenti nel mercato, dai fitness tracker ai sistemi di climatizzazione, o ancora le lampade o le prese elettriche intelligenti. Si stanno inoltre integrando concetti di alto livello come ad esempio l'Autonomia, l'Intelligenza artificiale, i sistemi ad agenti goal-oriented e i sistemi auto organizzati.

Con il forte sviluppo in questo ambito, si è sempre più propensi alla ricerca dell'antropomorfizzazione delle applicazioni e degli oggetti. Questi non avranno solo la capacità di interagire con gli utenti tramite interfacce grafiche o linguaggio naturale, ma saranno in grado di comprendere situazioni, prendere decisioni, e raggiungere degli obiettivi, mantenendo l'idea che il goal finale comune sarà sempre l'utente e la sua soddisfazione.

Gli ambiti applicativi sono numerosi, come ad esempio la domotica, il monitoraggio industriale, il tema della sicurezza, delle città o della robotica.

Gartner, Inc., stima che nel 2020 ci saranno circa 26 miliardi di oggetti connessi; per ABI Research saranno più di 30 miliardi. Da tutto questo deriva la possibilità che nella vita quotidiana possano essere influenzate molte azioni e abitudini delle persone, senza la necessità di avere un background informatico. Quando si dice che qualunque oggetto può diventare intelligente, non si fanno molte eccezioni. Si intende infatti qualsiasi cosa ci possa venire in mente. Il detto è "anything that can be connected, will be connected." [19].

1.2 ChatBot

L'utilizzo del servizio di messaggistica abbinato agli assistenti virtuali è ormai largamente utilizzato dagli utenti nei settori relativi all'acquisto di beni e all'utilizzo di servizi. Tanti sono infatti gli assistenti virtuali presenti

sul mercato. Alcuni dei più conosciuti, presentati da aziende internazionali, sono Google Now, Samsung S voice, Apple siri, e Cortana della Microsoft. In questi casi è l'utente a inizializzare l'interazione e a richiedere senza guida una qualche informazione o richiesta.

Sono presenti diverse soluzioni, molto meno utilizzate, per quanto riguarda invece l'applicazione delle chat agli oggetti e ai sistemi dell'Internet of Things. I chatterbot si possono distinguere in due macro tipologie: quelli che si basano sull'iniziativa del sistema e quelli basati sulla gestione del dialogo [10]. Con la prima tipologia l'esperienza dell'utente si riduce ad un botta e risposta guidato dall'agente che si trova dietro le quinte della chat. Nel secondo caso si ottiene una user experience completamente libera ed eventualmente guidata da alcune domande laddove si presentino delle ambiguità della richiesta da parte dell'utente o per altre necessità.

Esistono alcune piattaforme che offrono servizi di questo genere senza la necessità di implementare nulla e prescindendo da qualsiasi competenza di sviluppo software o algoritmica. Facilitano quindi l'integrazione da parte delle aziende di questi servizi a supporto dei cliente, con minimi sforzi e una resa accettabile. Queste mediano direttamente fra gli utenti attraverso diversi canali (come Facebook Messenger, sms, Telegram , ...) traducendo con dei template standard il testo scritto in input, in azioni da eseguire tramite le API dei device IoT. Alcune di queste piattaforme permettono inoltre la personalizzazione dei template dell'agente conversazionale dando la possibilità allo sviluppatore di addestrare il chatbot definendo gli elementi, le entità, ma soprattutto le azioni che deriveranno da determinate frasi. Alcuni esempi famosi sono Chatflue, Botsify, Motion.ai.

1.2.1 L'agente software conversazionale

Un chatbot non è che un programma software in grado di dialogare con l'utente utilizzando il linguaggio naturale tramite voce o testo scritto, con

l'obiettivo di simulare nel miglior modo possibile una conversazione con un essere umano. In letteratura ci si riferisce a questi sistemi di interazione in linguaggio naturale con diversi nomi, fra cui : chatbot, agenti conversazionali, chatterbot, spoken dialogue system, dialog agents [2].

Attualmente il loro sviluppo non è elevato, ma si inizia a registrare la loro presenza con il ruolo di customer care in contesti come lo shopping, prenotazioni, o l'assistenza agli utenti per determinati servizi. Una delle previsioni dell'azienda Gartner, Inc., riguarda proprio lo sviluppo dei bots: nel 2021 si prevede che più del 50% delle imprese spenderà più tempo nello sviluppo dei bots rispetto a quello di app tradizionali, con l'avvento dell'arrivo dell'era post-app [9]. Il continuo aumento della loro presenza è dovuto ad alcuni fattori, fra cui i bassi costi di sviluppo, la dominanza delle applicazioni di messaggistica, la riduzione dei download di applicazioni diverse l'una dall'altra, sia mobile che desktop, e dal supporto a disposizione dalle grandi aziende, come ad esempio Facebook, Microsoft, IBM [12].

Esistono diverse tipologie di agenti conversazionali che si distinguono in base alla complessità e al risultato che si vuole ottenere: i chatbot basati su regole, detti scripted, e basati sull'AI. I primi ricercano nel testo in input dall'utente delle parole chiave a cui associare delle risposte e delle azioni predefinite. Questo approccio ovviamente ha una potenzialità limitata. Per ottenere risultati più complessi occorre considerare più fattori, come la semantica di una intera frase e il contesto. La complessità cresce ulteriormente con le ambiguità presenti nella lingua utilizzata. Una ulteriore complicanza è relativa all'accomodamento dell'utente: più il bot risponde correttamente, più lo user tende a rilassarsi utilizzando un linguaggio maggiormente informale. Per questo, i chatbot più complessi utilizzano algoritmi di intelligenza artificiale AI con natural language processing (sistemi NLP). Alcune implementazioni considerano anche l'auto apprendimento. Con Natual Language Understand si intende invece la capacità di una macchina di comprendere il linguaggio umano.

Sono state nominate in precedenza le intenzioni e i desideri che l'utente esprime al chatbot e che devono essere compresi da questo. Si possono infatti individuare due elementi importanti che un chatbot deve identificare: gli intents e le actions.

- **Intents:** Qualsiasi input dell'utente esprime intenzioni, volontà, che devono essere mappate e comprese dall'agente conversazionale. Questo deve quindi produrre degli Intent specificati a tempo di design dell'agente stesso. Possono essere relativi alla richiesta di informazioni, riguardanti lo stato del sistema o dei suoi componenti, o relativi alla richiesta di esecuzione di azioni. A partire dall'intent saranno collegate, mappate, delle azioni, dette actions e le informaizioni aggiuntive, di contesto. Ad esempio, dato l'input dell'utente "Accendi le luci del bagno", "fai luce in bagno", dovrà essere riconosciuto un intent del tipo "lightOn" con informazione di contesto relativa alla stanza: "bathroom". Gli utenti possono infatti esprimere una stessa richiesta, volontà, in molti modi diversi ed è spesso difficile prevederli tutti. La collezione delle varianti possibili per uno stesso input può essere ottenuta quindi con una prima parte di testing, ricerca, fra gli utenti che un domani utilizzeranno il servizio.

- **Actions:** Sono le azioni effettive che dovranno essere eseguite, sono collegate alle intentions espresse dall'utente. Possono avere informazioni aggiuntive, parametri. Ad esempio nel caso di accensione di una luce, con presenza in un ambiente di più di una lampadina, sarà necessario ottenere una informazione aggiuntiva, un parametro, indicante quale lampadina nello specifico dovrà accendersi.

- **Dominio/Context:** Lo stato in cui si trova il sistema nel momento in cui un'intenzione viene espressa dall'utente. Il contesto potrebbe essere di aiuto ad eliminare ambiguità nelle richieste dell'utente, evitando così la necessità di rispondergli in modo negativo, o chiedendo per l'appunto parametri aggiuntivi.

- **Entities:** Ogni intenzione riguarda almeno un termine, un oggetto o una categoria di oggetti, definiti come entità. Dato l'esempio delle luci nel bagno si potrebbe definire l'entità "Light" che include di conseguenza una collezione di intents e di actions predefinita. Si possono infine collegare a una entità i suoi sinonimi.

Si possono individuare tre componenti relativi a un agente conversazionale:

- **Channel:** L'interazione con l'utente avviene tramite un canale di comunicazione specifico che presenta un linguaggio di sviluppo diverso dagli altri. Gli esempi più famosi sono: Facebook Messenger, Skype, Telegram, Teams, Twilio (sms).
  - Uno dei maggiori colossi nell'ambito dei servizi di messaggistica instantanea, Whatsapp, con 1.3 miliardi di utenti, si sta muovendo proprio in questo periodo (primi mesi del 2018) mettendo a disposizione una versione aziendale, Whatsapp Business. Con questa versione per le aziende, attualmente rilasciata solo per Android, si potranno automatizzare alcuni messaggi verso il cliente tramite l'applicazione. Le API non sono ancora state rese disponibili.
  - Telegram è stato il primo a far emergere questa nuova opportunità di business legata ai chatbot. Include comandi visuali, cioè la possibilità di integrare all'interno della chat pulsanti e elementi di input personalizzabili. Un vantaggio è l'estrema apertura delle API, che rende il sistema open-source. Un punto da tenere in considerazione è l'assenza di un assistente virtuale di base, che rende quindi il chatbot tanto smart quanto più elevate sono le capacità dello sviluppatore.
  - Facebook Messenger, con circa 1.6 bilioni di utenti attivi mette a disposizione i chatbot legandoli a delle pagine create su Facebook.
- **Engine:** La componente responsabile della comprensione dell'input dell'utente e della derivazione delle sue richieste, cioè delle intenzioni.

Si utilizzano in questa fase algoritmi di Machile Learning e di Natural Language Processing (NLP) che permettono l'integrazione di device e di utenti per quanto riguarda il linguaggio e la conseguente sua comprensione. Si possono utilizzare servizi a disposizione da terze parti (SaaS) . Gli esempi più rilevanti di NLP sono : IBM Watson Conversation, Luis della Microsoft, wit.ai di Facebook, Amazon Lapi.ai.

- **Platform:** Le fasi testing e di deploy possono essere attuate su piattaforme cloud (PaaS) . Si pongono come sistemi online per l'interazione diretta del chatbot con l'utente finale, eseguendo le azioni richieste e comunicando con altri sistemi e piattaforme.

1.2.2 Progettazione di un chatbot

In accordo con Wikipedia [6], si possono individuare tre fasi per la progettazione di un chatbot:

- **Design:** Si definiscono le interazioni previste con l'utente. In questa fase occorre definire le domande e le risposte che il chatbot dovrà essere in grado di elaborare. Per questo è utile effettuare una raccolta dei sinonimi e delle possibili variazioni di frase per esprimere uno stesso concetto o richiesta. La raccolta può essere stilata attraverso interviste o sondaggi agli utenti (Alpha Test).

- **Building:** Si divide nel processo di comprensione dell'intenzione dell'utente,intent, e nel processo di produzione della risposta corretta. Nel primo caso occorre considerare il livello di semplicità del bot, in quanto nel caso di chatbot basati su regole si potrebbero considerare parole chiave e pattern prelevati da un database costruito a tempo di design. Nel caso di chatbot AI, si elabora il testo con l'uso di un engine Natural Language Processing. Il secondo processo deriva da più elaborazioni che possono variare in base al contesto applicativo. Per entrambi si possono utilizzare i framework di sviluppo offerti dalle aziende che propongono degli strumenti di base per realizzare il proprio chatbot.

- **Analytics:** Fase di testing e di monitoraggio della correttezza del chatbot. Si può prevedere una lista di input possibili con i corrispondenti output e intents derivati. In questo contesto è altamente consigliato effettuare test con la collaborazione di utenti finali selezionati, con caratteristiche e background diversi, detti beta tester.

1.2.3 Framework di sviluppo

Permettono uno sviluppo più veloce, offrono strumenti, funzioni e classi per definire i comportamenti degli agenti conversazionali. Molti però permettono l'utilizzo e la scelta di un solo canale di comunicazione. I più famosi sono:

- Microsoft Bot Framework: Si divide in due componenti: Bot connector (framework di integrazione) e il component NLU (natural language understanding) detto LUIS.ai (Language Understanding Intelligent Service), della famiglia dei prodotti di Microsoft Azure. A differenza di molti framework, Microsoft, attraverso il bot connector permette l'integrazione del chatbot con diversi canali, come Telegram, Skype, Facebook messanger, Sms, Office365. Il pagamento va in base alle risorse che vengono consumate. É presente la possibilità di utilizzare la lingua italiana e l'utilizzo gratuito di Luis è limitato in base al numero delle chiamate API effettuate, in particolare limitato a 10,000 transazioni al mese. Mette a disposizione l'integrazione con i servizi Azure e con gli altri servizi cognitive.

- IBM Watson: è presente la possibilità di utilizzare la lingua italiana. Presenta gli algoritmi più potenti per il suo scopo ma la gestione delle API non è immediata. Sono disponibili funzionalità per l'analisi del testo. Anche in questo caso è presente una componente per l'integrazione di alcuni canali: IBM Bluemix (PaaS) . Il servizio è gratuito con un limitato numero di intent ed entity utilizzabile.

- Facebook Bot Engine: Come canale è utilizzato solo Facebook Messenger, data la vasta utenza che possiede. Si basa sulla tecnologia Wit.ai, startup nata con lo scopo di consentire la conversione di input vocali in formato testuale, adattabile ad altri utilizzi simili. Questo corrisponde all'engine e si basa su algoritmi di Machine Learning. Estraendo le intenzioni degli utenti permette inoltre l'analisi dei sentimenti derivanti dal testo inserito. Mette a disposizione un set di entità predefinite, derivanti anche dai modelli forniti da Cortana. Sono disponibili servizi di conversione del parlato in formato testuale scritto ed è presente la possibilità di utilizzare la lingua italiana. L'utilizzo di Wit.ai è gratuito e senza limitazioni.

- API.ai di Google: Integra diversi canali di messaggistica e mette a disposizione molti contesti, comprendenti similitudini, intenzioni e azioni, evitando così l'eventuale inserimento da parte dello sviluppatore delle alternative a un determinato input da parte dell'utente. I chatbot creati possono essere esportati come moduli integrabili in Cortana, Facebook, e altri portali. Mette a disposizione servizi di conversione del parlato in formato testuale scritto. L'NLU si basa su algoritmi che sfruttano l'apprendimento supervisionato, per mezzo di esempi per addestrare il sistema e fornire riferimenti di collegamenti input e output corretti. É presente la possibilità di utilizzare la lingua italiana.

- Amazon Lex: Una delle ultime aziende che ha introdotto un framework per i chatbot. Mette a disposizione servizi di conversione del parlato in formato testuale scritto, utilizza algoritmi di apprendimento di Amazon Alexa, col possibile uso della lingua italiana. Permette inoltre l'integrazione dei servizi di AWS. Il servizio è completamente gratuito e i pagamenti si basano sulla quantità di utilizzo effettivo.

1.3 Opportunità

In questa sezione si procede con l'individuare le problematiche e le opportunità da cui deriva il progetto.

1.3.1 Sviluppo dei Chatbot

L'evoluzione tecnologica di questi anni sembra che sia destinata all'arrivo dell'era post-app, come dichiarato da David Willis, Vice President and Distinguished Analyst , Gartner, Inc., in un articolo del 2016 [8] . In questa nuova fase evolutiva, il linguaggio umano sarà per l'appunto la nuova interfaccia utente in cui le applicazioni saranno unificate e integrate in sistemi di messaggistica, evitando così il largo utilizzo di diverse applicazioni, in favore di una sempre maggiore semplicità e familiarità. Il focus sarà sulle tecnologie emergenti come i chatbots, gli assistenti virtuali personali, interfacce conversazionali.

1.3.2 Applicazioni di Messaggistica

Dato il largo utilizzo delle chat, come Facebook Messenger, Telegram o Whatsapp, è facile pensare come sarebbe utopico poter utilizzare tutte le funzionalità dei device IoT all'interno di queste app già presenti nella maggiorparte degli smartphone, con il linguaggio naturale, come se si stesse parlando con un operatore o un conoscente.

1.3.3 Usabilità

Con l'introduzione nelle case e nella vita quotidiana di questi nuovi dispositivi, cresce automaticamente la necessità di ottenere semplicità e uniformità nell'interazione con essi.

L'utente nella maggior parte dei casi ha l'opportunità di interagire e di ottenere informazioni dai dispositivi di una determinata azienda, utilizzan-

do una interfaccia web o una applicazione messa a disposizione dall'azienda stessa.

Questa richiede che l'utente impari i comandi e le modalità di interfacciamento con le features presentate. L'utente dovrebbe utilizzare un applicativo graficamente semplice e intuitivo, che non lo confonda nell'utilizzo ma che gli consenta maggiore velocità e un maggiore voglia di confronto con la nuova tecnologia. Molto spesso si ci ritrova di fronte a software con una grafica, una user experience ottimale, che però non permettono di ottenere tutte le funzionalità a disposizione.

Una usabilità non ottimale può portare a un ridotto utilizzo dei device.

1.3.4 Eterogeneità

L'utente è spesso costretto a installare applicazioni mobile o desktop per ogni singolo dispositivo, con interfacce e modalità di interazioni diverse. Questo perché non c'è una uniformità tra le aziende nel distribuire nel mercato le interfacce per l'utilizzo dei prodotti e dei servizi. La moltitudine di app che l'utente si ritrova all'interno dello smartphone, insieme alle diverse regole e ai diversi utilizzi che queste comportano, crea confusione e sconforto nell'utilizzazione. Per questo si è scelto di selezionare un'interfaccia comune e semplice, le chat, mediando così fra gli utenti utilizzatori e i device con i loro sistemi già sviluppati dalle aziende, tramite dei livelli e delle entità intermediarie intelligenti.

1.3.5 Installazione, Supporto e Manutenzione

Uno degli ostacoli maggiori che porta l'utente al non acquistare e al non utilizzare i device IoT, sono le installazioni e le configurazioni necessarie per l'interazione. Queste dovrebbero essere ridotte al minimo indispensabile, se non eliminate totalmente. Soprattutto per il fatto che molti device IoT nascono con l'intento di semplificare e/o migliorare azioni di vita quotidiana in cui l'utente non utilizza strumenti tecnologici. Deve essere quindi stimolato

ad interagire con la tecnologia consapevole del fatto che senza alcun sforzo otterrà dei benefici.
Inoltre, durante l'utilizzo, sono da considerare i possibili errori e guasti, così come i necessari ed eventuali aggiornamenti software. Anche questi dovrebbero essere automatizzati ed effettuati da parte del sistema, senza che l'utente debba preoccuparsene.
Tramite una applicazione di messaggistica in cui l'utente comunica direttamente con il dispositivo attraverso un assistente, un chatbot, si potrebbe creare un supporto interno e dei tutorial nel caso in cui siano necessarie delle azioni da parte dell'utente.

1.3.6 Notifiche Push

Si consideri inoltre che recentemente sono nate nuove forme di interazioni con gli oggetti, in quanto non solo l'utente invia comandi a questi, ma riceve da essi informazioni, notifiche, per volontà degli oggetti stessi. Questo può essere semplicemente pensato come un messaggio ricevuto dall'utente finale all'interno della chat.

1.3.7 Antropomorfizzazione

Spesso gli utenti hanno una tendenza all'antropomorfizzazione dei robot, degli oggetti e dei servizi, che si esplica con la comunicazione verbale. Si pensi ad esempio ai litigi nati con un elettrodomestico malfunzionante o all'attribuzione di un nome alla propria auto e all'incoraggiamento di questa nel caso in cui non si metta in moto prima per andare al lavoro... questi sono alcuni esempi in cui gli oggetti sono trattati come esseri umani. In particolare le persone tendono a usare il linguaggio informale, parlato, con gli oggetti di fronte a situazioni nuove e/o critiche. Gli utenti tendono inoltre a fare collegamenti e confronti con comportamenti umani, e a parlare con questi.

1.3.8 Chattando con le Lampadine

Si pongono due diversi obiettivi realizzativi per la progettazione di questo sistema. Considerando che si vuole progettare e realizzare una architettura che ponga gli utenti in grado di comunicare tramite chat con tutti i device, questo implica la considerazione del problema dell'eteogeneità dei device delle diverse aziende. L'architettura dovrà essere modulare e con diversi livelli di astrazione per demandare in un secondo momento questa problematica.
Un primo prototipo esclude a livello pratico il problema, prendendo in considerazione una tecnologia specifica scelta, cioè il sistema Philips Hue. Il secondo caso, posticipato a una fase futura di sviluppo dovrà essere l'omogeneizzazione, per quanto possibile, di tutti i sistemi e dispositivi smarts presenti e non sul mercato, con metodologie possibilmente automatizzate.

Si è selezionato il sistema Hue della Philips che include un set di lampadine intelligenti, controllabili con una applicazione mobile. Il sistema della Philips è composto da un dispositivo coordinatore che funge da intermediario fra delle lampadine e un'applicazione desktop e mobile che l'utente può utilizzare per l'accensione, lo spegnimento, la modulazione della luminosità e la tipologia del colore della lampadina.

L'idea è che quindi l'utente possa interagire con una lampadina passando da un chatbot come se fosse una persona, ponendo domande come "le luci del soggiorno sono accese?" o comandi come " spegni le luci del bagno" .

Capitolo 2

Analisi dei Requisiti

L'obiettivo da realizzare è quello di fornire all'utente la possibilità di dialogare con i device e quindi di ottenere all'interno di una chat una antropomorfizzazione di un oggetto smart. Questa deve inglobare le funzionalità messe a disposizione dal dispositivo IoT così che l'utente possa utilizzare la chat sostituendo completamente le applicazioni rilasciate o/e i comandi fisici presenti sull'oggetto stesso. Il sistema Hue della Philips può includere da 1 a 50 lampadine per bridge. Si pone l'obiettivo di considerare la gestione multipla di lampadine, eventualmente posizionate in stanze diverse, per studiare e ottenere un sistema più complesso all'interno della chat.

Si sfrutterà quindi l'integrazione di un Chatbot che comunicherà e interagirà con lo user tramite l'applicazione di messaggistica, e con l'end-device IoT, le lampadine della Philips.

2.1 Glossario

Termine	Descrizione	Sinonimi	Collegamenti
IoT	Una rete di oggetti fisici embedded (arricchiti con hardware e/o software) connessi a internet	Internet of Things; Internet delle cose	Oggetto Smart
Oggetto smart	Un qualsiasi oggetto dotato di capacità computazionali e collegamento alla rete	Device IoT; Oggetto intelligente; Smart thing	IoT
Chat	Software di messaggistica istantanea, per la comunicazione fra più utenti in tempo reale	Applicazione di messaggistica	ChatBot
ChatBot	Software in grado di simulare una conversazione in linguaggio naturale fra un utente reale e una macchina	Agenti conversazionali, Chatterbot, Spoken Dialogue System, Dialog Agents	Chat

Termine	Descrizione	Sinonimi	Collegamenti
Sistema Philips Hue	Sistema composto da un bridge che ha la funzione di gestire le lampadine tramite l'utilizzo di una applicazione mobile o delle API	Lampadine; Oggetti IoT	Oggetto Smart; IoT

Tabella 2.1: Glossario

2.2 Requisiti Funzionali

Partendo dall'obiettivo realizzativo concreto che include il sistema di illuminazione delle lampadine Hue della Philips, un utente potrà generalmente avere l'intenzione di effettuare tre tipologie di operazioni nei confronti di una singola lampadina.

- Accensione e spegnimento di una lampadina
- Modifica del colore o della luminosità di una lampadina
- Richiesta di informazioni riguardanti lo stato di una lampadina

Il tutto dovrà essere richiesto tramite l'uso della chat attraverso il linguaggio naturale che per l'utente risulta familiare e semplice. Così lo user non dovrà avere la necessità di studiare la documentazione per l'interazione (Es. regole, sintassi, pulsanti grafici, ...). Il sistema dovrà essere in grado di rispondere all'utente in caso di ambiguità per chiedere informazioni aggiuntive. Si desidera che il sistema sia in grado di fornire suggerimenti all'utente determinati in completa autonomia.

2.3 Casi D'uso

2.3.1 Attori

Gli attori del sistema sono due: l'utente e l'oggetto IoT con cui si vuole interagire per mezzo del sistema tramite la chat.
In un contesto di uso standard delle lampadine l'utente deve installare l'applicazione fornita dalla Philips nel dispositivo mobile che si vuole utilizzare (smartphone, tablet, pc, ...). Tramite questa può effettuare diverse operazioni riassumibili in:

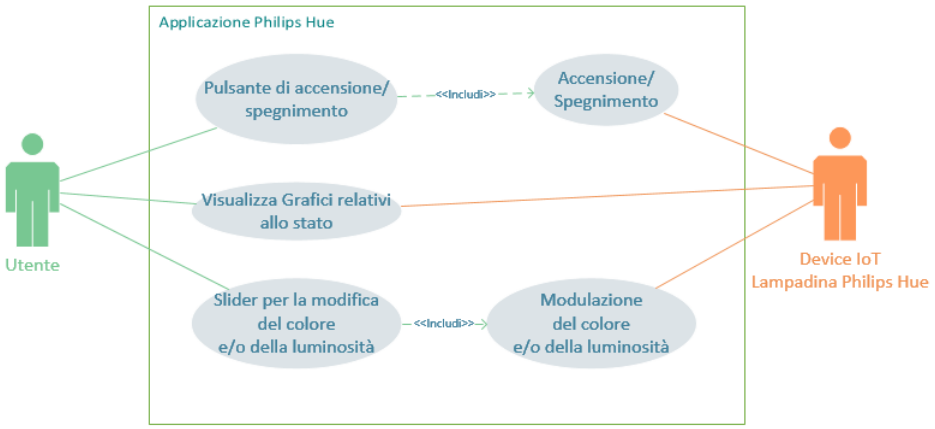


Figura 2.1: Casi D'uso relativi all'Applicazione Mobile Philips Hue

Incluse in queste funzionalità sono presenti operazioni composte e che integrano altri servizi, come ad esempio l'esecuzione di sequenze di cambi di luce e colori o l'impostazione di timer.

Un caso d'uso aggiuntivo che si può integrare nel sistema è dato da un livello più elevato di intelligenza che si vuole ottenere. Non si vogliono avere solo le funzionalità di base che i device IoT mettono a disposizione all'esterno, ma si vuole ottenere un comportamento autonomo, proattivo intelligente, dato dal controllo di errori, consumi energetici e altre caratteristiche, per fornire suggerimenti e avvisi all'utente.

Dato l'obiettivo del sistema da realizzare, l'utente potrà ottenere i medesimi risultati delle operazioni di base, richiedendoli direttamente all'interno di una applicazione di messaggistica istantanea, riferendosi all'entità con cui si chatta come se fosse la lampadina vera e propria.

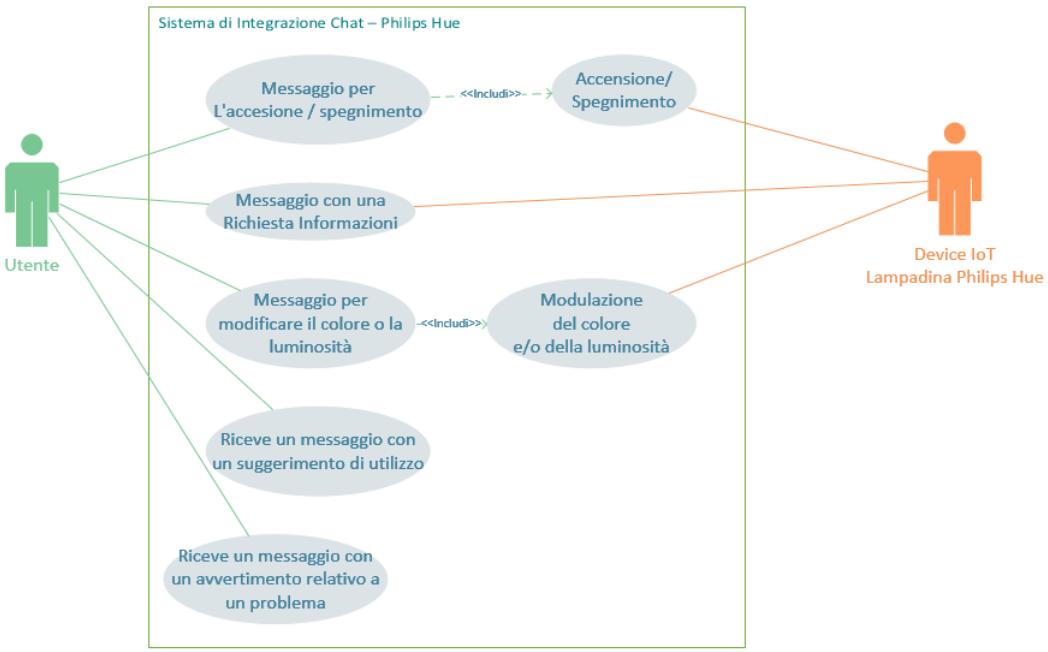


Figura 2.2: Casi D'uso relativi al sistema di integrazione ChatBot - IoT

ID	Caso D'uso
RF-1	Messaggio per l'accensione / spegnimento
RF-2	Messaggio con una Richiesta di informazioni
RF-3	Messaggio per modificare il colore o la luminosità
RF-4	Riceve un messaggio con un suggerimento di utilizzo
RF-5	Riceve un messaggio con un avvertimento relativo a un problema
RF-6	Accensione/Spegnimento
RF-7	Modulazione del colore e/o della luminosità

Tabella 2.2: Requisiti Funzionali

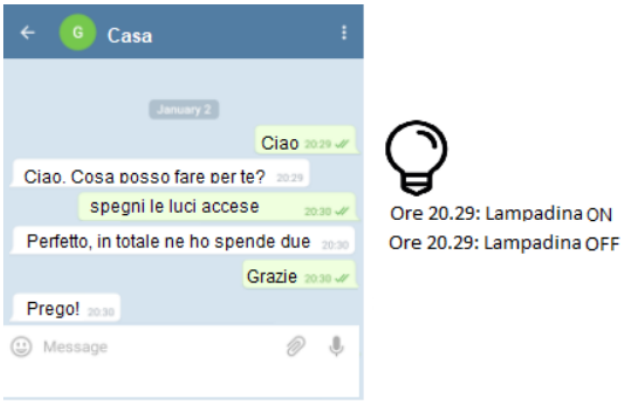
2.3.2 Scenari

1.
   - **ID Caso D'uso:** RF-1, RF-6
   - **Priorità:** 1
   - **Descrizione:** L'utente richiede l'accensione
   - **Durata:** Immediata, massimo qualche minuto considerando gli scenari alternativi
   - **Attore Primario:** Utente
   - **Attore Secondario:** IoT Device
   - **Precondizioni:** Le lampadine e il bridge devono essere collegati alla rete. Occorre aver stabilito una connessione fra l'app di messaggistica e il device. Il fatto che la lampadina sia accesa prima della richiesta non influirà sul comportamento.
   - **Garanzie di Successo:** La/e lampadina/e deve essere correttamente accesa
   - **Scenario Principale:**

Ore 20.29: Lampadina OFF
Ore 20.29: Lampadina ON

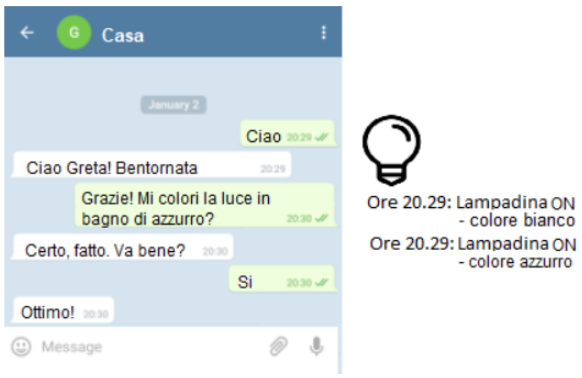
- **Scenari Alternativi:**
  - Nel Caso di ambiguità relativamente a quale lampadina accendere il sistema dovrà chiedere maggiori informazioni all'utente o avvertirlo della possibile problematica
  - Possono essere forniti suggerimenti da parte del sistema. Come ad esempio lo spegnimento delle luci presenti in altre stanze per risparmiare energia
  - Nel caso in cui la lampadina sia guasta o non sia raggiungibile, avvisare l'utente

2. - **ID Caso D'uso:** RF-1, RF-6
   - **Priorità:** 2
   - **Descrizione:** L'utente richiede lo spegnimento di una lampadina
   - **Durata:** Immediata, massimo qualche minuto considerando gli scenari alternativi
   - **Attore Primario:** L'utente
   - **Attore Secondario:** Device IoT
   - **Precondizioni:** Le lampadine e il bridge devono essere collegati alla rete. Occorre aver stabilito una connessione fra l'app di messaggistica e il device. Il fatto che la lampadina sia spenta prima della richiesta non influirà sul comportamento.
   - **Garanzie di Successo:** La lampadina è spenta dopo la richiesta
   - **Scenario Principale:**



- **Scenari Alternativi:** Sono i medesimi del caso d'uso RF-1 (2.2)

3.
- **ID Caso D'uso:** RF-3, RF-7
- **Priorità:** 3
- **Descrizione:** L'utente richiede la modifica del colore o della luminosità
- **Durata:** Immediata, massimo qualche minuto considerando gli scenari alternativi
- **Attore Primario:** L'utente
- **Attore Secondario:** Device IoT
- **Precondizioni:** Le lampadine e il bridge devono essere collegati alla rete. Occorre aver stabilito una connessione fra l'app di messaggistica e il device. Le lampadine devono essere accese.
- **Garanzie di Successo:** Le lampadine devono essere correttamente modificate
- **Scenario Principale:**



- **Scenari Alternativi:**
  - Nel caso in cui la lampadina sia spenta, richiedere all'utente se la vuole accesa
  - Nel caso di ambiguità relativamente a quale lampadina accendere: il sistema dovrà chiedere maggiori informazioni all'utente o avvertirlo della possibile problematica
  - Possono essere forniti suggerimenti o curiosità relativi al colore scelto o al livello di luminosità
  - Nel caso in cui la lampadina sia guasta o non sia raggiungibile, avvisare l'utente

4.
   - **ID Caso D'uso:** RF-2
   - **Priorità:** 4
   - **Descrizione:** L'utente vuole sapere se una lampadina è accesa o spenta, o altre informazioni relative al colore o alla luminosità
   - **Durata:** Immediata, massimo qualche minuto considerando gli scenari alternativi
   - **Attore Primario:** L'utente
   - **Attore Secondario:** Il device IoT
   - **Precondizioni:** Le lampadine e il bridge devono essere collegati alla rete. Occorre aver stabilito una connessione fra l'app di messaggistica e il device.

- **Garanzie di Successo:** L'utente ottiene informazioni coerenti e consistenti
- **Scenario Principale:**

Ore 20.29: Lampadina ON - colore azzurro

- **Scenari Alternativi:**
  - Nel caso di ambiguità della richiesta da parte dell'utente

5.
- **ID Caso D'uso:** RF-4 , RF-5
- **Priorità:** 5
- **Descrizione:** Il sistema invia in modo autonomo (per propria volontà) notifiche all'utente per errori, guasti, suggerimenti
- **Durata:** Immediata
- **Attore Primario:** L'utente
- **Precondizioni:** Le lampadine e il bridge devono essere collegate alla rete.
- **Garanzie di Successo:** Le informazioni devono essere coerenti e consistenti
- **Scenario Principale:**

2.4 Requisiti Non Funzionali

Per la definizione dei requisiti non funzionali si seguono le linee guida per la qualità del software individuate dalle normative ISO/IEC 9126.

ID	Requisito	Descrizione
RNF-1	Funzionalità	In quanto deve soddisfare i requisiti funzionali
RNF-1.1	Accuratezza	Nei risultati prodotti dalle operazioni richieste eseguite
RNF-1.2	Interoperabilità	Nei confronti di diverse applicazioni di messaggistica, ma soprattutto nei confronti di diversi oggetti smart, di diverse aziende e di diversa tipologia
RNF-1.3	Sicurezza	Nei confronti di eventuali violazioni dei dati personali e nell'utilizzo da parte di persone non autorizzate degli oggetti smart

ID	Requisito	Descrizione
RNF-2	Affidabilità	Il sistema deve assicurare affidabilità e sicurezza nella gestione degli oggetti, quello che l'utente richiede deve essere rispettato e non deve effettuare azioni che l'utente non desidera o che possono nuocere in alcun modo all'utente
RNF-2.1	Maturità	Capacità del sistema dell'evitare malfunzionamenti o che i risultati siano scorretti. Questo deve avvenire a partire da una adeguata gestione dei guasti da parte del sistema
RNF-2.2	Tolleranza agli errori	Il sistema di fronte a situazioni impreviste o errate, come un input da parte dell'utente non riconosciuto o ambiguo, deve rispondere in maniera coerente
RNF-2.3	Aderenza	Ad alcuni standard, ad esempio in modo che sia coerente nelle risposte date all'utente
RNF-3	Efficienza	In termini di velocità nella risposta e nell'esecuzione delle richieste da parte del cliente, senza consumare risorse aggiuntive nei device utilizzati (applicazione di messaggistica e oggetto smart)
RNF-3.1	Comportamento rispetto al tempo	Per mantenere adeguati i tempi di risposta e di elaborazione delle richieste dell'utente

ID	Requisito	Descrizione
RNF-3.2	Utilizzo delle risorse	Deve essere adeguato e non portare disagi aggiuntivi all'utente. Ad esempio non deve portare a un maggior consumo energetico da parte dei dispositivi usati
RNF-4	Usabilità	Il requisito con maggior rilevanza, dato che è la motivazione scatenante della realizzazione del sistema. L'utente deve capire, utilizzare e accettare il sistema senza rilevanti complessità aggiuntive.
RNF-4.1	Comprensibilità	Per l'utente deve essere immediato intuire come il sistema funziona
RNF-4.2	Apprendibilità	Facilità nell'apprendimento dell'utilizzo del sistema
RNF-4.3	Operabilità	Semplicità di utilizzo, soprattutto per quanto riguarda il linguaggio utilizzato per interagire con il sistema, che deve esssere il più naturale possibile
RNF-4.4	Attrattività	Il sistema può soddisfare bisogni e desideri dell'utente
RNF-5	Manutenibilità	A fronte di modifiche, queste possono essere effettuate in modo agile, ciò deriva principalmente da una buona progettazione e dalla modularità del sistema
RNF-5.1	Modificabilità	Facilità di modifica da parte degli sviluppatori, questo deve avvenire grazie a una buona progettazione iniziale

ID	Requisito	Descrizione
RNF-5.2	Stabilità	Modifiche del sistema non devono comportare effetti indesiderati o imprevisti in fare di esecuzione
RNF-5.3	Testabilità	Tutti i requisiti funzionali, dettagliati con i possibili scenari, devono essere verificabili

Tabella 2.3: Requisiti Non Funzionali

2.5 Entita' del Sistema

Da questo emerge intuitivamente un primo schema dei componenti e delle interazioni con gli attori:

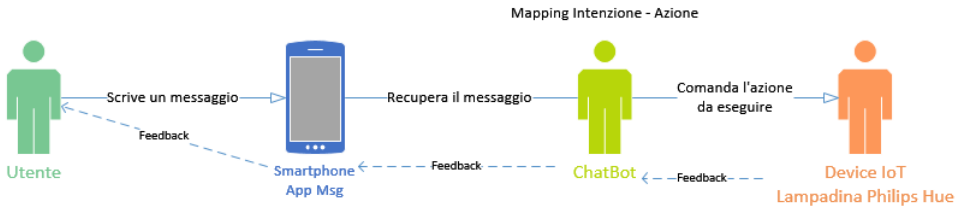


Figura 2.3: Schema delle entità principali e delle interazioni di base

2.6 Stato dell'arte

L'analisi della letteratura è stata effettuata tramite la documentazione pubblicata attraverso i portali IEEE, ACM Digital Library, e Google Scholar. Si sono ricercati sistemi e architetture similari che includono device IoT (o sistemi Smart Home) e una chat come interfaccia per l'utente. Si sono utilizzate due categorie di parole chiavi, una relativa al chatbot e una ai device IoT, utilizzando i collegamenti e i sinonimi visti in precedenza all'interno del glossario, in lingua inglese:

- Chatbot, Conversational Agent, Chat, Messaging App
- Internet of Things, Smart Object, Device IoT, Smart Thing, Philips Hue

Sono diversi i casi individuati di sistemi composti da uno o più device IoT e una chat. Il confronto di questi si baserà sulle entità individuate nella figura 2.3: Utente, App-msg, Chatbot, IoTDevice. Si procede inoltre per livello di astrazione e di complessità, partendo dai sistemi più semplici e diretti nel realizzare l'obiettivo prefissato.

- In [1] si è individuato un sistema che come App-Msg propone una web application che connette qualsiasi IoT Device nella stessa rete locale LAN. Sono stati utilizzati per il primo prototipo dei semplici sensori e non device IoT presenti nel mercato. Per la realizzazione del sistema, in particolare per l'implementazione dell'entità chatbot intermediaria, è stato utilizzato il microcontrollore raspberry Pi che sfrutta algoritmi di base NLP, con semplice collegamento fra azioni e API. Non sono utilizzati algoritmi di Machine Learning. Come features aggiuntiva, per ottenere un comportamento di maggior impatto, è stato implementato un controllo di intrusione tramite della semplice sensoristica.
- In [16] è stata sviluppata una smart home che si basa su un protocollo alternativo al Wifi, cioè Zigbee, tipico dell'IoT dati i suoi bassi costi e consumi. Si è individuata una delle motivazioni emersa nella presentazione della problematica del presente documento: il difficile approccio iniziale dell'utente con i vari IoTDevice. In particolare si è posto il focus nella configurazione inziale, in cui l'utente si ritrova ad utilizzare molte applicazioni, l'una diversa dall'altra. L'obiettivo che il lavoro si pone è l'annullamento di qualsiasi configurazione iniziale da parte dell'utente e la creazione di un semplice accoppiamento fra azioni e input testuale (utilizzando il software weChat come canale), per i comandi degli IoTDevice. Non è utilizzato alcun algoritmo di elaborazione testuale (NLU).

- In [15] è stato implementato un sistema che ha l'obiettivo di fornire una architettura semplice per collegare gli IoTDevice con il canale Facebook Messenger. Sono stati utilizzati IoTDevice simulati. Anche in questo caso si ha un semplice mappaggio di input in comandi diretti, senza l'utilizzo di alcun algoritmo di elaborazione del testo.

- In [11] si discute e si analizza la possibilità e la potenzialità di progettare un sistema IoT controllabile con linguaggio naturale, utilizzando la piattaforma Quickscript, sviluppata dal team stesso degli autori. Questa permette di creare entità di conversazioni virtuali e il documento approfondisce proprio la possibilità di includere algoritmi NLP per controllare oggetti IoT.

- Col brevetto [7] e [4], si è implementato un sistema in cui l'utente tramite un'applicazione di messaggistica può interagire, anche da remoto, con i device di una smart home tramite l'uso del linguaggio naturale, interrogandone lo stato e inviando semplici comandi predefiniti. Sebbene quest'ultima scelta è limitante in quanto l'utente dovrà essere formato nell'utilizzo della chat per poter ottenere i risultati desiderati.

- Un'analisi puramente teorica è stata effettuata in [14] in cui vengono esplorate le ipotesi future di interfacciamento fra l'IoT e gli utenti, fra cui appunto l'interfaccia conversazionale. Fra le varie ipotesi emerge la necessità di avere un livello middleware per aggiungere un grado di intelligenza e di autonomia al sistema, utilizzando approcci relativi all'auto organizzazione, ai sistemi multi agente, e con tecniche di Machine Learning. Si parla per l'appunto del passaggio da un sistema basato su comandi (dell'utente ai device) a sistemi basati su goals da parte degli IoTDevice. Un esempio è l'auto relagolazione della luminosità di una lampada in base al contesto ambientale in un dato momento.

- Il caso più rilevante e similare agli obiettivi preposti in precedenza, si trova nel paper [10] . In questo viene presentato infatti un approccio di

più alto livello in cui si utilizza un agente intelligente conversazionale per l'IoT home. A differenza di molti altri è incluso il concetto di NLU per determinare l'intenzione espressa dall'utente tramite linguaggio naturale. Non un semplice chatbot a comandi. Sono inclusi altri livelli intermedi per determinare il piano da eseguire relativamente allo stato dell'ambiente in ogni determinato momento, per generare delle azioni da eseguire. Questo segue il paradigma ad agenti goal-oriented. Uno dei vantaggi dato da questa architettura è la possibilità di avviare la conversazione da parte dell'utente e anche da parte del sistema stesso. Il sistema è stato attualmente implementato e testato utilizzando alcuni device fra cui un condizionatore, un termostato, sensori e anche il sistema Hue della Philips.

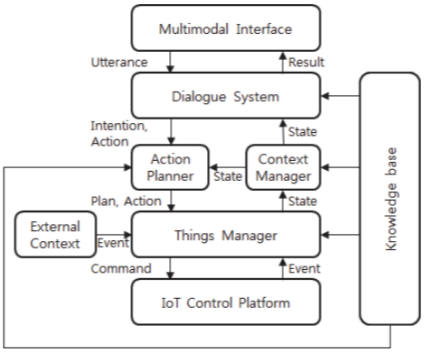


Figura 2.4: Architettura proposa nel paper [10] relativa a un agente intelligente conversazionale per l'IoT Home

Opportunità

Le entità principali individuate dall'analisi dei requisiti sono presenti pressochè in tutti i riferimenti riportati nello stato dell'arte. Tuttavia l'unione di tutti i requisiti funzionali e non, è assente.

Facendo riferimento ai requisiti funzionali RF-1,-2,-3,-6,-7 questi possono essere assimilabili a semplici comandi corrispondenti alle API messe a disposizione dai device IoT, cosa in comune a tutti i sistemi individuati in letteratura. Una differenza grande con molti è invece l'assenza del concetto

di autonomia, di analisi di alto livello dei dati per far sì che siano gli oggetti smart stessi a prendere iniziativa e ad agire con l'ambiente e con l'utente stesso (RF-4,-5). Un altro punto è l'assenza di comportamenti complessi, cioè composti da azioni elementari.

In letteratura molti sistemi si basano su sistemi conversazionali a comando, in cui è l'utente che inizializza la conversazione, e in cui i comandi sono predefiniti o mappati per parole chiave alle corrispondenti azioni, limitando l'usabilità e la facilità di utilizzo del sistema per l'utente: RNF-4, -4.1, -4.2, -4.3, -4.4. Sono infatti pochi i casi in cui è utilizzata qualche tecnica di Intelligenza Artificiale, o più nello specifico di NLU, per derivare dal linguaggio naturale dell'utente le intenzioni.
Un altro requisito non soddisfatto da molti è l'interoperabilità, RNF-1.2, sia per quanto riguarda la possibilità di usare diversi canali di messaggistica e diversi dispositivi IoT, appartenenti a differenti aziende produttrici.
Un punto critico è l'utilizzo di almeno uno dei canali più utilizzati dai dispositivi mobile, come ad esempio Facebook Messenger o Telegram, RNF-4.4, in quanto sono stati utilizzati canali diversi,come weChat, o app create ad hoc, che richiederebbero quindi l'installaziona una nuova app mobile e un minimo di formazione relativo al suo utilizzo, RNF-4, -4.1, -4.2, -4.3, -4.4.

Importanti sono inoltre le motivazioni che si trovano dietro alla raccolta dei requisiti. L'utente potrà coordinare gli oggetti intelligenti e cooperare con essi, utilizzando il suo linguaggio naturale. Di conseguenza, per poter meglio soddisfare le necessità dell'utente e meglio rispondere ad esso, occorre analizzare quali sono le motivazioni dell'utilizzo dei sistemi IoT, capire il perché è necessario integrare le interfacce conversazionali all'uso dei device. Non basta più un'analisi dei benefici, ma occorre un'analisi della psicologia e del comportamento dell'utente stesso.

Capitolo 3

Architettura

3.1 Entità

L'utente utilizzerà il linguaggio informale per esprimere un suo desiderio, una sua volontà, indicata con il termine "intenzione", tramite uno dei canali di messaggistica pre-esistenti. Questa deve essere elaborata in modo tale da ottenere una azione mappata con le corrispondenti API dei device IoT. Ne deriva la necessità di avere una entità autonoma in grado di elaborare il testo tramite algoritmi di apprendimento e di elaborazione, per poter derivarne l'intenzione ed eventuali soggetti e contesti. Questo compito è assunto dall'agente conversazionale, detto chatbot. Inoltre a seconda dello stato dell'ambiente, del sistema, e del contesto della conversazione, una certa intenzione potrebbe assumere un diverso risultato o produrre una diversa risposta.

Per quanto riguarda le features consistenti in attività autonome prodotte dal sistema, come l'invio di suggerimenti o il monitoraggio dello stato del sistema stesso, occorre considerare che il chatbot non dovrà essere solo in grado di effettuare operazioni di mapping intention - azione, ma ragionamenti e azioni eseguiti in autonomia, proattivamente, a prescindenre dall'input dell'utente. Si può quindi wrappare l'entità chatbot in una entità di più alto livello che funge da intermediario e che si compone della componente chatbot

e della componente autonoma, proattiva di ragionamento e monitoraggio. Per la caratteristica di autonomia completa, dalla letteratura scientifica, anche questa sarà detta agente. Si può rinominare l'entità intermediaria come Agente middleware.

Si otterrà una macro entità intermediaria fra lo user e l'end-device scomponibile in due agenti: uno conversazionale, il Chatterbot, che ha lo scopo di prendere in input i dati grezzi , il testo informale dell'utente e di derivarne l'intenzione con algoritmi NLP passandola quindi a un secondo agente , che sarà detto agente di reasoning che si occuperà poi della gestione e dell'interazione con i device IoT. Questo agente rappresenterà un device IoT singolo o un insieme, memorizzerà informazioni sull'utente, sullo stato del sistema, eventualmente sull'ambiente, ed effettuerà operazioni ed elaborazioni sui dati in input da entrambi gli attori proattivamente, con la possibilità di triggerare esso stesso delle conversazioni.

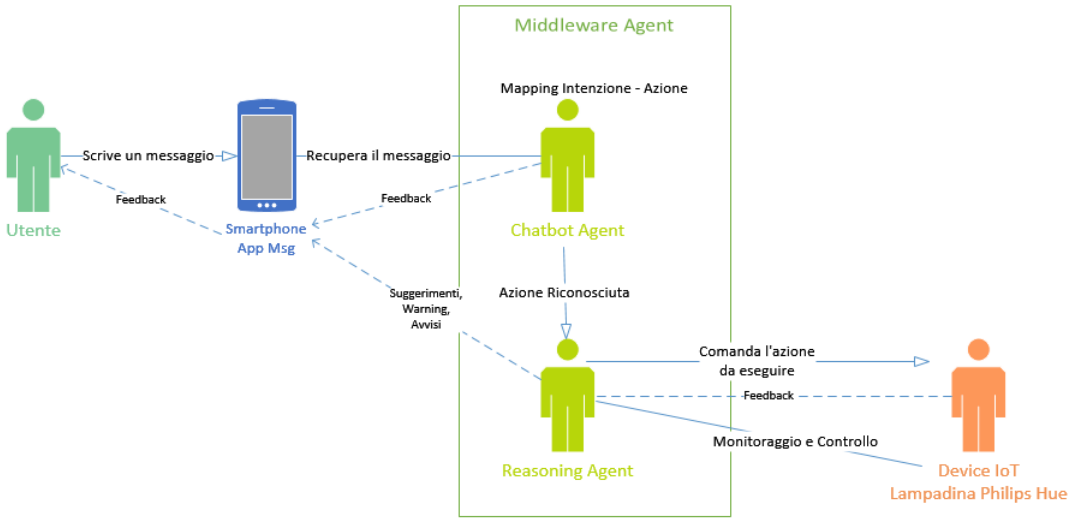


Figura 3.1: Entità Del sistema

Data l'ottima offerta di servizi offerti dalle aziende relativamente allo sviluppo dei chatbot e ai servizi di NLP, si procede utilizzando uno dei framework analizzati nella sezione 1.2.3 . Si è scelto di utilizzare il framework

offerto dalla Microsoft: **Microsoft Bot Framework**. Questo può interfacciarsi con la maggior parte dei canali (come Skype, Facebook, Teams,Slack, Telegram, SMS,...). Inoltre fornisce risultati ottimi in termini di elaborazione del testo e di altre funzionalità, grazie ai servizi cognitivi di Azure, come l'analisi dei sentimenti dal testo, estrazione di significati o il controllo ortografico. Inoltre è completamente open source, disponibile su GitHub. All'interno del framework è utilizzato il bot engine chiamato **LUIS.ai**, punto di partenza per la creazione del bot, responsabile della comprensione di quello che l'utente desidera ed esprime all'interno dell'applicazione di messaggistica

3.2 Interazioni

Le interazioni principali del sistema avvengono fra lo user, tramite l'app di messaggistica, e il chatbot, e fra l'agente middleware/reasoning e gli IoT Device. Per quello che concerne il comportamento e le interazioni interne al middle agent, queste dipende dalla tecnologia utilizzata.
Le soluzioni possono essere diverse. Si possono integrare i due agenti in un unico solo, sfruttando la stratificazione e la modularizzazione a livello di codice. In alternativa si mantengono i concetti architetturali progettando i due agenti separatamente, e l'interazione dipenderà anche in questo caso dalla tecnologia di sviluppo scelta.

3.2.1 User - Agent

Ricordando che l'agente conversazionale, il chatbot, ha il compito di ottenere i dati in input e di mapparli in azioni, la modalità con cui l'interazione avviene tra lo user, tramite l'app di messaggistica, e il chatbot, è strettamente dipendente con la tecnologia scelta.

Uno degli obiettivi principali è quello di massimizzare la semplicità e l'usabilità del sistema da parte dell'utente, permettendo ad esso di accedere e interagire con i device tramite un'applicazione di messaggistica di largo uti-

lizzo, e probabilmentegià presente all'interno dello smartphone. Fra i channel visti in 1.2.1 si è scelto di utilizzare **Telegram** data la sua fama, la sua flessibilità e la sua apertura verso gli sviluppatori.

Il framework della Microsoft supporta questa fase risolvendo internamente la comunicazione e le chiamate API fra il canale di comunicazione (Telegram in questo caso) e l'elaborazione dei messaggi dal bot. Il bot, integrando Luis grazie al framework, riceverà direttamente le intentions e le entities derivate dal bot engine.

3.2.2 Agent - IoT Device

La seconda questione da affrontare è l'interazione con gli IoT Device. Per poter comunicare con questi occorre innanzitutto che siano connessi alla rete internet, ottenendo un indirizzo IP, elemento fondamentale per poter definire l'appartenenza all'Internet of Things. In secondo luogo, i device dovrebbero mettere a disposizioni le proprie funzionalità, tramite delle API, possibilmente con un'unica modalità standard, dato che ogni azienda utilizza protocolli di comunicazione diversa. Alcuni dispositivi già includono questa prerogativa, nella maggioranza dei casi vengono fornite API utilizzando il protocollo HTTP con metodologia RESTful, in remoto o in locale .

La scelta dei device da utilizzare come caso di studio è ricaduta sul sistema di illuminazione **Philips Hue** proprio grazie alla disponibilità delle API per l'interazione con le lampadine.

Generalizzando, occorre considerare un ulteriore livello di astrazione per poter accedere ai device in modo unificato e standard. Questo lo si ottiene integrando il concetto di **Web Of Things**. Seguendo questa filosofia, i dispositivi sono tutti connessi con tecnologie simili e medesime interfacce, tramite comuni URL, per poter essere acceduti e utilizzate in ugual modo

da ogni altro dispositivo, sistema o persona[19], tramite i comuni e standardizzati meccanismi del Web (Http, Json, Socket, ...) . Si raggiunge così un elevato livello di interoperabilità. Con il WoT si pone un livello superiori all'IoT per far sì che i device siano presenti sul web tramite delle API, a partire dal livello più inferiore detto di Access, fino al livello di utilizzo da parte di applicazioni, passando per livelli di accesso alla rete in cui i device possono essere ricercati sul web (semantic web), e un livello detto Share che aggiunge uno strato di accesso sicuro e di filtro per i dati messi a disposizione al web.

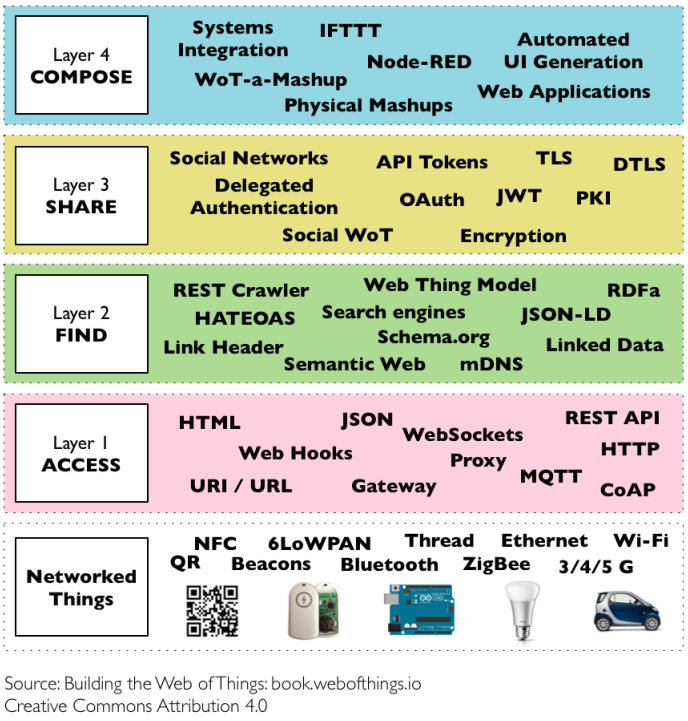


Figura 3.2: Architettura Web Of Things

3.3 Schema Architetturale

Ponendo il focus sull'antropomorfizzazione degli oggetti smart, device IoT connessi alla rete, questi possono essere arricchiti di capacità e di funzionalità con l'integrazione di diversi layer superiori al livello dell'Internet of Things.

Si è già individuato il livello superiore del Web Of Things per il fatto che l'end device non solo comunica attraverso la rete, ma diventa disponibile all'interno di essa tramite API richiamabili con richieste HTTP di tipo Restful, così come qualsiasi altra risorsa del web. Questo permette, a patto di conoscerne un riferimento per il contatto, di invocare e richiedere l'esecuzione delle funzionalità del dispositivo.

Molti prodotti offerti e messi a disposizione dalle aziende non rendono pubbliche e disponibili tutte le funzionalità, per avere un maggior controllo del prodotto. Pertanto emerge la necessità di aggiungere un livello di ragionamento, detto reasoning layer, che tramite le API a disposizione wrappa ancora una volta l'end device in una entità maggiormente intelligente, permettendo comportamenti complessi, personalizzabili, proattivi e autonomi. Si può sfruttare eventualmente una base di dati a cui far riferimento, per memorizzare il contesto e lo stato del sistema in un determinato momento. In questo livello si possono sfruttare architetture di alto livello presenti in letteratura per quanto riguarda il concetto di agenti, date le caratteristiche di autonomia, proattività e elaborazioni complesse. Ad esempio si può seguire l'architettura BDI che introduce meccanismi di alto livello, o tecniche di Machine Learning, di apprendimento con rinfornzo. Questo livello può essere definibile come **Agents for Things**.

Il device rappresentato dall'agente di reasoning dovrà poi interagire con l'utente finale tramite il servizio di messaggistica e includendo quindi gli step di elaborazione del testo, tramite algoritmi di NLP, e di derivazione delle intentions - actions. Questo livello di cui si è già discusso può essere visto come livello superiore a quello degli Agents for Things, in quanto funge da livello applicativo verso l'utente, traducendo i messaggi e facendo da intermediario con i livelli sottostanti. Questo livello è detto **Interfaces for Things**.

Si deriva pertanto una architettura verticale, per livelli, in cui ognuno di

essi sfrutta le funzionalità dei layer immediatamente sovrastanti o sottostanti, cooperando e comunicando con essi tramite lo scambio di dati che può avvenire in diverse modalità. Questo fa sì che il sistema sia maggiormente modularizzato, controllabile e manutenibile. Il livello superiore, Interfaces for Things, può essere visto come un livello applicativo simile a quello presente nel modello ISO/OSI, in cui vengono offerti servizi direttamente alle applicazioni finali utilizzate dagli utenti e fornendo interfacce di interazione fra o user e il destinatario del servizio, il device IoT nel caso in questione.

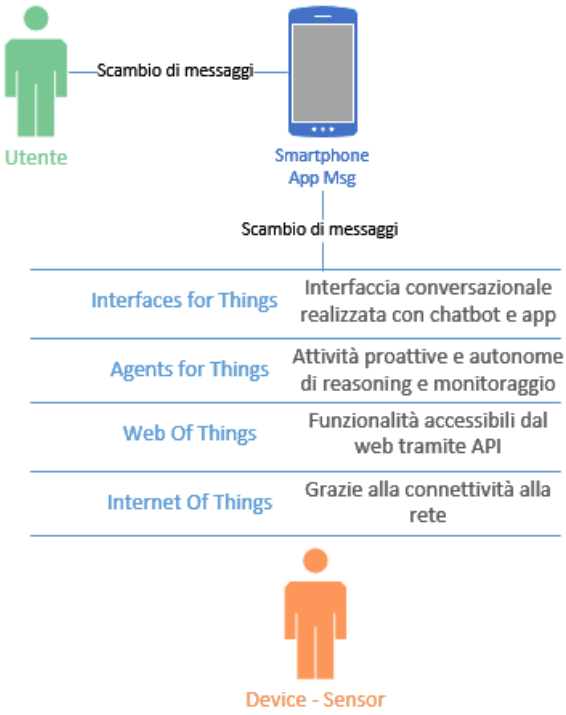


Figura 3.3: Architettura per livelli

Capitolo 4

Tecnologie Abilitanti

4.1 IoT Device

4.1.1 Il kit Philips Hue

> *"Migliora la tua vita quotidiana. Philips Hue ti dà il benvenuto a casa. Può svegliarti, darti energia e farti sentire al sicuro. Migliora il tuo umore. Ti offre un'esperienza di intrattenimento più coinvolgente. Può persino tenerti informato sulle condizioni meteo o le chiamate in arrivo. Una volta che inizierai a scoprirle, le possibilità sono infinite."*
>
> - Offical Philips Hue WebSite ,

Philips Hue è il sistema IoT di illuminazione connessa wireless che consente di controllare con facilità le luci e di creare l'atmosfera giusta per ogni momento [17].

Componenti

- **Bridge**: Hub Intelligente in grado di collegare fino a 50 lampadine Smart del sistema Philips Hue. La configurazione delle connessioni e delle interazioni fra i componenti, e quindi l'inizializzazione della rete, avverrà automaticamente grazie ad esso. Sono attive due tipologie

diverse di connessione: una relativa al router presente all'interno del bridge, connesso alla rete internet, tramite il wi-fi. In questo modo qualsiasi dispositivo che presenta una connessione alla rete domestica,come uno smartphone, un tablet o un pc, può interfacciarsi con le lampadine. La seconda rete di tipo mesh, è creata tramite il protocollo Zigbee, e connette le lampadine al bridge. Questa permette la massima efficienza energetica e il minimo consumo di banda; É infatti una delle tecnologie abilitanti più utilizzata nell'IoT.

- **Lampadina Led**: con un ottimo livello di efficienza energitica, rappresentano gli attuatori del sistema e possono essere di due tipologie: Colorazione RGB, o monocromatiche bianche con luce calda o fredda. Possono quindi emettere diverse gradazioni di luminosità, mantenere la luce costante, ma anche lampeggiare. Sono disponibili in commercio in diverse forme (bulbo, nastro, ecc...). Nel caso di studio in questione sono state utilizzate le lampadine RGB, e corrispondono all'end-device IoT.

- **Application**: L'utente può interagire con un software della Philips scaricabile per un dispositivo mobile, Android, IoS o Windows Phone. Con l'app si possono ottenere tutte le funzionalità messe a disposizione dal sistema. L'utente può anche utilizzare applicativi forniti da terze parti, per mobile, desktop, o qualsiasi altro device.

Figura 4.1: Starter Kit composto dal Bridge e da tre lampadine RGB

Funzionalità

Sono tutte disponibili attraverso l'utilizzo dell'applicazione:

1. Accensione / Spegnimento delle lampadine:

2. Regolazione della luminosità e del colore

3. Organizzazione delle lampadine per stanze: questo permette di controllare le luci direttamente riferendosi alle stanze, e quindi anche a gruppi di lampadine

4. Scene e atmosfera: Si possono creare scene/atmosfere con palette consigliate dalla philips stessa, per simulare ambienti e situazioni come un'alba, avere un maggior livello di concentrazione, la primavera, e così via...

5. Attività Personalizzate: Imposta programmi di illuminazione a orari predefiniti con scene personalizzabili.

6. A casa / Fuori Casa: Si può collegare il servizio di geolocalizzazione in modo tale che non appena si acceda all'abitazione le luci siano accede, stessa cosa per l'uscita e lo spegnimento.

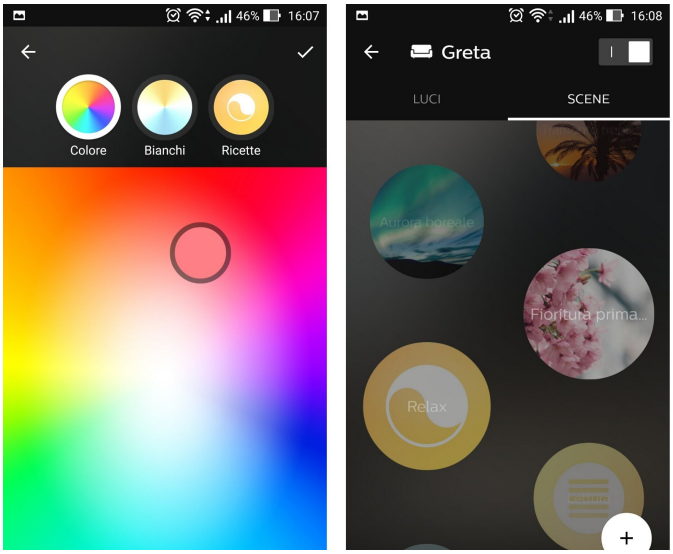


Figura 4.2: Android App - Regolazione del colore e della luminosità e impostazione delle scene

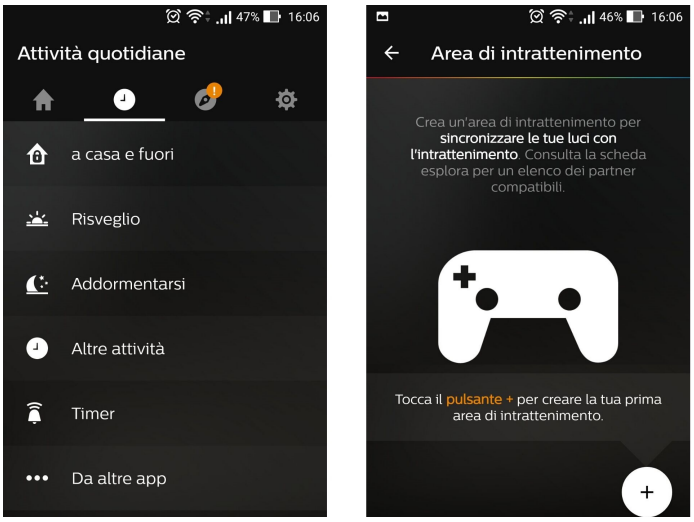


Figura 4.3: Elenco di alcune funzionalità disponibili dall'app

Installazione e Configurazione

Ogni lampadina può essere normalmente utilizzata una volta collegata all'alimentazione con un normale interruttore di accensione/spegnimento.

- Collegamento delle lampadine all'alimentazione
- Collegamento del bridge al router wi-fi; Questo segnalerà la conclusione della configurazione tramite l'attivazione delle tre spie presenti sul bridge.
- Download dell'applicazione Mobile. Il device deve essere anch'esso connesso alla rete domestica, al wi-fi in cui si trova il bridge. In questo modo verrà effettuato il collegamento e il bridge potrà creare la sottorete per comunicare con le lampadine.

Sviluppo - Developer Program

Philips mette a disposizione delle API con la relativa documentazione per far sì che gli sviluppatori possano creare propri progetti, applicazioni mobile e desktop, o per altri device, interfacciandosi con le potenzialità offerte dalle

lampadine smart. Tutto il sistema è rappresentato da un unico IP relativo al bridge. Dato questo si può accede a qualsiasi dispositivo connesso al bridge. Sono disponibili anche SDKs per IOS, OSX, Java multi-platform e Android. Questo rappresenta il livello Web of Things posto superiormente alle lampadine, IoT Devices.

Tutte le API restituiscono una risposta in formato JSON, con codifica UTF8. Si utilizza una POST Request HTTP . Sono eseguibili all'interno del debugger fornito. Si può utilizzare infatti una interfaccia web collegata al bridge e raggiungibile con `http:/IP_BRIDGE/debug/clip.html`, detta Debugger.

Un grosso vincolo da considerare per l'utilizzo delle API emerge dal fatto che le lampadine, collegate grazie al bridge alla rete domestica wi-fi, non possono essere utilizzate direttamente da remoto, ma solo all'interno della stessa rete a cui è connesso il bridge. Occorrerà quindi considerare questo ostacolo e valutarne il superamento. Philips all'interno della documentazione fornisce una prima soluzione che consiste nell'utilizzo dell'interfaccia IFTTT.

- Per poter ottenere l'accesso alle lampadine tramite il bridge ed utilizzare le API, occorre registrarsi alla pagina [18] dedicata agli sviluppatori.

- Occorre ottenere poi l'autorizzazione. Per cui si recuperano l'indirizzo IP e l'ID assegnato al bridge. Philips mette a disposizione un broker server per questo, entrambi ottenibili in formato JSON dalla pagina `https://www.meethue.com/api/nupnp`.

- La richiesta dello username autorizzato tramite una determinata richiesta HTTP è da effettuare immediatamente dopo la pressione del pulsante posto sul bridge che funge da controllo di sicurezza per evitare l'utilizzo delle API da parte di utenti esterni all'ambito domestico

Address	`http://<bridge ip address>/api`
Body	`{"devicetype":"my_hue_app#iphone peter"}`
Method	`POST`

Figura 4.4: Richiesta di uno username autorizzato

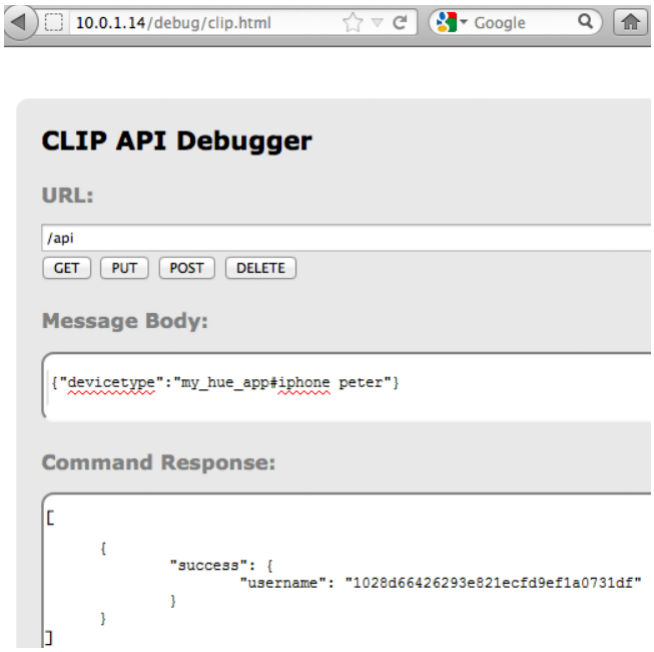


Figura 4.5: Interfaccia web - Debugger

API Principali

Le API messe a disposizione sono chiamate HTTP di tipo RESTful. Sono tutte utilizzabili solamente all'interno della rete domestica attraverso lo username autorizzato ottenuto. Le richieste HTTP sono effettuate all'URL utilizzato in precedenza a cui si aggiungerà la richiesta che si desidera eseguire.

- Lista delle lampadine connesse al bridge: Richiesta GET, con url `http://<bridgeipaddress>/api/<bridgeipusername_ottenuto>/lights`

- Stato di una specifica lampadina, dato il suo ID: Richiesta GET, con url `http://<bridgeipaddress>/api/<bridgeipusername_ottenuto>/lights/ID-LIGHT`

- Accensione/Spegnimento: Richiesta PUT con url `http://<bridgeipaddress>/api/<bridgeipusername_ottenuto>/lights/ID-LIGHT/state`. Attraverso il body della richiesta si imposta lo stato con il valore da modificare. Ad esempio per accendere la lampadina occorrerà inserire nel body {"on":false}.

- Regolazione del colore e della luminosità: viene effettuata aggiungendo i relativi campi al body utilizzato nell'accensione/spegnimento. Ad esempio {"on":true, "sat":254, "bri":254,"hue":10000} in cui si modifica l'intensità con 'sat' (saturazione della luce), la luminosità con 'bri' e il colore con 'hue'. I primi due campi utilizzano un range 0-255 mentre il campo relativo al colore 0-65535.

4.1.2 IFTTT

Acronimo di "If This Than That", si tratta di un servizio on-line gratuito offerto al pubblico tramite un sito web che permette di creare delle coppie condizionali di causa-effetto (azione) con una grande flessibilità. La coppia è composta quindi da due servizi digitali. Quando una determinata causa si verifica allora una azione viene eseguita in completa autonomia, senza che l'utente debba interagire con il sistema. Ad esempio si possono creare rapporti del tipo "Se metto un like ad una canzone su Youtube, questa deve essere aggiunta alla playlist di Spotify" o "Invia una e-mail se domani è prevista pioggia in una certa città". Questo permette agli utenti che eseguono azioni abituali di risparmiare tempo automatizzandole. Si individuano due termini all'interno del servizio: Channels e Applets

- **Channel - Service**: Un servizio che mette a disposizione delle azioni, ad esempio Facebook, Spotify, Gmail, Youtube, Instagram, Evernote,...

- **Applets**: Sono le coppie casua-effetto, gli automatismi scelti dall'utente fra due servizi, channels. La causa può essere detta trigger, in quanto scatena l'evento relativo all'esecuzione di una azione, detta action. La piattaforma ne mette a disposizione alcuni predefiniti.

Philips Hue Channel

Una volta registrati alla piattaforma si può effettuare il collegamento al bridge della Philips Hue in modo tale da poter superare l'ostacolo dell'inaccessibilità delle API da remoto, cioè da una rete diversa dal wi-fi domestico. Esiste una pagina dedicata, `https://ifttt.com/hue`, da cui si può effettuare il collegamento con le lampadine passando per una fase di autorizzazione consistente nella pressione del pulsante sul bridge.

Figura 4.6: Creazione una Applet - IFTTT

Actions Disponibili

Quasi tutte le funzionalità offerte dal sistema della Philips sono utilizzabili da IFTTT. Ognuna di queste può essere applicata a tutte le luci, di un gruppo, a di una specifica lampadina:

- Accensione/spegnimento
- Applicazione di una scena ad una stanza
- Blink, lampeggio
- Variazione della luminosità di un certo valore
- Cambio di colore, dato uno scelto dall'utente, o randomicamente

4.2 Chatbot

4.2.1 Telegram Channel

La creazione di un chatbot all'interno di Telegram è guidata da un chatbot detto BotFather. Si ottiene da questo un account simile a quello degli utenti normali, fatta eccezione per l'assenza del numero di telefono collegato e delle informazioni sullo stato online e della data di ultimo accesso. Con questo account si può iniziare una normale conversazione. Una importante nota da tenere presente è che normalmente i bot creati tramite le API di Telegram non avviano conversazioni spontaneamente con gli utenti ma reagiscono al loro avvio della comunicaizone.

I passi della creazione sono relativamente semplici in quanto basta avviare con esso una conversazione e digitare i comandi `/start` per ottenere la lista di comandi disponibili, fra cui compare `/newbot` che permette appunto la creazione di un proprio bot, al seguito di un botta e risposta con il BotFather, per il nome e altre informazioni. Per l'attivazione del bot creato si utilizzerà il token ricevuto come messaggio in chat. Al momento non è disponibile creare un bot interagendo con il BotFather programmaticamente, ma solo tramite l'app di Telegram.

Una volta creato il bot, si possono utilizzare testi ed elementi grafici per interagire con l'utente, come pulsanti e menu, possibilità di pagamenti, immagini.

In particolare, i diversi tipi di input forniti dall'utente vengono trasmessi con richieste HTTP ai software presenti nei server di Telegram che fungono da intermediari per interfacciarsi con le API di Telegram.

4.3 Framework

4.3.1 Bot Engine - LUIS.ai

Questo servizio offerto dal framework Microsoft Bot utilizza gli algoritmi di Machine Learning con apprendimento per rinforzo per derivare da delle porzioni di testo elementi rilevanti a seconda del contesto. Supporta 12 lingue, compreso l'italiano ed è completamente gratuito per le prime 10mila transazioni al mese. In particolare, a partire dall'input degli utenti si è in grado di derivare quello che desiderano esprimere, le intentions. Si interfaccia quindi con applicazioni client ricevendo da esse l'input testuale e fornendo come risultato le intenzioni e altre informazioni derivate.

Occorre creare un'app LUIS che consiste in un modello sviluppato per un dominio specifico a partire da una lista di semplici intenzioni. Queste saranno individuate dall'engine attraverso gli algoritmi di machine learning a partire da una serie di frasi, dette utterances, inserite dallo sviluppatore come esempio, come training per l'app. La fase di addestramento tramite le utterance è fondamentale per migliorare la qualità dei risultlati ottenuti dal servizio. Una volta avviato il modello, questo comncia una fase di apprendimento attivo, mettendo a disposizione dello sviluppatore tutte le nuove utterance, frasi, in input dagli utenti in modo da poter confermare o modificare i match individuati dall'engine.

Luis prende in input le utterances e le scompone in parole, dette token, componente più piccola etichettabile come entità. All'interno delle utterances saranno inoltre individuate, ed etichettate dallo sviluppatore, delle entità principali, informazioni rilevanti, dette entities. Sono disponibili infine entità e modelli pre-costruiti per velocizzare il processo di sviluppo. Questi modelli raggruppano entità di una stessa sfera concettuale e sono disponibili solo per alcuni paesi. Esempi di domini a disposizione sono Fitness, Eventi, o Musica. Modelli di dominio predefiniti non sono attualmente disponibili nella lingua italiana. Una volta pubblicata l'app si ottiene un link di endpoint. Tramite questo l'app riceverà l'input testuale dell'utente da un'altra

entità client attraverso richieste HTTP e risponderà con le intenzioni derivate attraverso il formato JSON.

Sviluppare LUIS App

Per prima cosa occorre individuare gli intents e le entities che corrispondono alle definizioni riportate nella sezione 1.2.1.

Ricapitolando, dal punto di vista dello sviluppatore, gli Intents possono essere visti come le funzioni, metodi, da eseguire a fronte di un certo input dell'utente. Queste chiamate a funzioni potrebbero avere dei parametri aggiuntivi, dei valori, che corrispondono ai token, parole, etichettate come entities.

Una volta pianificati e inseriti gli elementi principali all'intenro dell'applicazione, si prosegue con l'addestramento dell'app tramite delle utterance di esempio, tutto possibile attraverso una semplice interfaccia web messa a disposizione dalla Microsoft. Il processo di addestramento consiste nell'inserimento di frasi che ricalcano i possibili inputs dell'utente, e l'etichettamento delle parole in base alle entità create, raggruppate per gli intents.

Quando si pubblica il modello Luis si può attivare l'opzione per abilitare l'utilizzo di Bing Spell Checker per correggere automaticamente gli errori di ortografia dell'utente prima che LUIS processi il testo. Pubblicando l'app si ottiene un URL endpoint nel formato `https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/xxx?subscription-key=yyy&timezoneOffset=0&verbose=true&q=` che si può utilizzare direttamente dal broswer per una fase di test inserendo in coda all'indirizzo la frase ipotetica di input di un utente e ottenendo il JSON di risposta con gli scores per ogni elemento riconosciuto da LUIS.

4.3.2 Microsoft Bot Framework

Con questo si è in grado di costruire, connettere, testare, installare e gestire un chatbot. Una volta creato questo, si può modificare tramite le interfacce messe a disposizione, o programmaticamente attraverso un IDE come Visual Studio. Il Bot Builder inoltre fornisce il supporto sia a Node.js sia a .NET. Lo sviluppo può essere ulteriormente facilitato grazie alla disponibilità di modelli e di bot predefiniti da usare come buona base per creare i propri bot; Anche la fase di testing è supportata grazie al Bot Framework Emulator e a un Channel Inspector con cui si può verificare la correttezza del chatbot in diversi canali, dato che alcuni elementi, in particolare quelli visuali, potrebbero non essere supportati e convertiti in testo in alcuni canali [3]. Microsoft inoltre fornisce la possibilità di utilizzare HTTPS e l'autenticazione con il Bot Framework per assicurare che l'endopoint del bot sia utilizzato solo ed esclusivamente dal Bot Framework Connector, registrando il bot tramite il suo appID e password. Il connector ha la funzione di fornire una singola API di tipo Rest per la comunicazione e il collegamento del bot attraverso i canali di messaggistica (come Telegram).

Sviluppare Microsoft Bot App

Dalla pagine dei servizi azure, così come la pagina dedicata ai bots `https://dev.botframework.com/bots`, si può attivare la procedura di creazione di un agente conversazionale. Le modalità di creazione variano in base alle esigenze. Per sfruttare l'esecuzione del bot sui server di Azure si può attivare la risorsa Web App Bot. I dati richiesti per la creazione della risorse sono il Bot Name che sarà visualizzato nei canali, la sottoscrizione, la location geografica, il tariffario, l'app name che corrisponde all'URL univoco per il bot, ad esempio `http://myawesomebot.azurewebsite` . Infine occorre scegliere un template, con C# e node.js in cui Microsoft prepari un bot base con possibilità di modifiche successive dal compilatore presente nell'ide o integrando il progetto ad esempio in Visual Studio. Nel progetto si è utilizzato il linguaggio C# con template LUIS, per cui si può collegare un endpoint HTTP

che interpreta l'input in termini di intenzione e di entità; In automatico con un template di questo tipo si crea una LUIS App vuota, modificabile da `https://www.luis.ai/`, o il corrispondente url europeo. Una volta creato il bot si può procedere con lo sviluppo tramite un IDE e impostando l'opzione continuous deployment con cui si può sincronizzare il progetto pubblicato su una piattaforma come GitLab o GitHub.

Integrazione Luis - Web App Bot

Per l'integrazione dei due servizi tramite azure e l'interfaccia web messa a disposizione, occorre modificare i parametri presenti alla sezione Application Setting, in particolare: LuisAppId, LuisAPIKey, LuisApiHostName. Successivamente a questo occorre modificare il bot a livello di implementazione, usando l'IDE che si desidera e inserendo i medesimi dati nel file di configurazione. Per compilare il bot con le modifiche, passo necessario per poterlo testare successivamente, occorre eseguire il file build.cmd che si occupa dell'inclusione e dell'installazione dei riferimenti interni all'app.

Capitolo 5

Progettazione

5.1 Valutazione Rischi

Questo capitolo affronta dapprima i punti critici e le problematiche individuate, in quanto alcune di queste possono influire sulla progettazione.

Si procede con l'analisi e la progettazione per livelli, a partire da quello superiore l'Interfaces for Things. Questo per mantenere il più possibile la separazione fra i livelli e la modularità.

- **Descrizione:** Occorre considerare che ogni singolo utente comunicherà con i propri dispositivi. Pertanto l'opzione di utilizzare un unico bot per tutti gli utenti non è considerabile in quanto si creerebbe un collo di bottiglia per le molte richieste e interazioni. Quindi si considera la generazione di un bot per ogni user e per ogni sistema. Questo nella maggiorparte dei casi non può essere effettuato da API col fine di evitare del carico di lavoro all'utente. Solitamente la generazione di un bot avviene tramite l'app di messaggistica, in questo caso Telegram. Questo può mirare ai requisiti non funzionali di usabilità e di semplicità nella configurazione iniziale per l'utente.
  Soluzione: Si è pensato che con l'acquisto delle lampadine e del sistema all'utente sarà fornito l'ID del bot da contattare su telegram che può essere inizialmente il codice a barre del prodotto, in modo da

evitare la creazione e l'interazione con il BotFather e la gestione del token.

- **Descrizione:** L'accesso alle API locali richiede alcuni parametri e alcuni account registrati ai portali utenti e sviluppatori. Inoltre è necessaria la pressione del pulsante sul bridge. Questo può mirare ai requisiti non funzionali di usabilità e di semplicità nella configurazione iniziale per l'utente.
  Soluzione: Si è pensato di utilizzare degli account sviluppatori predefinito ed effettuare le configurazioni utilizzando le API locali, richiedendo allo user, la prima volta che si accede al sistema tramite la chat, di effettuare il click del pulsante. In questo modo si riduce al minimo la configurazione da parte dell'utente.

- **Descrizione:** L'accesso alle API messe a disposizione dalla Philips non è disponibile da remoto
  Soluzione: La Philips mette a disposizione la possibilità di richiedere l'accesso alle API da remoto. Questa è stata effettuata ma ancora in attesa di risposta. E' necessaria pertanto l'integrazione del servizio web IFTTT.

- **Descrizione:** L'utilizzo conseguente del servizio IFTTT è vincolante in termini di funzionalità e per il soddisfacimento di alcuni requisiti non funzionali per l'interazione con il sistema Philips Hue.
  Soluzione: Dare precedenza all'utilizzo delle API locali laddove possibile. Cercare di sfruttare le poche features fornite per comporre comportamenti più complessi.

- **Descrizione:** Con l'integrazione di IFTTT non è parametrizzabile l'ID della lampadina
  Soluzione: Occorre creare una applet per ogni funzionalità e per ogni lampadina differente.

- **Descrizione:** Con IFTTT non è possibile ottenere informazioni riguardanti lo stato di una o più lampadine. Ad esempio ottenere informazioni come lo stato di accensione/spegnimento, il colore attualmente usato, il livello di luminosità, la scena impostata.
  Soluzione: Utilizzare uno stato interno per tenere traccia delle modifiche.
  Limitazioni: Se l'utente utilizza nel frattempo un'altra app si può ottenere un disallineamento delle informazioni. Pertanto l'utente deve essere avvertito di questa casistica e occorre fornire l'opzione di allineamento con un "reset" del sistema (luce bianca e spegnimento).

- **Descrizione:** Le Applets create con IFTTT non sono pubblicabili con un normale account utente. Questo implica che ogni singolo utente utilizzatore del sistema dovrà creare tutte le applets necessarie per l'esecuzione delle funzionalità. Questo mira all'usabilità e alla semplicità in termini di configurazione inziale.
  Soluzione: Supportare il processo di configurazione tramite immagini e spiegazioni step by step con verifiche incrementali del servizio

5.2 Interfaces for Things

Si parte con l'analisi degli scenari di utilizzo del sistema individuati nella sezione 1. Le azioni come l'accensione e lo spegnimento delle lampadine sono immediate e dovranno gestire internamente i casi in cui le azioni siano già attive da un momento precedente.

Le altre azioni come il cambio di colorazione e di luminosità implicano invece che la lampadina sia accesa e la loro esecuzione ne potrebbe comportare la conseguente accensione. Occorre quindi gestire una interazione aggiuntiva con l'utente inizializzata dal bot. Questo dovrà chiedere conferma all'utente se per eseguire la sua richiesta può procedere con l'accensione della lampadina o meno. Ne deriva che oltre agli intents corrispondenti a questi casi d'uso occorrerà che il bot engine Luis individui le risposte alle domande poste dal

Bot che corrisponderanno a una accettazione o meno nei confronti della domanda. E' quindi necessario aggiungere al modello un intent di risposta da parte dell'utente che individua una accettazione o un rifiuto. Si crea un SimpleAnswer intent collegato alla Entity Confirm o alla entity Reject.

Luis mette a disposizione diverse tipologie di Entities fra cui Simple, List, Hierarchical e Composite, a seconda del peso che si vuole associare al valore della entity rilevata. Per quello che concerne la Confirm e la Reject non sarà necessario nè importante conoscere quale termine è stato inserito, come può essere ad esempio un "sì" piuttosto che un "perfetto", quindi si assume la tipologia di entità Simple.

Una domanda aggiuntiva che potrebbe sorgere da parte del bot potrebbe essere relativa alla scelta del colore qualora questo non sia stato specificato. Per il primo prototipo del sistema, questa casistica è gestita selezionando un colore casuale da parte del bot. Quindi l'intent ChangeColor potrà avere opzionalmente il collegamento a una Entity di tipo Color, corrispondente al nome del colore italiano, o HexColor, in quanto l'utente può inserire direttamente il colore desiderato in formato esadecimale. Per quanto riguarda i valori di Color occorrerà mantenere una diversificazione concettuale fra un valore e un altro, specificando anche quali colori il sistema può o meno interpretare. Pertanto si utilizza la tipologia messa a disposizione da Luis, detta List che rappresenta un insieme fisso di valori per i quali ognuno di essi può avere collegata una lista di sinonimi. Essendo questi fissi non saranno aggiornati dal bot engine tramite apprendimento, ma manualmente.

Per aumentare l'usabilità e l'attrattività del sistema si integra una componente di saluti individuata dall'intents Greetings. Rilevata l'intenzione dell'utente di salutare il sistema o nello specifico una certa lampadina, si considera questo equivalente all'avvio di una conversazione. Di conseguenza il sistema risponderà al saluto. In questo modo si può integrare immedia-

tamente l'invito ad avviare la configurazione iniziale qualora non sia stata mai avviata. L'utente potrà poi, in caso di perplessità sull'utilizzo, richiedere aiuti così che il bot gli possa fornire le informazioni necessarie all'utilizzo della chat.

Primo Avvio

Occorre valutare come affrontare la prima configurazione del sistema. L'utente avvierà in un determinato momento, successivo all'acquisto, la chat utilizzando il nome utente del chatbot, considerando il codice a barre, univoco, e avviandolo tramite il comando `/start` o il pulsante di Telegram avvia. Da qui l'utente può salutare il sistema ottenendo un messaggio che lo avverte che deve effettuare un reset, una prima configurazione. Il suggerimento al reset è fornito a fronte di altre azioni richieste, come l'accensione della luce o il cambio di colorazione, laddove necessario.

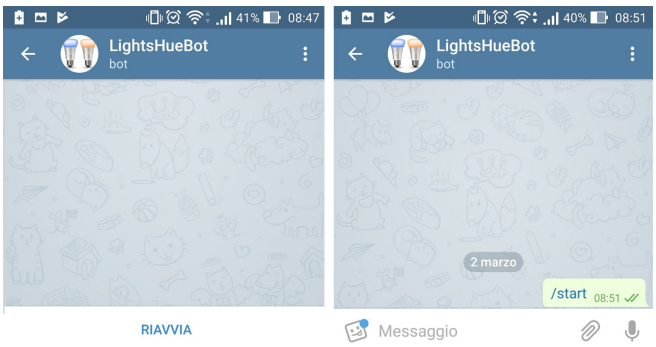


Figura 5.1: Le due modalità di Avvio del bot su Telegram

Luis Model

Partendo dall'elaborazione dell'input dell'utente in intenzioni, è necessario progettare il modello da usare per il bot engine Luis.ai. In particolare occorre individuare quelli che sono gli intents, le actions e le entities necessari per lo sviluppo del sistema e per ottenere il mapping successivo con i requisiti

funzionali del sistema.

Intent	**Descrizione**	**Utterance d'esempio**	**Req. Funz.**	**Entities Collegate**
TurnOn	Rappresenta la richiesta di accensione di una lampadina	"Accendi la luce", "Accendi", "On", "Illumina la stanza"	RF-1	Light
TurnOff	Rappresenta la richiesta di spegnimento di una lampadina	"Spegni", "Off", "Luce spenta"	RF-1	Light
GetStatus	Richiesta di informazioni riguardanti lo stato, come il colore e on/off	"La luce è accesa", "Di che colore è la luce?","Informazioni sull'illuminazione"	RF-2	Light
Increase Brightness	Aumento della luminosità di una o più lampadine	"Aumenta la luce", "Illumina di più", "Luce più forte"	RF-3	Light
Decrese Brightness	Riduzione della luminosità	"Luce più scura", "Fai meno luce", "Ho bisogno di una illuminazione minore"	RF-3	Light

Intent	**Descrizione**	**Utterance d'esempio**	**Req. Funz.**	**Entities Collegate**
Change Color	Richiesta di cambio colore. Accetta il parametro relativo al colore, per identificarlo. Nel caso di assenza sarà selezionato uno casuale.	"Cambia il colore in rosso" , "Verde", "Voglio una luce gialla", "Cambia in #fff000"	RF-3	Hex Color, Color, Light
Simple Answer	Accettazione o rifiuto di una proposta o domanda da parte del bot	"Sì", "Va bene", "No","Perfetto"		Reject, Confirm
None, Help	Richiesta di aiuto o un input non riconosciuto.	"Aiuto", "come funziona" , "aaaa"		
Reset	Il comando di avvio della configurazione iniziale del sistema	"Reset","start", "Primo avvio", "Configura"		

Tabella 5.1: Intents

Entity	**Tipo**	**Descrizione**	**Esempi**
Light	Simple	Rappresenta il riferimento alla lampadina	"Luce", "Lampadina", "Lampada"
Color	List	Sono stati inseriti i nomi italiani dei colori testati e verificati come funzionanti per le lampadine. Dovranno essere mappati in lingua inglese per essere compresi dal sistema Philips.	"Rosso", "Verde", "Blu"
Hex Color	Simple	La forma esadecimale di un colore	"#000000", "#fff000"
Confirm	Simple	Corrisponde all'accettazione/ conferma	"Sì", "Certo", "Ok", "Yes", "Ottimo"
Reject	Simple	Rappresenta il rifiuto/ negazione	"No", "Non va bene", "Non mi piace"

Tabella 5.2: Entities

5.3 Agents for Things

Tramite l'utilizzo del framework della Microsoft saranno progettati i due agenti che compongono i due successivi livelli di astrazione dell'end device.

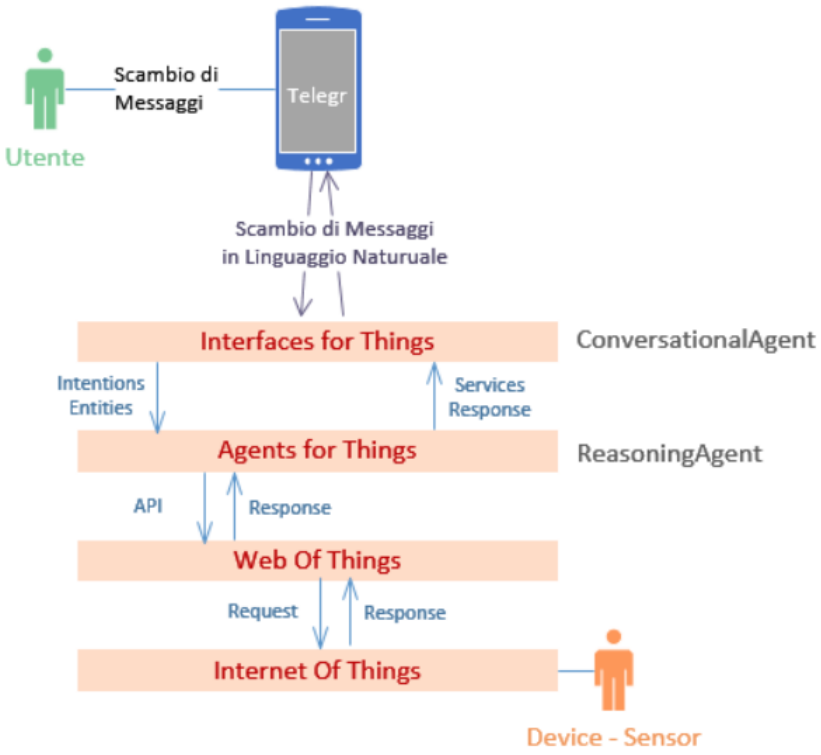


Figura 5.2: Interazioni Fra I Livelli dell'Architettura

5.3.1 Conversational Agent

L'agente conversazionale otterrà dal Bot Engine Luis gli intents e le entities derivati dal testo inserito dall'utente. Questo effettuerà una gestione semplice del dialogo e nella maggiorparte dei casi attualmente da gestire passerà i risultati, in termini di intents e di entities, all'agente reasoner che dialogherà direttamente con l'end device.

Due delle intenzioni individuate non richiedono l'interazione con l'end-device per l'esecuzione di azioni. Per quanto riguarda i Greetings, la gestione dei saluti, consiste in un semplice botta e risposta fra l'utente e il bot. Quello che l'agente dovrà fare è rispondere al saluto riconosciuto dal bot engine Luis, salutando a sua volta. Un'altra risoluzione banalmente gestita dal ConversationalAgent è relativa alla richiesta di aiuto sull'utilizzo del sistema, o per quanto riguarda comandi / input non riconosciuti dal bot engine. In entrambi i casi il sistema risponderà con una lista delle azioni permesse con esempi di utilizzo. Si possono sfruttare in questo caso componenti visuali per facilitare l'interazione iniziale con l'utente.
In entrambi i casi occorre però avvertire l'utente qualora le azioni e operazioni richiedibili non siano disponibili per la necessità di effettuare una prima configurazione. Si utilizzerà il servizio isResetRequired fornito dal livello sottostante, cioè dall'agente reasoner.

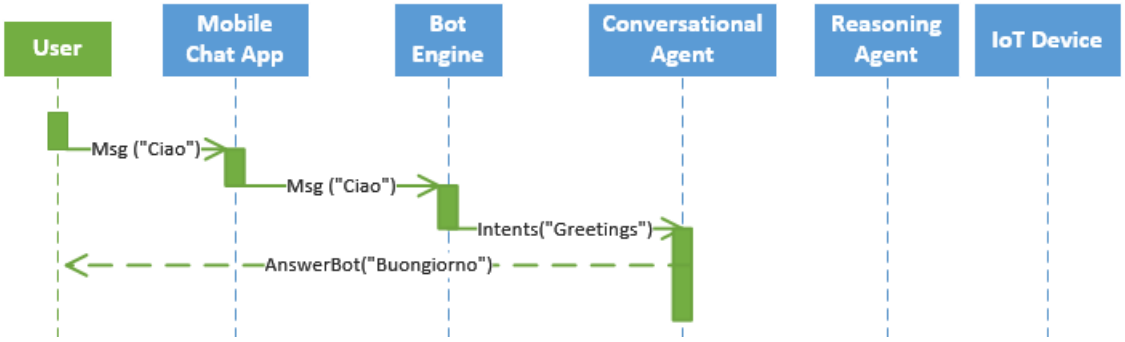


Figura 5.3: Diagramma di sequenza relativo all'intent Greetings

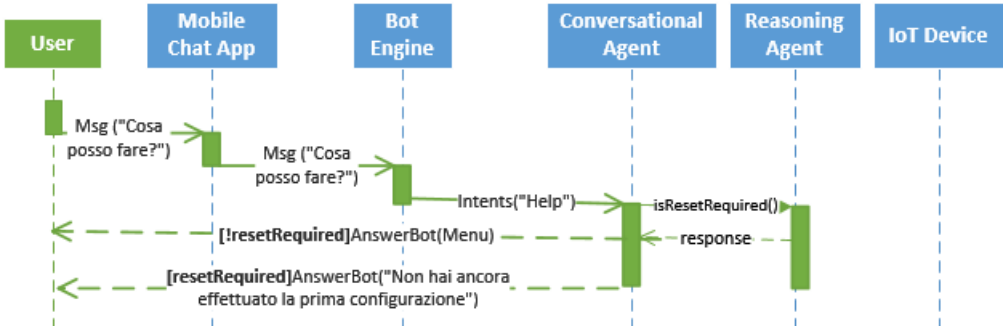


Figura 5.4: Diagramma di sequenza relativo all'intent Help

Per quanto riguarda le altre interazioni con l'utente, le intenzioni derivate insieme alle entities saranno inoltrate al reasoner da cui si riceveranno i risultati delle elaborazioni in modo da fornire o inoltrare dei feedback all'utente su quello che il sistema è riuscito o meno ad eseguire.

5.3.2 Reasoning Agent

Questo agente appartiene al livello dell'architettura detto Agents for Things per il fatto che incarna caratteristiche di autonomia, proattività, capacità di comunicare con gli altri agenti ed entità del sistema. Si occupa della gestione diretta con il layer Web Of Things e quindi si occupa dell'invio dei comandi, delle azioni, mappate tramite API e messe a disposizione dal livello sottostante. Inoltre gestisce attività proattive più complesse di ragioramento per controllo e monitoraggio di problemi e per l'invio di suggerimenti all'utente. Per quanto riguarda le azioni TurnOn, TurnOff, ChangeColor, Decrease/IncreaserBrisghtness, GestSatus, queste termineranno con l'esecuzione di chiamate HTTP alle API messe a disposizione dal sistema della Philips, rispettivamente delle PUT per la modifica di valori e GET per quanto riguarda la richiesta di informazioni circa lo stato del sistema.

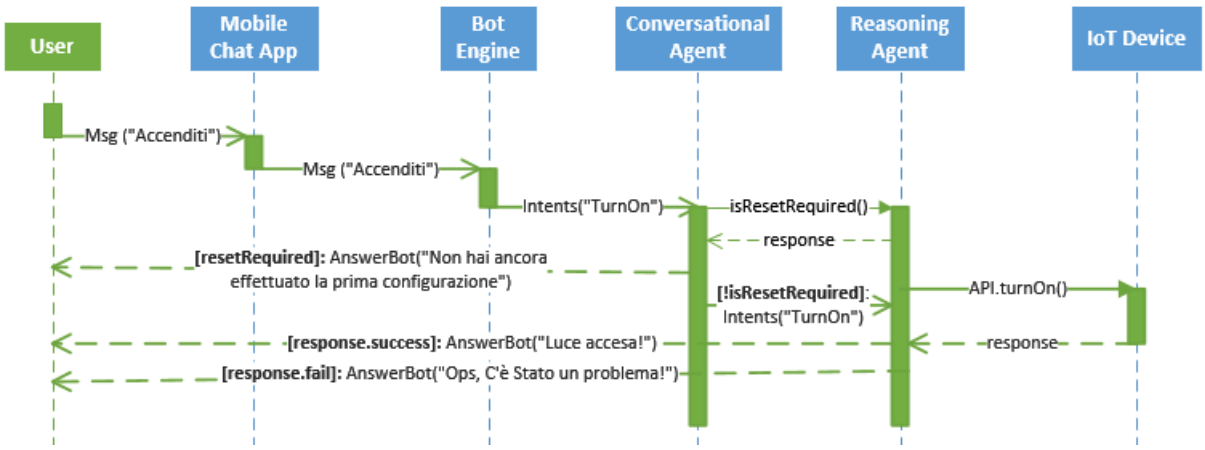


Figura 5.5: Diagramma di sequenza relativo all'intent TurnOn in locale

Il discorso appena effettuato vale però solamente in locale. Si è visto infatti come da remoto non sia possibile accedere alle API direttamente, passando quindi per il servizio di IFTTT da cui occorrerà effettuare delle chiamate GET al WebHook appositamente creato per le applets.

Occorre quindi considerare che al termine delle valutazioni e delle fasi di reasoning per ogni intents, l'agente richiederà l'esecuzione eventuale di un comando/azione al device tramite le API locali, se disponibili, con priorità assoluta, o tramite il servizio IFTTT.

Le intenzioni di accensione e spegnimento sono le più semplici. Queste non comportano interazioni aggiuntive con l'utente, neppure qualora sia richiesta l'accensione di una lampadina già accesa. Entrambe quindi comportano la richiesta all'end-device di esecuzione del comando corrispondente, tramite il successivo layer WoT.

Richieste in sospeso

Il cambio di colorazione e del livello di luminosità hanno invece come precondizione che la lampadina sia stata accesa in precedenza, con riferimento agli scenari visti nella sezione 3. Qualora questa sia spenta occorre effettuare una interazione con l'utente per richiedere la conferma, l'autorizzazione, ad accendere la lampadina per modificarne quindi il colore o la luminosità. All'interno del dialogo l'agente dovrà memorizzare la richiesta in sospeso in attesa di una risposta.

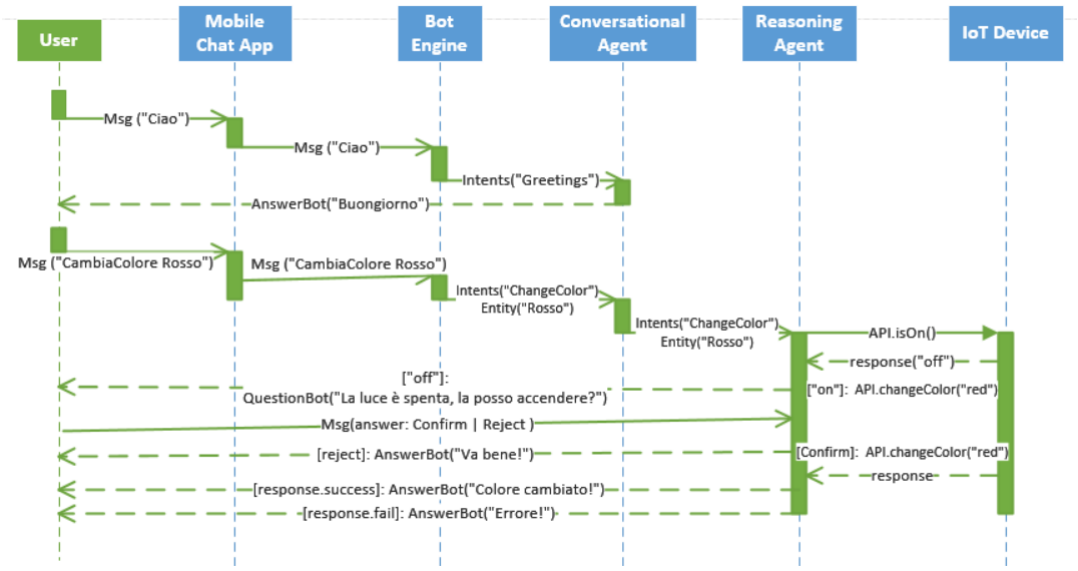


Figura 5.6: Diagramma di Sequenza del cambio colorazione, con il sistema già configurato

Qui occorre inoltre scegliere come effettuare la gestione di un cambio di domanda, o di una diversa risposta.

Data l'assenza di un flusso del dialogo articolato, con richieste da parte dell'utente con diversi parametri, si è scelto di procedere nel seguente modo: le richieste e le interazioni sono mantenute in sospeso per una successiva risposta da parte dell'utente e successivamente cancellate per dare priorità ai nuovi input dell'utente.

Per gestire questo mantenendo un livello di generalità tale da permettere una gestione di interazioni più complesse, si memorizzano le richieste, e quindi gli intents con le connesse entities, per ordine cronologico, facendo in modo che vengano di volta in volta analizzate dal bot le ultime. Le ultime richieste arrivate avranno la precedenza. Da questo ne è derivata la risoluzione tramite una pila, stack, che conterrà quindi tutte le richieste.

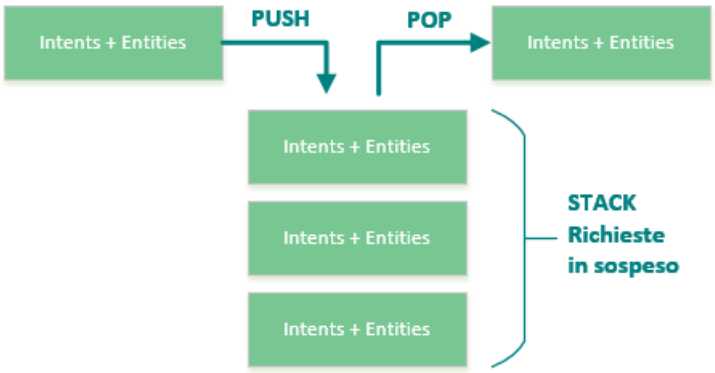


Figura 5.7: Gestione delle richieste in sospeso

Comportamenti autonomi

Il livello di autonomia e di proattività si individua nella gestione e nel monitoraggio dello stato delle lampadine per eventuali warnings ed errori, nella gestione dei suggerimenti e dei consigli da inviare all'utente e di altre funzionalità in background utili al sistema come gli aggiornamenti firmware dei device IoT.

Per quanto riguarda la gestione degli errori e dei **problemi**, questi emergono in particolare al seguito di ogni interazione dell'agente reasoning con l'end-device, tramite le chiamate alle API disponibili dal livello del WoT. Studiando le API queste restituiscono dei codici di errore specifici per il differente problema individuato. Inoltre nel caso in cui l'interazione avvenga nella stessa rete, in locale, si possono utilizzare tutti i parametri specifici messi a disposizione dal bridge per la comunicazione e lo stato delle lampadine. Ad esempio il campo "reachable" presente per ogni lampadina e indicante la raggiungibilità di essa, qualora assuma valore negativo, questo è interpretabile come un guasto o come la semplice disconnessione di questa dall'alimentazione.

Giornalmente, una volta che l'utente inizia l'utilizzazione e l'interazione con le lampadine, si effettua un immediato controllo relativo a questi valori

per notificare all'utente la presenza di problemi di raggiungibilità delle lampadine.

Un comportamento proattivo lo si individua inoltre nella gestione degli **aggiornamenti** del bridge e delle lampadine. Periodicamente, giornalmente o settimanalmente, il sistema se connesso alla stessa rete del bridge controlla la disponibilità di updates. Qualora siano presenti, il sistema attualmente prende l'iniziativa e decide di installarli in completa autonomia, avvertendo l'utente. Questo potrebbe comportare il momentaneo spegnimento del bridge, pertanto si potrebbe integrare in un secondo momento una ulteriore interazione con l'utente per richiedere l'autorizzazione a proseguire con l'istallazione degli aggiornamenti. Si potrebbe integrare con la posticipazione dell'aggiornamento a un momento preciso successivo stabilito dall'utente, cosa che richiederebbe la modellazione di diverse entità aggiuntive come la data, l'ora e il giorno.

L'agente reasoning ha il compito di effettuare un'analisi dell'utilizzo del sistema e dello stato, al fine di fornire **suggerimenti** all'utente. Questa ulteriore interazione nasce completamente per iniziativa dell'agente stesso. Come prototipo iniziale si integrano i controlli sull'utilizzo della lampadina accesa. Se dopo diverse ore questa rimane nello stato di accensione si può suggerire all'utente di spegnerla nel caso non gli serva. O ancora qualora l'utente utilizzi sempre, per diverse ore o per utilizzi successivi, lo stesso colore, si può suggerire all'utente di utilizzare, se disponibili, le diverse colorazioni e luminosità fruibili dal sistema della Philips. Anche la scelta dei colori è personalizzabile e può dar vita a suggerimenti. Ad esempio qualora l'utente selezioni un colore forte come il rosso, si può inserire qualche frase di curiosità relativa a quella colorazione.

5.4 Web of Things

Questo layer coinvolge le interazioni fra l'agente reasoner e il device IoT. Nel caso in questione le modalità di accesso e di comunicazione con le lampadine, e di conseguenza col bridge, dipendono dalla tipologia di connessione. Occorre distinguere infatti i due casi in cui l'utente è connesso alla medesima rete del bridge o a una diversa. La distinzione equivale a chiarire se le chiamate vengono effetuate in locale o in remoto. Questo è necessario a causa della limitazione imposta dalla Philips. In remoto occorre effettuare una interazione con il servizio intermediario IFTTT che si pone come livello superiore alle API. Questo servizio è limitativo in termini di funzionalità perché costringe l'utilizzo delle applets messe a disposizione dalla piattaforma perdendo interazioni importanti come il controllo dello stato e il recupero delle informazioni delle lampadine, o il monitoraggio relativo agli aggiornamenti del bridge. La configurazione delle applets inoltre occorre che sia effettuata prima di qualsiasi utilizzo del sistema e divienta quindi necessario modificare ulteriormente il tutorial e l'assistenza per l'utente in fase iniziale.

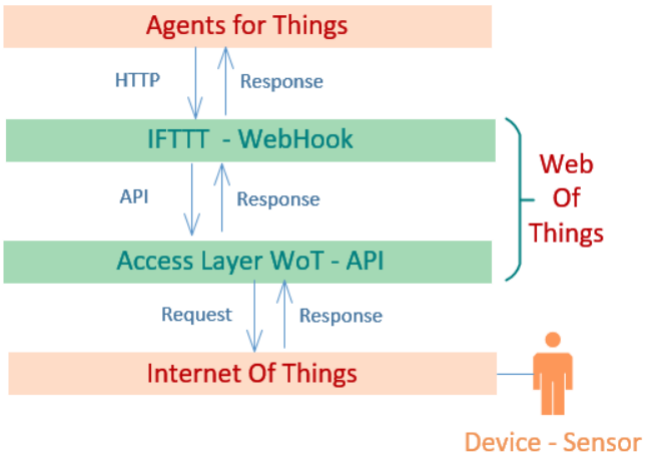


Figura 5.8: Integrazione del servizio IFTTT nell'architettura

Per mantenere il più possibile integro l'uso del servizio IFTTT si considera uno stato che dovrà integrare l'accensione/spegnimento e la colorazione di una singola lampadina. Questo non è necessario con l'uso delle API locali

in quanto per evitare disallineamenti con l'utilizzo di altre app da parte dell'utente, si effettuano controlli dello stato del sistema realmente fornito dalle lampadine.

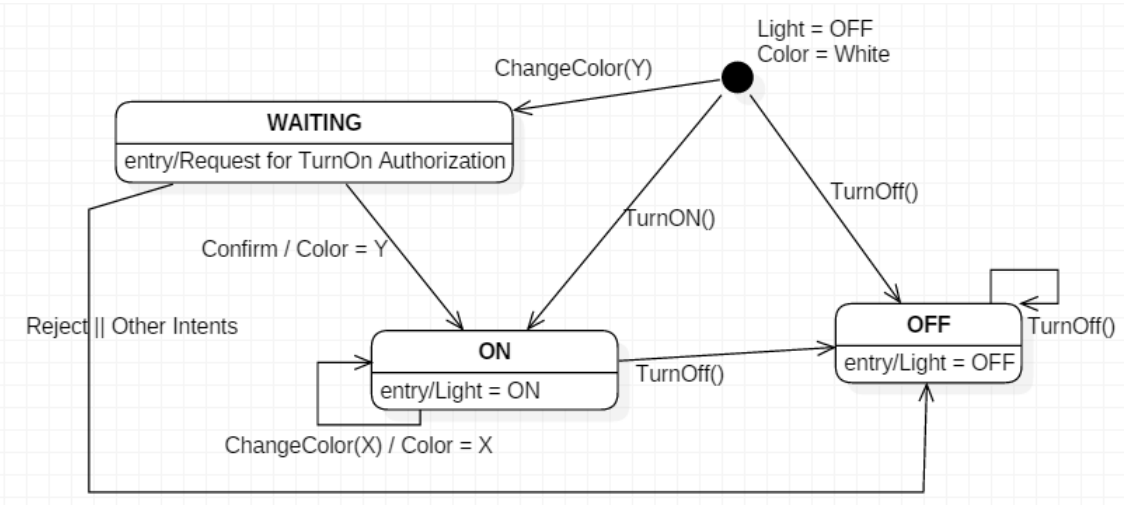


Figura 5.9: Diagramma degli stati utilizzato con IFTTT

Capitolo 6

Sviluppo

Anche per la fase di sviluppo si è proceduto seguendo l'architettura individuata nella progettazione, a partire dal livello superiore corrispondente all'Interfaces for Things, responsabile diretto dell'interazione con l'utente e l'applicazione di messaggistica.

In questa prima fase si è implementato e testato il modello presentato nella sezione 5.2 tramite i servizi offerti da `https://eu.luis.ai/`. All'interno si è effettuata anche una prima fase di training e di testing del modello sviluppato nel bot engine.

6.1 Agents for Things

In questo layer sono presenti i due agenti sviluppati utilizzando il linguaggio di programmmazione C# , data la maggiore documentazione relativa all'utilizzo dei servizi cognitivi e di elaborazione del testo della Microsoft.

6.1.1 Programmazione Asincrona

Entrambi gli agenti sono stati implementati come delle classi contenenti metodi asincroni. Questa scelta è dovuta al fatto che entrambi potranno lavorare parallelamente per l'esecuzione di operazioni che includono compu-

tazioni e richieste di I/O verso il web, in particolare chiamate HTTP alle API nel caso di comunicazione locale fra il sistema e le lampadine, o tramite un Webhook nel caso remoto con l'uso del servizio di IFTTT.

In particolare si utilizzano i Task appartenenti al namespace System.Threading.Tasks e abbinati alla parola chiave async, i quali vengono eseguiti dal thread correntemente in esecuzione e delegano il lavoro al sistema operativo, con un altro processo, quando necessario [13]. Ci sono due tipologie di attività da utilizzare in base al fatto che restituiscano o meno un risultato al chiamante. Qualora sorga la necessità di proseguire con l'esecuzione del codice da parte del conversationAgent, e quindi di non attendere il completamento di qualche Task, si utilizza la modalità con valore di ritorno corrispondente a un oggetto di tipo Task, questo perché restituendo l'attività stessa si includono gli eventuali errori catturati, ed è più facile controllare l'andamento di esso ed effettuare test di correttezza. Cosa che non è possibile utilizzando la metodologia di esecuzione di un Task con valore di ritorno void.

Infine, si utilizza l'operatore await per far sì che il chiamante si ponga in attesa di un risultato del Task, in modo sincrono, qualora questo lo restituisca tramite Task<?> consentendo comunque la reattività da parte del thread principale a fronte di eventi esterni.

Dialoghi

Si utilizzano i dialoghi tramite l'implementazione dell'interfaccia IDialog<> dell'SDK Microsoft per modellare e gestire il flusso e lo stato delle conversazioni. Analizzando gli esempi forniti dalla Microsoft per l'integrazione di Luis con il Bot Builder SDK, il flusso di esecuzione degli agenti nasce a fronte della ricezione di un primo messaggio.

Per catturare i messaggi sottoforma di richieste HTTP in arrivo da Luis e per inoltrarle nuovamente in direzione dell'utente, si implementa l'action method 'Post' rappresentante una richiesta POST effettuata al percorso `api/Messages`. L'implementazione è effettuata all'interno di un Controller,

detto MessageController, che estende le funzionalità dell'interfaccia ApiController, introdotto con .NET 4.0, in cui i metodi devono avere il prefisso corrispondente al verbo relativo alla richiesta HTTP (get, post, put, ...) senza che questi siano case-sensitive e che restituiscono automaticamente oggetti di tipo JSON o XML.

Da questo si utilizza il metodo della classe Conversation, sendAsync, che ha lo scopo di creare un dialogo, una conversazione, instanziando i componenti necessari e occupandosi della deserializzazione e serializzazione dello stato, per ogni dialogo creato. La prima volta che viene invocato questo metodo si instanzia il ConversationalAgent insieme allo stato, entrambi validi per il tempo di vita del dialogo in atto.

```
[ResponseType(typeof(void))]
public virtual async Task<HttpResponseMessage> Post
                        ([FromBody] Activity activity)
{
   if (activity.GetActivityType() == ActivityTypes.Message)
   {
        //ricezione di un messaggio dal canale/app
        await Conversation.SendAsync(activity, () =>
                                new ConversationalAgent());
    }
}
```

L'agente conversazionale estende la classe LuisDialog che rappresenta un dialogo specializzato nell'interazione con il bot engine Luis. Con la sua instanza è invocato il metodo StartAsync che si pone in attesa di ricevere messaggi non dal canale di comunicazione, cioè dall'app di messaggistica Telegram, ma direttamente dal Luis.

Una volta in esecuzione, alla ricezione di ogni intent da parte del bot engine, l'agente interagirà qualora sia necessario con l'agente di reasoning, e si porrà, eventualmente dopo aver processato le risposte ricevute dal secondo agente, in attesa di ricevere nuovamente messaggi tramite la chiamata Wait(MessageReceived) invocata su un oggetto di tipo IDialogContext, rap-

presentante il punto di accesso ai servizi di comunicazione con Luis e allo stato del dialogo. MessageReceived rappresenta invece la callback da invocare nel momento successivo all'arrivo di un intents da parte di Luis. Una volta elaborato ogni messaggio ricevuto, è fondamentale porsi nuovamente in attesa, o dichiarare il contesto del dialogo in atto come in attesa, Wait, chiuso con successo, Done, o con un fallimento, Fail. Questo perché altrimenti il bot framework lancerebbe una eccezione alla successiva ricezione, in quanto non sarebbe definito il punto di elaborazione del messaggio in ingresso.

Per poter rispondere all'utente, inoltrare quindi un messaggio o dei contenuti diversi, occorre utilizzare il metodo PostAsync, anch'esso disponibile tramite un oggetto IDialogContext. Quello che viene inoltrato all'utente è definito dal Bot Framework come una Activity. Questa può essere di diversi tipi come un messaggio testuale, IMessageActivity, e includere allegati, attachments, come immagini, video o audio, impostando il tipo MIME di un oggetto IAttachment. In alternativa si può inviare all'utente una Card, un pulsante a cui si collega una azione corrispondente da eseguire.

6.1.2 Conversational Agent

L'agente si occupa della ricezione degli intents e delle entities elaborate dal bot engine Luis. A partire dalla lista degli intents individuata in 5.2 si deriva una interfaccia, con l'obiettivo di implementare poi i corrispondenti metodi di elaborazione degli intents, all'interno di una classe denominata ConversationalAgent.

```
public interface IConversationalAgent
{
    Task ChangeColor(IDialogContext c, LuisResult r);
    Task DecreseBrightness(IDialogContext c, LuisResult r);
    Task GetState(IDialogContext c, LuisResult r);
    Task GreetingIntent(IDialogContext c, LuisResult r);
    Task IncreaseBrightness(IDialogContext c, LuisResult r);
    Task NoneIntent(IDialogContext c, LuisResult r);
```

```
    Task Reset(IDialogContext c, LuisResult r);
    Task SimpleAnswer(IDialogContext c, LuisResult r);
    Task TurnOff(IDialogContext c, LuisResult r);
    Task TurnOn(IDialogContext c, LuisResult r);
}
```

Listing 6.1: Interfaccia dell'Agente Conversazionale

L'agente rappresenta inoltre il dialogo gestendone il flusso ed estende quindi la classe LuisDialog che mette a disposizione i metodi per ottenere ed elaborare i risultati di Luis. L'agente, incarnando il dialogo, è inoltre dichiarato come oggetto serializzabile. Questo consente al bot framework di salvare lo stato del dialogo, interno all'agente conversazionale, nell'IBotDataStore così da recuperarlo alla ricezione di ogni messaggio.

Per l'implementazione della risposta all'intent Help si sfrutta la possibilità di utilizzare elementi grafici, come i pulsanti, supportati da diversi channel come ad esempio Telegram, Skype e Facebook Messenger, con l'obiettigo di eseguire le azioni corrispondenti. Si creano delle action corrispondenti alle operazioni disponibili che se cliccate dall'utente attraverso l'applicazione di messaggistica, Telegram, comporteranno l'inoltro al bot di un messaggio che lancia l'azione descritta.

```
var cardActions = new List<CardAction>
{
    new CardAction
    {
        Title = "Accendi la luce",
        Type = ActionTypes.ImBack,
        Value = "Accendi la luce"
    },
    //Other CardAction
};
var card = new HeroCard
```

```
{
    Title = "Cosa posso fare per te? ",
    Text = "Queste sono le operazioni attualmente
       disponibili",
    Buttons = cardActions
};
var activity = context.MakeMessage();
activity.Id = new Random().Next().ToString();
activity.Attachments.Add(card.ToAttachment());
await context.PostAsync(activity);
```

Listing 6.2: Creazione di un menu di pulsanti interno alla Chat

Si utilizzano le HeroCard con l'obiettivo di visualizzare all'interno della chat dei semplici pulsanti immediati per l'utente. Ogni CardAction raccolta all'interno del menu specifica il testo che l'utente leggerà tramite il parametro title, il valore inviato effettivamente al bot tramite il parametro value, e il type che identifica il fatto che sarà eseguita l'azione relativa all'invio del value della card al bot, con mittente l'utente.

Per quanto riguarda le interazioni che includono un invio dell'intenzione all'agente di reasoning, nell'elaborazione di queste saranno inoltrate innanzitutto al secondo agente, ponendosi poi eventualmente in attesa di risposta con l'operatore await della programmazione asincrona.

```
[LuisIntent("TurnOn")]
public async Task TurnOn(IDialogContext c, LuisResult r)
{
  if (ReasoningAgent.Instance.isResetRequired())
  {
    await context.PostAsync("Richiedi un reset per
       configurare il sistema");
  }else{
    var response=await ReasoningAgent.Instance.ManageOn(c);
    await manageHttpResponse(c,
```

```
                                (HttpResponseMessage)response);
    }
    c.Wait(this.MessageReceived);
}
```

Listing 6.3: TurnOn Conversational Agent

6.1.3 Reasoning Agent

Implementato come Singleton in quanto non ne sono necessari più di uno all'interno del sistema. Istanza creata con l'utilizzo di un oggetto di lock per evitare problemi di concorrenza come deadlock qualora più dialoghi siano attivi in una futura modifica del sistema.

```
interface IReasoningAgent
{
    Boolean isResetRequired();
    Task<HttpResponseMessage> ManageOn(IDialogContext c);
    Task<HttpResponseMessage> ManageOff(IDialogContext c);
    Task ManageReset(IDialogContext c);
    Task ManageConfirm(IDialogContext c, LuisResult r);
    Task ManageReject(IDialogContext c);
    Task ManageChangeColor(IDialogContext c, LuisResult r);
    Task ManageBrightness(IDialogContext c, LuisResult r,
                                            Boolean increase);
}
```

Listing 6.4: Reasoner Agent

Prima Configurazione

L'agente conversazionale utilizza spesso il servizio del reasoner detto isResetRequired per verificare se l'utente abbia avviato o meno la configurazione iniziale del sistema, a seconda che il sistema sia in locale o in remoto. Questo

è discriminato dal fatto che per essere configurato in una delle due modalità, deve essere attivo lo stato (nel caso remoto) o essere registrato l'indirizzo IP del bridge internamente alla medesima rete. Il conversationalAgent notificato del fatto che un reset è necessario avviserà l'utente. Qui avviene una prima interazione non banale, in quanto è stato predisposto il fatto che l'utente può rispondere direttamente con un sì, innescando i meccanismi di reset. Questo implica il push immediato dell'azione di reset all'interno dello stack di richieste in pending, in sospeso, gestito da parte del reasoner.

```
[Serializable]
public class PendingRequest : IPendingRequest
{
    private Stack<Action> stack;
    public PendingRequest(){...}
    public void pushAction(String i , List<String> e){...}
    public Action popAction(){...}
    public Boolean isAnyPendingReq(){...}
    public void deleteAll(){...}
}
```

Listing 6.5: Stack delle richieste in sospeso

Nella gestione di ciascun intent ricevuto dall'agente conversazionale, il reasoner per richiedere l'esecuzione di una certa azione al device IoT dovrà utilizzare a sua volta un servizio appartenente al livello sottostante nell'architettura, dal Web Of Things, tramite il metodo isLocalAPIEnable. Da ciascuna funzionalità eseguita attraverso l'uso del livello sottostante, si riceverà una risposta di successo o fallimento in modo tale da notificare l'utente dell'esecuzione della sua richiesta.

```
public async Task<HttpResponseMessage> ManageOn(
   IDialogContext c)
{
 setTimerCheckOn();
 setTimerSuggestChangeColor();

 if (WoT_TalkingLights.isLocalAPIEnable())
 {
    response = (HttpResponseMessage)await WoT_TalkingLights
       .CallHueLocal(c, WoT_TalkingLights.Command.On);
 }
 else
 {
    response = await WoT_TalkingLights.CallHueRemoteIFTTT(c
       ,          WoT_TalkingLights.Command.On);
    if (response.IsSuccessStatusCode) state.on(c);
 }
 return response;
}
```

Listing 6.6: TurnOn Reasoner Agent

Comportamenti Autonomi

Le azioni di ragionamento e di monitoraggio sono implementate ancora una volta con le meccaniche della programmazione asincrona. Alcune attività come ad esempio il controllo del numero di ore in cui una lampadina è tenuta accesa, o il controllo della disponibilità degli aggiornamenti del firmware, avvengono periodicamente. Per questo si utilizzano dei timer che devon consentire all'agente di reasoning di effettuare altre azioni nell'attesa, viene quindi implementato con l'idea che sia una sorta di sveglia periodica per l'agente, utilizzando una sequenza, stream, di eventi temporali. Si raggiunge questo obiettivo tramite i meccanismi di programmazione reattiva in cui si osserva, si ci pone in ascolto, di eventi generati da una sorgente, stream, servizi disponibili dal namespace System.Reactive. L'agente di reasoning si

pone quindi come Observer di un oggetto Timer della classe Observable impostando l'intervallo di tempo e sottoscrivendosi a esso specificando l'azione da eseguire. Questo può includere computazioni e controlli con l'eventuale inoltro di un messaggio proattivo tramite il metodo Resume che ha lo scopo di recuperare la conversazione attualmente in atto senza interromperne il flusso.

```
private void setTimerCheckOn()
{
  IObservable<long> timer = Observable.Timer(
            TimeSpan.FromHours(WARNING_HOUR_ON));
  IDisposable timerCheckOn = timer.Subscribe(x =>
  {
    Resume(MessagesController.conversationId,
        MessagesController.channelId, "Non voglio
        disturbarti, ma la tua luce e' accesa da un po',
        ricordati di spegnerla se non ti serve!");
  });
}
```

Listing 6.7: Configurazione monitoraggio periodico

Per quanto riguarda il controllo degli aggiornamenti del firmware, questo viene effettuato periocamente ogni settimana. Seguendo la documentazione delle API per l'update di Philips Hue, si procede nel seguente modo:

1. Occorre verificare che il bridge presenti nella configurazione la connessione al portare tramite il parametro 'signedon' presente nel file JSON di config recuperabile con una richiesta GET all'url `api/USER_KEY/config/` all'interno del campo portalstate/signedon [20].

Figura 6.1: Parte del JSON del file config del bridge

2. Accertati della possibilità di effettuare aggiornamenti, si procede con il controllo della presenza di updates disponibili. Questo varia in base all'api version utilizzato dal bridge, dato ottenibile dal file config.

   - Con le API-Version ≥ 1.2 : si modifica il valore 'checkforupdate' impostandolo a true con una chiamata HTTP e metodo PUT. In questo modo il bridge automaticamente controllerà la disponibilità degli aggiornamenti e modificherà lo state di 'swupdate2'. Si con-

trolla successivamente il valore assunto da state di 'swupdate2'. Nel caso di a 'nyreadytoinstall' o di 'allreadytoinstall', si ha la possibilità di installare gli updates. Si procede quindi con una seconda PUT per modificare il valore 'install' interno a 'swupdate2' con true.

- Con le API-Version < 1.2 si procede in modo similare. Dapprima si avvia il controllo degli aggiornamenti da parte del bridge con una PUT per impostare il valore 'checkforupdate', interno a 'swupdate', a true. Dato questo si controlla successivamente il valore assunto da parte del campo 'updatestate', sempre all'interno di 'swupdate'. Qualora questo abbia assunto valore pari a 2, si procede con l'installazione modificandolo con una PUT con il valore 3.
- In entrambi i casi prima di procede con l'installazione, si avvisa l'utente che il bridge potrebbe essere non raggiungibile per qualche minuto. Questo potrebbe essere integrato con una richiesta di conferma da parte dell'utente o di rischedulaggio dell"aggiornamento in un momento più comodo per la persona.

Configurazione Iniziale

La configurazione iniziale del sistema si distingue in base alla possibilità di attivare le API in locale o in remoto.

Per quanto riguarda l'integrazione del servizio IFTTT nel caso di connessione remota, si procede semplicemente creando lo stato da aggiornare al seguito dell'esecuzione di ogni azione che comprende una modifica di l'accensione/spegnimento o del colore.

Nel caso locale occorre ottenere l'IP interno del bridge e ottenere uno username personalizzato. Questo si ottiene utilizzando l'account dello sviluppatore all'interno del bot, senza la necessità che l'utente debba effettuare operazioni aggiuntive. Le richieste per l'ottenimento dell'IP e dello username autorizzato si effettuano utilizzando i servizi disponibili dal livello WoT: se-

tIPID, getAuthUser. Per ottenere il primo dato occorre però richiedere una minima interazione con l'utente in quanto per questioni di sicurezza, questo viene fornito in risposta dal bridge solo con la pressione del pulsante presente sullo stesso. L'interazione prevede quindi che il bot richieda all'utente di avvertirlo non appena premuto il pulsante, in modo tale che nei 60 secondi di attivazione di esso, il bot possa procedere con le richieste al bridge per ottenere i dati. Questo comporta l'utilizzo dello stack delle richieste in sospeso. Come visto per il menu opzioni anche in questo caso si genera un IActivity con un allegato visivo per aiutare l'utente nel processo di configurazione, seppur questo minimo, includendo la visualizzazione dell'immagine di pressione del pulsante.

```
await context.PostAsync("Reset In Corso...");
HttpResponseMessage r = await WoT_TalkingLights.setIPID();
if (r.IsSuccessStatusCode)
{
  var activity = context.MakeMessage();
  activity.Text = "Devi premere il pulsante presente nel
     bridge e AVVERTIRMI appena fatto. Questo perch  ho
     60 secondi di tempo per tentare la configurazione
     completa del sistema." +
      "Rimango in attesa di una tua risposta...";
  activity.Id = new Random().Next().ToString();
  Attachment att = new Attachment
  {
      Name = "Connecting_Bridge",
      ContentUrl = BRIDGE_IMAGE_PRESS,
      ContentType = "image/png",
  };
  activity.Attachments.Add(att);
  await context.PostAsync(activity);
  pendingReq.pushAction("Reset", null);
}
```

Listing 6.8: Configurazione Iniziale

Figura 6.2: Esecuzione del reset

6.2 Web of Things

Procedendo con l'approccio della programmazione asincrona si sviluppa la classe WoTTalkingLights che incarna le funzionalità del livello del Web Of Things, responsabile della comunicazione diretta con l'end-device tramite il bridge, e con il livello soprastante Agents for things.

All'interno di questa sono disponibili i metodi per effettuare le chiamate da remoto tramite il servizio IFTTT e da locale. Entrambe le funzionalità richiedono un riferimento al dialogo attualmente attivo e due stringhe corrispondenti al comando principale e al secondo parametro opzionale.

```
//API REMOTE - IFTTT
public static async Task<HttpResponseMessage>
        CallHueRemoteIFTTT(IDialogContext c, string cmd,
                                    string subCmd = "");
public static string MakeHueUrlAPI();

//API LOCALI
public static async Task<Object> CallHueLocal(
   IDialogContext c, string cmd, string subCmd = "");
public static async Task<HttpResponseMessage> getAuthUser(
                          string appName, string devName);
public static async Task<HttpResponseMessage>
   HTTPRequestLocalAPI(HttpMethod method, string url,
                                    JObject body = null);
public static async Task<Object> getLocalAPI(string url);
public static async Task<HttpResponseMessage> setIPID();
```

Listing 6.9: Metodi principali della classe WoTTalkingLights

Con il primo, CallHueRemoteIFTTT, si effettua una POST HTTP al webHook messo a disposizione da un account creato appositamente su IFTTT da cui è avvenuta la configurazione delle proprio lampadine. L'url della richiesta è della forma `https://maker.ifttt.com/trigger/{cmd}/with/key/{key}` dove cmd indica l'azione da eseguire principale e key indica la chiave IFTTT fornita all'interno dell'account. Le risponde forniscono una semplice risposta di successo o di errore di HTTP.

Per quanto riguarda la gestione dell'interazione all'interno della medesima rete, in locale, questa comporta una maggiore complessità. Anche in questo caso è presente una prima configurazione per ottenere il numero di lampadine disponibili e attive in un determinato momento così che l'utente possa in un secondo momento sceglierne una o più, assegnando eventualmente un nome, per far sì che le API siano indirizzate alla corretta lampadina desiderata.

Si utilizzano chiamate HTTP GET per l'ottenimento delle informazioni riguardanti lo stato, e chiamate con metodo GET e PUT per la modifica

dello stato della lampadina in un determinato momento.

Per il primo caso si può ottenere un oggetto StateAPILamp contenente i dati utili per lo stato e rilevanti per l'utente. Questo avviene deserializzando l'oggetto JSON ottenuto come risposta dall'API. Questo metodo è utilizzato anche per recuperare i dati del file config relativi al bridge.

L'accensione e lo spegnimento di una lampadina viene effettuato immediatamente senza controlli, se non in caso risposta ottenuta dal bridge al seguito dell'operazione eseguita. Con la modifica del colore e della luminosità viene effettuato prima un controllo sullo stato per far sì che si attivi l'interazione della richiesta di autorizzazione all'accensione qualora la lampadina sia spenta.

Il cambio di luminosità implica inoltre il recupero dell'attuale valore così che la chiamata PUT successiva, per la modifica, sia coerente con l'automento o l'abbassamento graduale della luce, considerando che il range delle API in locale per il valore 'bri' varia fra 0 e 255.

La richiesta di una modifica della colorazione implica una correzione del valore elaborato e individuato da Luis come entity, fornito dall'agente conversazionale. Questo potrebbe essere una colore espresso in italiano o in esadecimale. Dapprima si verifica la correttezza dell'input, per quanto riguarda l'esadecimale che sia nel corretto formato, e per quanto riguarda il nome italiano che questo sia presente in un elenco fornito al sistema a tempo di design, in base ai colori riconosciuti dal modello Luis e funzionanti con le lampadine Hue. Il valore sarà poi convertito prima in RGB e poi in XY secondo un algoritmo fornito proprio dalla Philips per permettere la conversione del colore nel formato di coordinate XY [5]. Occorre considerare anche il parametro 'colorMode' che può assumere valore 'hs' per saturazione e luce, 'xy' o 'ct' per temperatura di colore.

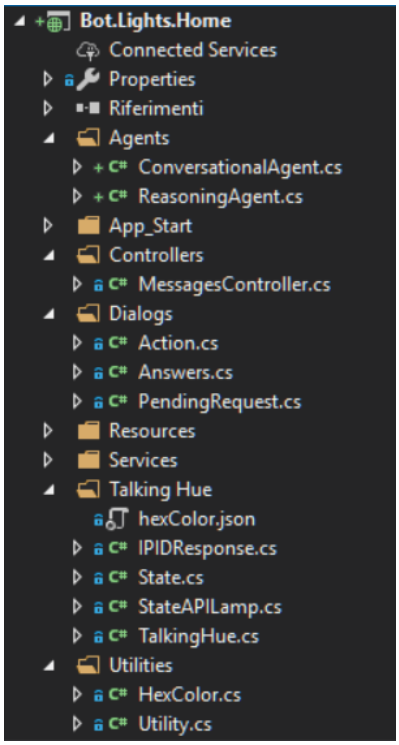


Figura 6.3: Organizzazione del progetto

Capitolo 7

Testing

I test sono stati effettuati alla conclusione di ogni singola features individuata nei requisiti funzionali in fase di analisi.

Questi sono stati effettuati dapprima in locale tramite l'emulatore fornito dalla Microsoft, ciò per evitare di dover attivare una sottoscrizione a consumo per utilizzare i servizi di Azure. Avviando l'applicazione tramite l'IDE utilizzato, Visual Studio 2017, si ottiene l'esecuzione del bot in locale, pertanto occorre impostare nell'emulatore la porta selezionata come URL di progetto, l'app ID di Azure e la relativa password entrambi presenti all'interno delle impostazioni dell'applicazione della web app bot configurata nel portale di Azure.

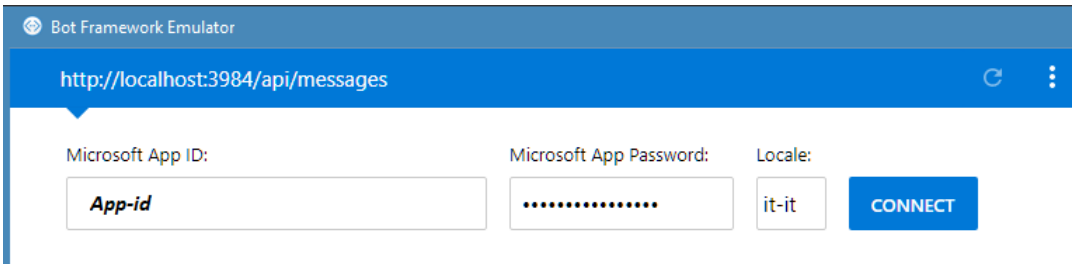


Figura 7.1: Configurazione del Microsoft BotFramework Emulator

Le funzionalità sono state tutte testate con successo, utilizzando le più comuni frasi, similari a quelle utilizzate nella fase di training del modello di Luis. Sono stati inoltre verificati i diversi casi più o meno complessi che si

possono creare, quali la richiesta di modifica del colore o della luminosità di una lampadina spenta, un intent non compreso, i comportamenti proattivi dell'agente reasoning, l'installazione degli aggiornamenti del firmware. Per verificare inoltre alcune modifiche si è utilizzato la richiesta dello stato per monitorare i valori modificati dal sistema e la loro correttezza risultante.

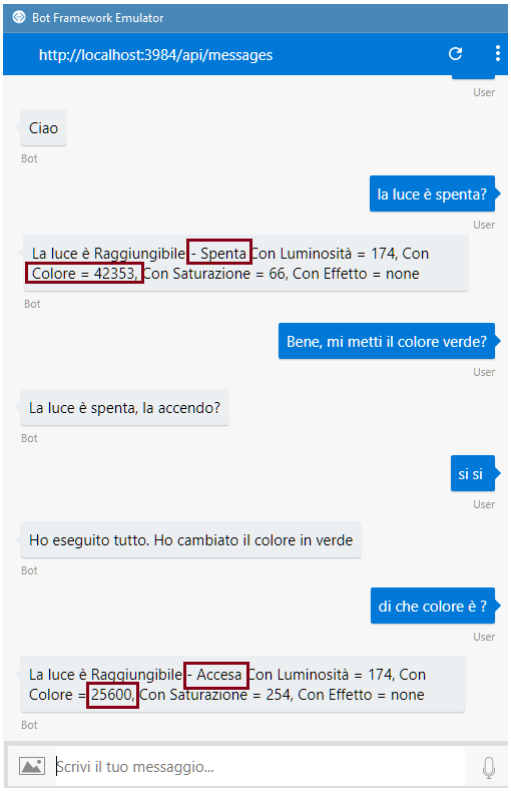


Figura 7.2: Test con il Microsoft BotFramework Emulator per il cambio colorazione

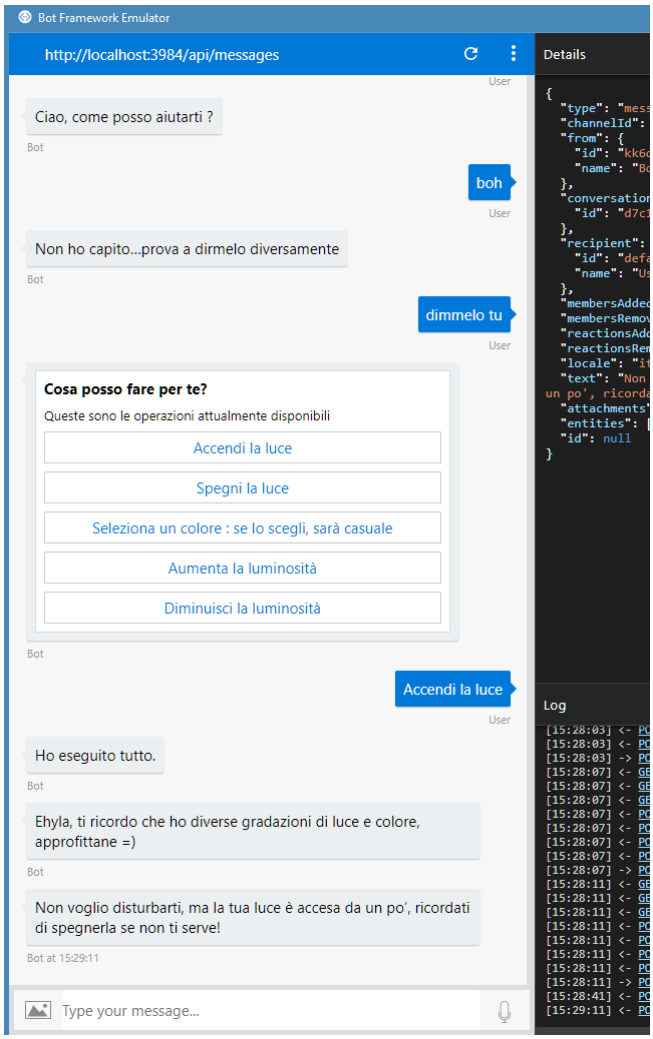


Figura 7.3: Test con il Microsoft BotFramework Emulator per il menu e i suggerimenti

Il deploy e il testing sul canale Telegram è preceduto da quello sulla web chat messa a disposizione dal portare di Azure. Per questo occorre aggiornare il progetto sincronizzandolo con il branch selezionato dal progetto caricato su GitLab. Controllare di aver correttamente modificato i parametri presenti alla sezione ApplicationSetting, in particolare: LuisAppId, LuisAPIKey, LuisApiHostName e compilare il bot avviando il file build.cmd presente nel compilatore online. Interessante è la possibilità di monitorare eventuali errori con la corrispondente descrizione, all'interno della sezione canali della web

app bot, che consente di individuare errori nell'utilizzo da parte degli utenti finali del sistema.

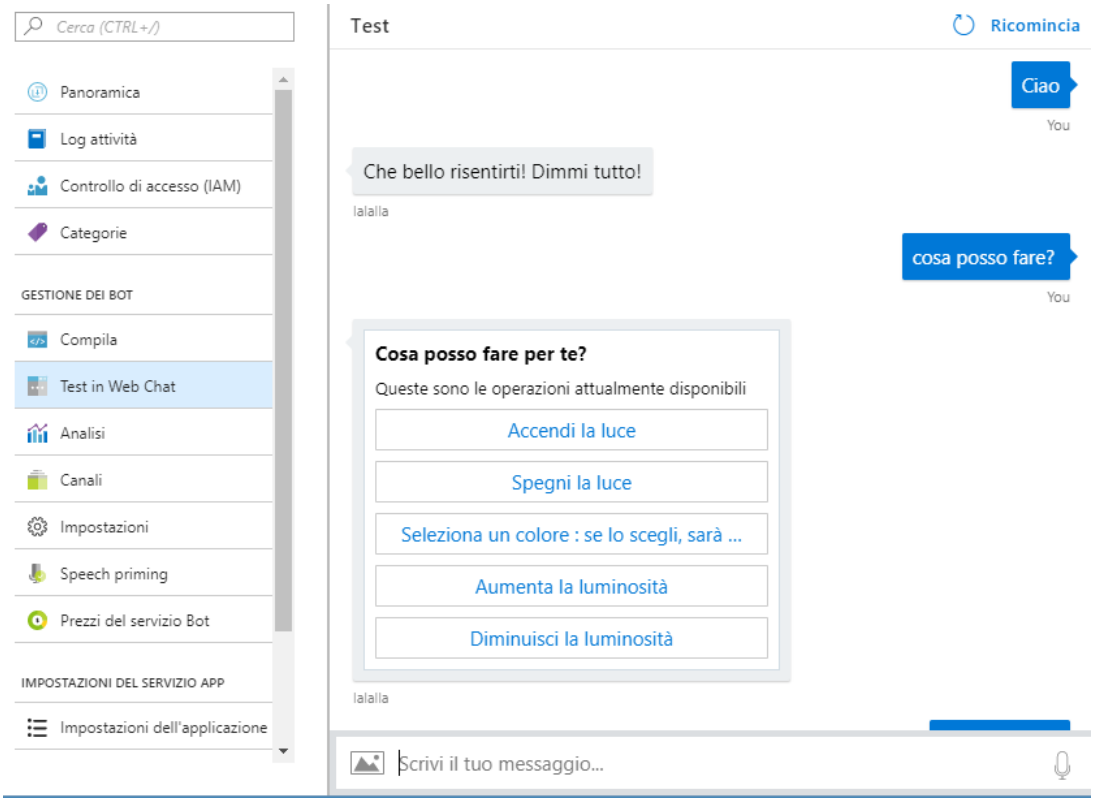


Figura 7.4: Test tramite l'interfaccia web di Azure

La connessione della web app bot con il canale Telegram avviene con l'inserimento del token del bot creato dall'applicazione di messaggistica con il BotFather, nella sezione canale.

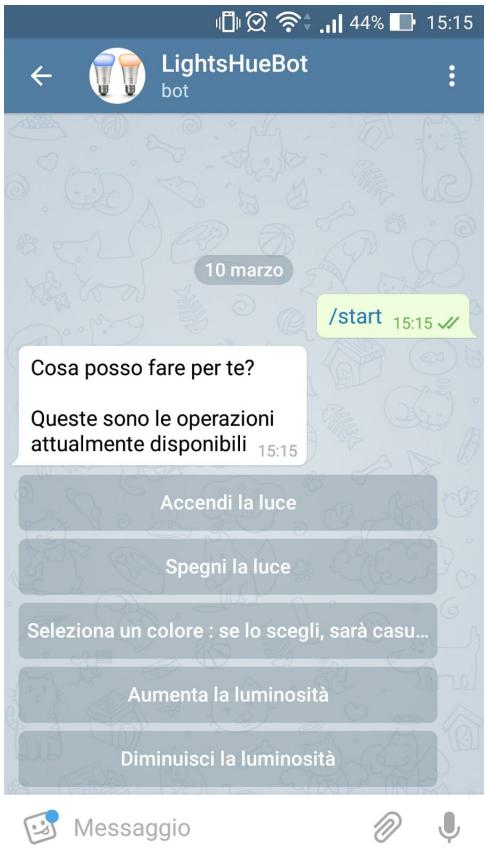


Figura 7.5: Test tramite l'app Telegram

L'utilizzo di Telegram ha consentito il test delle funzionalità non solo in locale ma anche da remoto tramite il servizio IFTTT. Pertanto due mie amiche si sono prestate come beta tester, chattando col bot tramite l'ID da me fornito. I test sono stati effettuati connettendosi in giorni e momenti diversi in quanto entrambe accedevano alla lampadina presente all'interno di casa mia. Il fatto di non aver formato le due persone all'utilizzo e all'interazione con la lampadina, mi ha permesso di addestrare notevolmente il sistema. Questo grazie al fatto che all'interno del portale di Luis si registrano le utterances in input al bot engine e la cui elaborazione non è stata confermata dallo sviluppatore.

Attraverso il servizio di Luis si può inoltre monitorare lo stato di utilizzo del servizio e analizzare alcuni dati statistici, come il numero di intents, enti-

ties, e utterances presenti. Un grafico chiamato Intent Breakdown mostra le percentuali di chiamate per ogni intents, così da capire l'utilizzo e la mancanza eventuale di intents qualora troppi input siano marcati come none. Allo stesso modo è mostrato un istogramma per visualizzare il numero di entity etichettate per tipo nelle utterances .

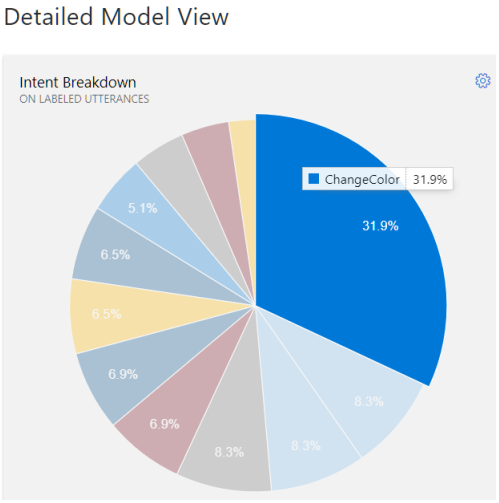


Figura 7.6: Intents BreakDown

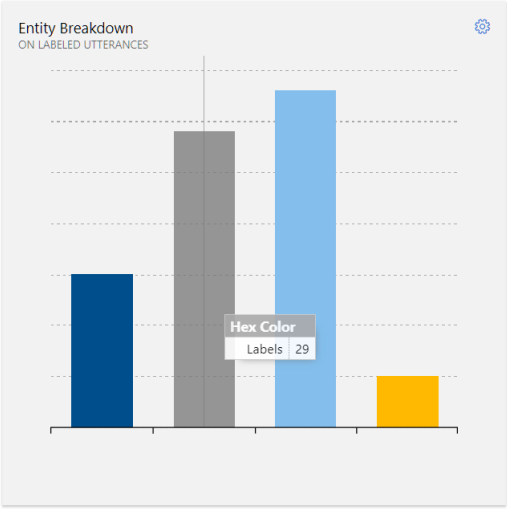


Figura 7.7: Entities BreakDown

Conclusioni

Risultati Ottenuti

I test effettuati su Telegram hanno confermato il soddisfacimento dei requisiti funzionali individuati, sia all'interno dell'ambiente domestico tramite wi-fi, sia accedendo ai dispositivi da remoto con una diversa connessione alla rete internet.

Per quanto riguarda i requisiti non funzionali, questi sono stati tutti raggiunti in quanto tramite le tecnologie scelte, l'utente potrebbe utilizzare diverse applicazioni di messaggistica in base alla propria preferenza, senza dover imparare nuove metodologie di utilizzo di app mobile e instaurando un contatto diverso, più familiare e informale, con gli oggetti IoT, mantenendo quindi un ottimo livello di attrattività per l'utente finale.

Per quello che concerne l'architettura, quella individuata permette l'integrazione di diversi servizi, di database esterni in appoggio, in particolare per il livello Agents for Things, a cui si possono applicare diverse architetture ad agenti come la BDI, o meccanismi di intelligenza artificiale per adattarsi maggiormente alle interazioni degli utenti e ai diversi dialoghi.

La struttura teorica dell'architettura può valere per qualsiasi tecnologia, così come qualsiasi end-device IoT ed è anche applicabile tramite l'uso di diversi framework, non solo quello utilizzato all'interno del progetto in questione.

Sviluppi Futuri

Dal punto di vista delle funzionalità relative all'interazione con le lampadine intelligenti, sono molte quelle aggiungibili ed estendibili. Innanzitutto la schedula di attività semplici come l'accensione o lo spegnimento, o più complesse, come ad esempio una sequenza di colorazioni e di luminosità per creare del dinamismo automatizzato nella luce, come al risveglio, o durante una cena. Si è inoltre in attesa di ottenere le autorizzazioni per l'uso delle API da remoto e per la pubblicazione di applets con IFTTT.

Il sistema ampliato a più lampadine dovrà gestire interazioni complesse con l'utente per capire come poterle identificare e distinguere, gestendo stanze e gruppi di luci, memorizzando preferenze e dati dell'utente per ottenere una esperienza personalizzabile al massimo.

Una delle idee iniziali era quella di ottenere un sistema funzionante con la maggiorparte dei dispositivi IoT, questione critica che richiede una attenta e approfondita analisi. Si è visto come la Philips abbia creato degli ostacoli non fornendo un sistema completamente aperto in merito alle API per l'interazione con le lampadine da remoto. Occorre considerare la grossa diversità dei servizi disponibili per tipologia di oggetto e per azienda. Una macchinetta del caffè smart potrebbe rendere pubbliche API ben diverse in una azienda rispetto ad un'altra, o semplicemente utilizzare meccaniche e nomenclatura diverse.

Con il superamento del problema dell'eterogeneità dei diversi smart device, o comunque migliorando l'intelligenza del sistema a fronte di diversi input e diverse funzionalità, anche nuove, si può raggiungere a pieno l'accettazione da parte degli utenti e la preferenza all'acquisto di uno smart object piuttosto del semplice oggetto comunemente conosciuto.

Qualsiasi persona non dovrà più sentirsi strana o pazza di fronte a un discorso con un oggetto, come l'inveire contro la propria automobile o l'incoraggiare il proprio computer nel caricare una pagina dal web.

Dato che l'antropomorfizzazione degli oggetti è in realtà una naturalezza

intrinseca dell'essere umano e non una stranezza, con l'integrazione dei chatbot all'IoT, questo diventerà una consuetudine e una opportunità continua di miglioramento dello stile di vita e delle esperienze personali...

mantenendo la promessa che Pillowe, il cuscino intelligente [19], un domani non troppo lontano, mi dia il buongiorno su Whatsapp, mi chieda se voglio preparato subito il caffè e parlando a sua volta con la macchinetta, mi dia la possibilità di trovarlo subito pronto e caldo in cucina.

Ringraziamenti

Ringrazio in primis il Prof A. Ricci, il Dott.T. Dionigi e il Prof L. Margara che mi hanno dato l'occasione di poter sviluppare questo progetto e con cui condivido la passione, l'entusiasmo e l'energia per l'IoT, per le sfide e nella ricerca continua di nuovi stimoli e idee.

Un ringraziamento speciale va a mia nonna e ai miei zii, la mia famiglia. Non hanno mai smesso di sostenermi di fronte a qualsiasi battaglia, con forza e affetto.

Vorrei inoltre ringraziare le mie più care amiche, Alba e Gaia, che mi sono rimaste vicine in ogni momento, senza esitazioni, con pazienza, condividendo gioie, momenti critici e un immancabile briciolo di pazzia

Bibliografia

[1] C. J. Baby, F. A. Khan e J. N. Swathi. "Home automation using IoT and a chatbot using natural language processing". In: *2017 Innovations in Power and Advanced Computing Technologies (i-PACT)*. Apr. 2017, pp. 1–6. DOI: 10.1109/IPACT.2017.8245185.

[2] Andrea Bolioli. *Il Natural Language Processing nei chatbot*. URL: https://www.celi.it/blog/2016/06/il-natural-language-processing-nei-chatbot/.

[3] *Bot Service Documentation*. URL: https://docs.microsoft.com/en-us/bot-framework.

[4] J.K. Choi e K.Y. Jeon. *Method and system for controlling internet of things (IoT) device*. US Patent 9,716,675. Lug. 2017. URL: https://www.google.it/patents/US9716675.

[5] *Color conversion formulas RGB to xy and back*. URL: https://www.developers.meethue.com/documentation/color-conversions-rgb-xy.

[6] Wikipedia contributors. *Chatbot — Wikipedia, The Free Encyclopedia*. 2017. URL: %5Curl%7Bhttps://en.wikipedia.org/wiki/Chatbot%7D.

[7] D.M. Deen, T. Chipman e R. Snelson. *Messaging system based building control*. US Patent 7,403,838. Lug. 2008. URL: https://www.google.it/patents/US7403838.

[8] Inc Gartner. *Gartner Says Organizations Need to Master Two Dimensions of Mobility*. URL: https://www.gartner.com/newsroom/id/3456618.

[9] Inc Gartner. *Gartner Top Strategic Predictions for 2018 and Beyond*. URL: https://www.gartner.com/smarterwithgartner/gartner-top-strategic-predictions-for-2018-and-beyond/.

[10] Heesik Jeon et al. *An Intelligent Dialogue Agent for the IoT Home*. 2016. URL: https://www.aaai.org/ocs/index.php/WS/AAAIW16/paper/view/12596.

[11] Anirudh Khanna et al. "A Discussion about Upgrading the Quick Script Platform to Create Natural Language based IoT Systems". In: *Indian Journal of Science and Technology* 9.46 (2016). ISSN: 0974 -5645. URL: http://52.172.159.94/index.php/indjst/article/view/106917.

[12] Anatoly Khorozov. *Trends Driving the Chatbot Growth*. URL: https://chatbotsmagazine.com/trends-driving-the-chatbot-growth-77b78145bac/.

[13] *La programmazione asincrona in dettaglio*. URL: https://docs.microsoft.com/it-it/dotnet/standard/async-in-depth.

[14] M. Lippi et al. "Coordinating Distributed Speaking Objects". In: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. Giu. 2017, pp. 1949–1960. DOI: 10.1109/ICDCS.2017.282.

[15] Wail Mardini, Yaser Khamayseh e Ashraf Smadi. "Messenger Bot for IoT Devices". In: *Proceedings of the 9th International Conference on Information Management and Engineering*. ICIME 2017. Barcelona, Spain: ACM, 2017, pp. 182–186. ISBN: 978-1-4503-5337-3. DOI: 10.1145/3149572.3149611. URL: http://doi.acm.org/10.1145/3149572.3149611.

[16] D. Peng e C. Peng. "A design and implement for simple smart home system for consumers". In: *2016 Chinese Control and Decision Conference (CCDC)*. Mag. 2016, pp. 4690–4694. DOI: 10.1109/CCDC.2016.7531831.

[17] *Philips Hue*. URL: https://www2.meethue.com/it-it.

[18] *Philips Hue Dev*. URL: https://developers.meethue.com/documentation/getting-started.

[19] Greta Sasso. ""Pillowe": progettazione e sviluppo di un cuscino intelligente". Tesi di dott. URL: http://amslaurea.unibo.it/8844/.

[20] *Software update*. URL: https://www.developers.meethue.com/documentation/software-update.

Listings

Elenco delle figure

Elenco delle tabelle