Alma Mater Studiorum
Università di Bologna

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

Computazione Embodied e Disembodied: Cloud-based IoT

Tesi in
Sistemi Distribuiti

Candidato
Nompleggio Pietro Antonio

Relatore
Prof. Andrea Omicini
Correlatore
Dott. Stefano Mariani

Anno Accademico 2015/2016 - Sessione I

A Giulia, per avermi dato la forza di continuare,
a mia sorella, che mi ha sempre aiutato,
ai miei genitori, senza i quali tutto questo
non sarebbe stato possibile.

Indice

Introduzione

Negli ultimi anni la rete ha subito un'evoluzione così straordinaria da permettere di parlare di "nuova era" e "nuovo mondo" connesso. Nel 1992 c'erano tanti dispositivi connessi a Internet quanto la popolazione di San Jose, California, oggi ci sono più dispositivi connessi di esseri umani sul pianeta. Questa espansione non è solo dovuta a telefoni, tablet o portatili, ma a elettrodomestici, luci del traffico, turbine per il vento, spazzolini elettrici, finestre, serrature, televisioni ecc., ora quasi tutto ha un indirizzo IP.

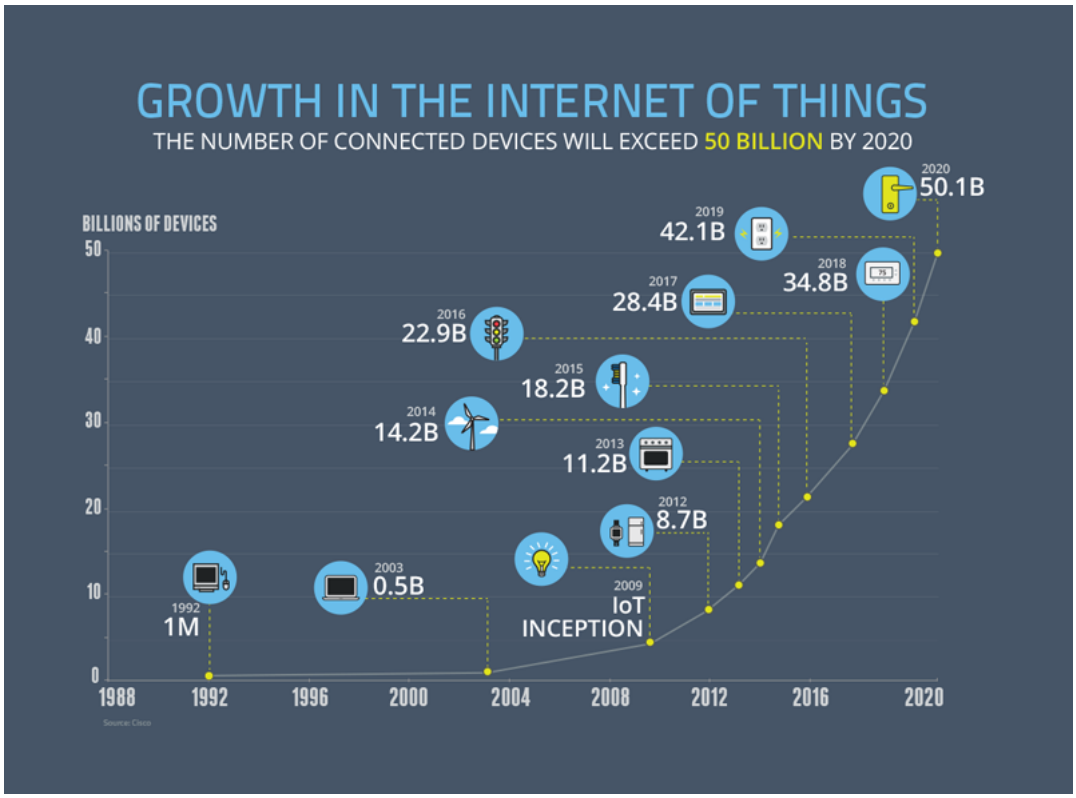

Figura 1: Crescita dell'IoT negli anni.

Come possiamo vedere dal grafico, l'Internet of Things (IoT) sta avendo una crescita esponenziale, e si calcola che nel 2020 ci saranno circa 50 miliardi di dispositivi connessi, aumentando sempre di più. E' un fenomeno che sta diventando una delle tendenze più rilevanti nella storia dell'industria del software e dell'hardware grazie anche alla connettività, lo spazio di archiviazione e di computazione che stanno diventando sempre più pervasivi.

La prima generazione di soluzioni IoT prettamente embodied si erano focalizzate su frameworks come Arduino o Raspberry Pi visto il loro basso costo e la facilità di comunicazione con sensori smart, ma sta emergendo una nuova generazione di dispositivi per l'IoT e soprattutto di piattaforme che ne abilitano capacità backend.

Questi nuovi dispositivi insieme all'ormai ubiquità della rete rende semplice e economico raccogliere informazioni nel mondo fisico su oggetti e luoghi che prima erano considerati "offline" e ora "prendono vita", riuscendo a catturare gli eventi che avvengono continuamente nel mondo fisico. Se immaginiamo un piccolo dispositivo equipaggiato con un sensore, da solo potrebbe generare una piccola quantità di dati, se moltiplichiamo questo sensore per centinaia, migliaia, milioni di sensori abbiamo davanti ai nostri occhi una quantità di informazioni enorme, che ha bisogno di uno spazio in cui essere archiviata, e elaborata.

E' qui che entra in campo la piattaforma cloud, che grazie alla sua natura disembodied riesce a gestire queste informazioni che riceve da tutti i dispositivi connessi fornendo una visualizzazione dei dati in real-time, una computazione veloce, e soprattutto scalabile in modo da avere accesso in qualsiasi momento a una vasta quantità di informazioni. Questi dati sono vitali per esempio per prendere decisioni strategiche nel ciclo produttivo di un'azienda o in molti altri campi per scopi differenti e la computazione di questi dati molte volte non può aspettare un'analisi offline, o in molti casi non è sufficiente.

Combinando quindi insieme la natura embodied dei dispositivi IoT con quella disembodied del cloud è possibile realizzare infrastrutture che possono

ricevere un flusso continuo di informazioni, elaborarle in run-time fornendo un'analisi predittiva, questo cambia profondamente il modo in cui si monitorano e gestiscono i dispositivi IoT, apre un nuovo mondo verso le soluzioni pro attive in risposta ad eventi real-time.
La tesi intende analizzare come i dispositivi IoT interagiscono con il Cloud realizzando una piattaforma Middleware che riesca a mettere in luce le caratteristiche principali di entrambi in particolare di come le funzionalità embodied e disembodied possono coesistere in questi sistemi.

Capitolo 1

Internet of Things (IoT)

In questo capitolo verrà introdotto l'IoT, i vari modi con cui può essere definito, i modelli di comunicazione, i protocolli a livello applicativo che possono essere scelti in base allo scenario e infine come si sta muovendo il mercato in cerca di uno standard.

1 Cos'è l'Internet of Things (IoT)?

Il termine "Internet of Things" (IoT) fu usato per la prima volta dall'imprenditore tecnologico britannico Kevin Ashton nel 1999 per descrivere un sistema nel quale gli oggetti situati nel mondo fisico possono essere collegati a internet attraverso i sensori. Ashton ha coniato questo termine perchè stava lavorando sui dispositivi RFID, e voleva usarlo per illustrare la potenza di collegare tags RFID a internet in modo da tracciare i beni senza l'intervento umano. Oggi, l'Internet of Things è diventato un termine popolare che descrive scenari in cui la connessione Internet e la capacità computazionale estende tanti oggetti.

Mentre il termine "Internet of Things" è relativamente nuovo, il concetto di combinare computer e rete per monitorare e controllare i dispositivi è nell'aria da decenni. Entro la fine del 1970, ad esempio, i sistemi per monitorare da remoto le reti elettriche tramite le linee telefoniche erano già in commercio. Nel 1990, i progressi della tecnologia wireless hanno permesso connessioni "machine-to-machine" (M2M) a livello industriale, per monitorare e operare

a distanza, molte di queste soluzioni però erano basate su una rete chiusa e proprietaria della specifica industria, piuttosto che su protocolli IP basati su reti e standard di Internet.

L'uso degli indirizzi IP per connettere dispositivi invece che computer non è un'idea nuova. Il primo dispositivo abilitato con un indirizzo IP fu un Tostapane che poteva essere acceso e spento attraverso Internet, fu mostrato a una conferenza Internet nel 1990. Negli anni successivi altre "things" furono "attrezzate" con un indirizzo IP, come per esempio un distributore di bevande gassate al Carnegie Mellon University in USA e una caffettierra all'University of Cambridge in UK (che rimase connessa fino al 2001). Da questi inizi stravaganti, un robusto campo di ricerca e sviluppo ha continuato lo studio nello "smart object networking" e ha aiutato a creare le fondamenta di quello che noi oggi chiamiamo Internet of Things.

Se l'idea di connettere gli oggeti tra di loro non è nuova, bisogna chiedersi, "Perchè l'Internet of Things è così popolare oggi?"
Da un'ampia prospettiva possiamo dire che la confluenza di diverse tecnologie e tendenze di mercato ha reso possibile connettere sempre più dispositivi a basso costo e in modo semplice, possiamo racchiudere questa crescita esponenziale in 6 punti chiave:

- **Ubiquità della connessione**, basso costo, alta velocità, pervasività della connessione di rete, rende quasi tutto "connettibile".
- **Diffusa adozione di reti IP-based**, IP è diventato lo standard globale dominante per la rete, fornisce una piattaforma ben definita e ampiamente implementata di software e strumenti che possono essere incorporati in una vasta gamma di dispositivi facili da utilizzare ed economici.
- **Capacità di calcolo economica**, guidata dai grandi investimenti delle industrie nella ricerca, sviluppo e produzione, la legge di Moore continua a fornire una maggior potenza di calcolo a basso costo e basso consumo energetico.
- **Miniaturizzazione**, i progressi costruttivi permettono di incorporare la capacità di calcolo all'avanguardia e la comunicazione in oggetti mol-

to piccoli. Questo, accoppiato con la capacità di calcolo a basso costo, ha alimentato l'avanzamento di piccoli e economici sensori che guidano molte applicazioni IoT.

- **Avanzamento nell'analisi dei dati**, nuovi algoritmi e la rapida crescita della potenza di calcolo, spazio di archiviazione e servizi cloud ha abilitato l'aggregazione, correlazione e l'analisi di grandi quantità di dati; questi vasti e dinamici insiemi di informazioni forniscono nuove opportunità di estrarre conoscenza.
- **Ascesa della computazione cloud**, la computazione cloud sfrutta l'elaborazione remota in rete di processi, gestione e archiviazione di dati, permette a piccoli dispositivi distribuiti di interagire attraverso un potente back-end con potenti capacità analitiche.

Da questa prospettiva, l'IoT rappresenta la convergenza di una varietà di computazione e connessioni che si è evoluta nel corso degli anni. Allo stato attuale, una vasta gamma di settori industriali come: automobilistico, assistenza sanitaria, fabbricazione, casalingo e elettronica di consumo e altre stanno prendendo in considerazione la possibilità di incorporare l'IoT nei loro prodotti, servizi e operazioni.
In un report del McKinsey Global Institute chiamato "Unlocking the Potential of the Internet of Things"[1], viene descritta l'ampia gamma di potenziali applicazioni in termini di "settore" dove l'IoT si prevede crei un valore per l'industria e per gli utenti, di seguito viene mostrato il riassunto di questo report.

[1]Manyika, James, Michael Chui, Peter Bisson, Jonathan Woetzel, Richard Dobbs, Jacques Bughin, and Dan Aharon. "The Internet of Things: Mapping the Value Beyond the Hype." McKinsey Global Institute, June 2015. p.3.

Settore	Descrizione	Esempio
Umano	Dispositivi collegati o all'interno del corpo umano	Dispositivi (wereable o ingeribili) per monitorare la salute e il benessere, malattie, fitness ecc.
Casa	Edifici Residenziali	Home Automation e sistemi di sicurezza
Uffici	Spazi di lavoro	Gestione dell'energia e sicurezza negli uffici, miglioramento della produttività
Veicoli	Sistemi all'interno dei veicoli	Veicoli che comprendono automobili, camion, navi, aerei, treni, includendo un sistema di manutenzione, statistiche, posizione
Città	Ambienti urbani	Spazi pubblici e infrastrutture urbane, adozione di controllo del traffico, contatori intelligenti, gestione delle risorse
Negozi	Spazi dove i consumatori commerciano	Negozi, banche, ristoranti, stadi, ovunque il consumatore compri qualcosa, possibilità di effettuare il conto senza un commesso, avviso di sconti all'interno dei negozi, ottimizzazione dell'inventario

Molte organizzazioni hanno sviluppato la loro tassonomia e categorizzazione per le applicazioni IoT e casi d'uso. Per esempio, "Industrial IoT" è un termine molto usato dalle aziende e associazioni per descrivere applicazioni IoT usate per produrre beni e servizi. Altre collegano l'IoT a elettrodomestici e wearables. Molti altri hanno focalizzato l'IoT in contesti di Smart Cities e Smart Home, in particolare l'ultimo che analizzeremo in breve successivamente è stato associato anche al nome Home Automation, collegato quindi con gli attuali sistemi domotici. Comunque sia l'uso in cui vengono applicati, è chiaro che l'IoT potrebbe estendere molti aspetti delle nostre vite.

Poiché il numero di dispositivi collegati a Internet cresce, la quantità di traffico che generano è destinato ad aumentare in modo significativo. Per esempio,

Cisco ha stimato che il traffico Internet generato da dispositivi non-PC aumenterà dal 40% nel 2014 al 70% nel 2019 [2]. Cisco ha inoltre previsto che il numero di connessioni "Machine-to-Machine" (che includono industrie, case, salute, automobile ecc.) aumentarà dal 24% nel 2014 al 43% nel 2019.

2 Definizioni differenti, stessi concetti

Nonostante il tanto parlare a livello globale sull'IoT, non c'è una singola definizione universale accettata. Definizioni differenti sono utilizzate dalle diverse parti che descrivono o promuovono una particolare visione di quello che significa IoT, e le sue principali caratteristiche. Alcune definizioni specificano il concetto di Internet o di Internet Protocol (IP), mentre altri, sorprendentemente non lo fanno. Diamo un'occhiata a queste definizioni:

- **Internet Architecture Board (IAB)**, RFC 7452 [3] *"Architectural Considerations in Smart Object Networking"*:

  > Il termine "Internet of Things" (IoT) denota una tendenza in cui un grande numero di dispositivi embedded impiegano i servizi offerti dai protocolli internet. Molti di questi dispositivi, spesso sono chiamati "smart objects", non sono direttamente gestiti da essere umani, ma esistono come componenti in edifici o veicoli o sparsi nell'ambiente.

- **Internet Engineering Task Force (IETF)**, il termine "smart object networking" è comunemnete usato in riferimento all'Internet of Things. In questo contesto, "smart object" sono dispositivi che in genere hanno

[2]'Cisco Visual Networking Index: Forecast and Methodology, 2014-2019." Cisco, May 27, 2015. `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.pdf`

[3]RFC 7452, "Architectural Considerations in Smart Object Networking" (March 2015), https://tools.ietf.org/html/rfc7452

vincoli significativi come energia, memoria e capacità di calcolo limitate[4].

- **International Telecommunication Union (ITU)**, *"Overview of the Internet of Things"* [5], pubblicato nel 2012, discute il concetto di interconnettività, ma non lega in maniera specifica l'IoT a Internet, questo è un estratto:

  > 3.2.2 Internet of Things (IoT): Una struttura globale per la società dell'informazione, abilita servizi avanzati interconnettendo cose (fisicamente e virtualmente) in base esistenti con tecnologie di comunicazione e informazione.

- **IEEE Communications Magazine** [6], collega l'IoT ai servizi cloud:

  > L'Internet of Things (IoT) è un framework nel quale tutte le cose hanno una rappresentazione e una loro presenza in Internet. Più specificatamente, l'Internet of Things ha come scopo quello di offrire nuove applicazioni e servizi facendo da ponte tra il mondo fisico e quello virtuale, in cui comunicazioni Machine-to-Machine rappresentano la comunicazione di base che abilita l'interazione tra le cose e il cloud.

- **The Oxford Dictionaries** [7], offre una definizione concisa che invoca l'Internet come un elemento dell'IoT:

[4] Thaler, Dave, Hannes Tschofenig, and Mary Barnes. "Architectural Considerations in Smart Object Networking." IETF 92 Technical Plenary - IAB RFC 7452. 6 Sept. 2015. Web. https://www.ietf.org/proceedings/92/slides/slides-92-iab-techplenary-2.pdf

[5] 'Int Area Wiki - Internet-of-Things Directorate." IOTDirWiki. IETF, n.d. Web. 06 Sept. 2015. http://trac.tools.ietf.org/area/int/trac/wiki/IOTDirWiki

[6] `http://www.comsoc.org/commag/cfp/internet-thingsm2m-research-standards-next-steps`

[7] "Internet of Things." Oxford Dictionaries, n.d. Web. 6 Sept. 2015. `http://www.oxforddictionaries.com/us/definition/american_english/Internet-of-things`

> Internet of things (sostantivo), l'interconnessione attraverso Internet di dispositivi computazionali embedded negli oggetti di tutti i giorni, abilitati a spedire e ricevere dati.

Tutte queste definizioni descrivono scenari in cui la connessione di rete e la capacità di computazione estende costellazioni di oggetti, dispositivi, sensori, e oggetti di tutti i giorni che non sono ordinariamente considerati "computer"; questo permette ai dispositivi di generare, scambiare, e consumare dati, spesso con il minimo intervento umano. Le varie definizioni dell'IoT non necessariamente sono in disaccordo, piuttosto sottolineano differenti aspetti del fenomeno IoT da differenti punti di vista e casi d'uso.

Tuttavia, le definizioni differenti possono essere fonte di confusione. Una confusione simile è stata vissuta negli ultimi anni riguardo la neutralità della rete e della computazione cloud, dove c'erano differenti interpretazioni dei termini molte volte ostacolo di dialogo. E' probabilmente non necessario pensare a una singola definzione dell'IoT, dovrebbe essere riconosciuto che ci sono molte prospettive differenti che vanno prese in considerazione nelle discussioni.

3 Modelli di comunicazione

Nel Marzo 2015, l'Internet Architecture Board (IAB) ha rilasciato il documento RFC 7452 [8], che spiega l'architettura di rete degli smart object, che delinea un quadro di quattro modelli di comunicazione utilizzati dai dispositivi IoT.

3.1 Comunicazione Device-to-Device

Il modello di comunicazione device-todevice rappresenta due o più dispositivi che sono connessi direttamente e comunicano tra di loro, piuttosto che

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

attraverso un server che fa da intermediario. Questi dispositivi comunicano attraverso diversi tipi rete che include reti IP o Internet. Spesso questo tipo di comunicazione utilizza protocolli come Bluetooth, Z-Wave, o ZigBee per stabilire una connessione device-to-device come mostrato nella figura sotto.

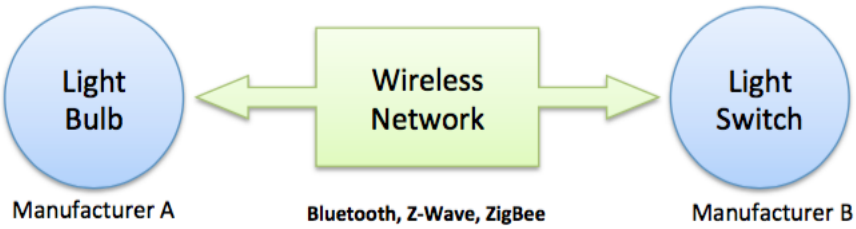

Figura 1.1: Esempio di comunicazione device-to-device.

Questo modello di comunicazione è comunemente usato in applicazioni come sistemi di home automation, che tipicamente utilizzano piccoli pacchetti di dati per la comunicazione tra i dispositivi. Per esempio dispositivi IoT come lampadine, interruttori di luce, termostati, serrature ecc. normalmente utilizzano piccole informazioni tra di loro per effettuare le loro operazioni in uno scenario di home automation, se pensiamo al messaggio di accensione o di spegnimento che può ricevere una lampadina è veramente piccolo.

Questo tipo di comunicazione generalmente avendo una diretta relazione tra i dispositivi, ha bisogno di impostazioni di sicurezza al loro interno in modo da creare un meccanismo di fiducia tra i dispositivi. Questo significa che i produttori di questi dispositivi hanno bisogno di effettuare maggior investimenti in termini di sicurezza per realizzare questi approcci e per implementare uno specifico formato di dati rispetto ad architetture più aperte.

Dal punto di vista dell'utente invece, sorgono problemi di compatibilità, forzando l'utente a scegliere una famiglia di dispositivi che utilizza uno specifico protocollo. Per esempio i dispositivi che utilizzano Z-Wave non sono nativamente compatibili con quelli della famiglia ZigBee. Questo si, limita la scelta, però essere anche un punto a favore perchè scegliendo una famiglia di dispositivi ha la certezza quei dispositivi tendano a comunicare in maniera efficiente tra di loro.

3.2 Comunicazione Device-to-Cloud

Nel modello di comunicazione Device-to-Cloud il dispositivo IoT si connette direttamente a un servizio Internet cloud. Questo approccio utilizza metodi di comunicazione tradizionali come connessioni Wi-Fi o Ethernet per stabilire una connessione tra il dispositivo e la rete IP connessa al cloud.

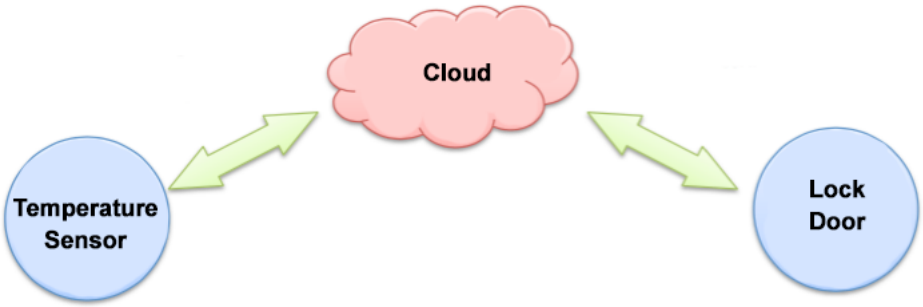

Figura 1.2: Esempio di comunicazione device-to-cloud.

Questo tipo di comunicazione è implementato da alcuni dispositivi IoT famosi sul mercato, come per esempio il Termostato Nest. Il termostato Nest trasmette i dati a un un dabatase cloud dove vengono analizzati per verificare i consumi energetici. Inoltre la connessione con il cloud permette all'utente di utilizzare il termostato da remoto tramite smartphone o tramite interfaccia web, oltre agli update software.

Tuttavia possono nascere problemi di incompatibilità quando si cerca di integrare dispositivi di differenti produttori, perchè molto frequentemente i dispositivi e il servizio cloud non sono dello stesso produttore. Questo limita fortemente l'utente perchè lo vincola a utilizzare un certo tipo di dispositivi dello stesso produttore, oppure utilizzare interfacce web o app differenti per ogni dispositivo di marca differente.

3.3 Comunicazione Device-to-Gateway

Nel modello di comunicazione device-to-gateway, il dispositivo IoT si connette attraverso un Gateway al servizio cloud. Detto semplicemente il gateway

fa da intermediario tra il dispositivo e il cloud fornendo tre ruoli principali.

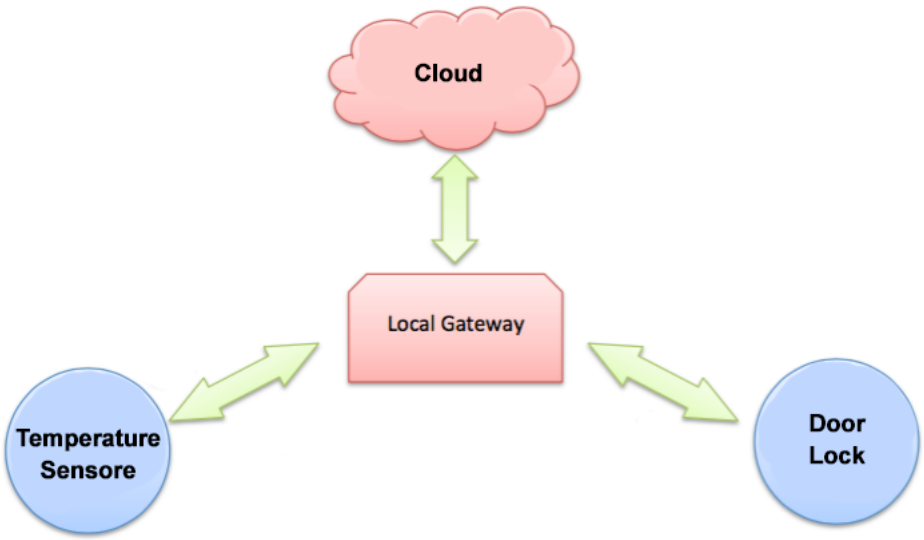

Figura 1.3: Esempio di comunicazione device-to-gateway.

Il primo ruolo è quello di trasformare i dati e normalizzarli, per esempio se il sensore non ha capacità di computazione e l'unica cosa che può fare è spedire il dato "grezzo" sarà allora il gateway che completerà l'informazione per esempio aggiungendo il timestamp e il nome del dispositivo che l'ha generato,oppure i dati generati dai sensori possono essere in diversi formati e quindi è il gateway che si occupa di normalizzarli in un unico formato. Il Gateway quindi acquisisce dati eterogenei provenienti dai vari dispositivi e li converte in un formato standard che sarà comprensibile per la fase successiva nella elaborazione dei dati.

Il secondo ruolo del Gateway è quello di supportare diversi protocolli di comunicazione, perché dispositivi IoT di diversi produttori molto probabilmente utilizzano protocolli di comunicazione differenti. Quindi il Gateway deve supportare protocolli diversi per le connessioni in arrivo, ma anche per le connessioni in uscita, perché tipicamente il Gateway si connette al Cloud. Alcuni dei più popolari protocolli usati in questo contesto sono: ReST, MQTT, CoAP, STOMP e anche SMS. In alcuni casi un Gateway può anche processare i dati e emettere avvisi in tempo reale, quindi non far arrivare una

specifica informazione al cloud se è stato così impostato per alcuni parametri.

Il terzo e ultimo ruolo non per importanza è quello di gestire la sicurezza, il gateway lavora come dispositivo di confine proteggendo i dispositivi IoT dalla rete pubblica e incrementando notevolmente la sicurezza.
Questo modello di comunicazione sembra essere ultimamente il più utilizzato in molti scenari:

- Nei Personal Fitness Tracker, il Gateway locale è l'app di un semplice smartphone che comunica con il dispositivo IoT e spedisce i dati al cloud. Per esempio gli smart band che permettono di tenere traccia del battito cardiaco, sonno, calorie, movimento quotidiano ecc. immagazzinano tutte queste informazioni e le spediscono allo smartphone tramite Bluetooth Low Energy ed è lui che si incarica di connettersi con il cloud e gestire le comunciazioni.
- Un'altro utilizzo molto diffuso è quello dell'Hub negli scenari di Home Automation. L'Hub si connette con i vari dispositivi IoT posizionati in casa e fa da ponte con il cloud. L'Hub più comune al momento in commercio è quello di Samsung chiamato SmartThings Hub, che permette di fare da Gateway con i più famosi dispositivi IoT di produttori di terze parti, supporta comunicazioni Z-Wave e Zigbee e quindi supporta una vasta gamma di dispositivi sul mercato.

3.4 Considerazioni modelli di comunicazione

Questi tre modelli di comunicazione base dimostrano le strategie di progettazione usate per permettere la comunicazione tra i dispositivi IoT. A parte alcune considerazioni tecniche, l'uso di questi modelli è in gran parte influenzato da prodotti open source contro prodotti proprietari che limitano la comunicazioni con i loro standard e la loro visione. Nel caso della comunicazione device-to-gateway, la prima caratteristica è l'abilità di superare le restrizioni del collegamento tra dispositivi di produttori differenti, ciò significa che l'interoperabilità del dispositivo e gli standard aperti sono considerazioni chiave nella progettazione e sviluppo di questi tipi di sistemi.

Dal punto di vista dell'utente, questi modelli di comunicazione aiutano ad illustrare le abilità di connessione di rete di questi dispositivi, permettendo all'utente di raggiungere un miglior accesso ai dati che questi dispositivi generano. Per esempio la connessione di questi dispositivi ad una infrastruttura cloud permette all'utente di poter leggere in maniera dettagliata i dati che vengono generati e poterli confrontare per esempio con i dati generati da dispositivi di altri utenti all'interno dell'infrastruttura, oppure poter settare alcune impostazioni per il risparmio energetico o nell'ambito dell'home automation poter controllare i dispositivi all'interno dell'abitazione tutto da una singola interfaccia. In altre parole, l'architettura di comunicazione è un valore molto importante che apre in base al modello scelto diverse strade con cui utilizzare i dispositivi IoT.

4 Protocolli di comunicazione a livello applicativo

Oltre alla scelta del modello di comunicazione, i dispositivi IoT per poter ricevere o inviare informazioni devono utilizzare un protocollo di comunicazione a livello applicativo. Sono diversi i protocolli di comunicazione utilizzati e possiamo dividerli in due paradigmi:

- request/response (polling), le informazioni vengono richieste e si rimane in attesa di una risposta, generalmente questo paradigma introduce un timer in modo da fare richieste continue per avere sempre il dato aggiornato
- publish/subscribe (push-based), la struttura è formata da una parte che si iscrive a certi topic di cui si vogliono avere aggiornamenti, una parte che pubblica nuove informazioni per certi topic, e una parte che gestisce le iscrizioni e pubblicazioni e si occupa dello scambio di messaggi. In questa tipologia quindi una volta che ci si è registrati a un certo topic si viene notificati per ogni nuova informazione,in questo modo la rete non viene saturata chiedendo continuamente aggiornamenti anche se non ci sono.

4.1 MQTT

MQTT è stato inventato da Andy Stanford-Clark (IBM) e Arlen Nipper (Eurotech) nel 1999, è stato creato con lo scopo di avere un protocollo che usasse una minima quantità di batteria e di banda, con l'idea di collegare gli oleodotti con connessioni satellitari. MQTT è un protocollo di trasporto di messaggi Client Server publish/subscribe, è leggero, open, e progettato per essere semplice da implementare. Queste caratteristiche lo rendono ideale per molte situazioni, che includono vincoli ambientali come per esempio in contesti per le comunicazioni Machine-to-Machine (M2M) e Internet of Things (IoT), dove è necessario una piccola quantità di banda e di batteria da utilizzare.

MQTT è formato da due parti Client e Broker:

- **Client**, include sia il publisher che il subscriber, perchè un client può essere entrambi. Un client MQTT è qualsiasi dispositivo che sia un microcontrolelr o un server che ha una libreria MQTT in esecuzione ed è connesso ad un MQTT broker su qualsiasi tipo di rete. Le librerie client MQTT sono disponibili per una vasta gamma di linguaggi di programmazione per esempio: Android, Arduino, C, C++, C#, Go, iOS, Java, Javascript, Python, .Net ecc.
- **Broker**, è il cuore del protocollo publish/subscribe. In base all'implementazione usata, un broker può gestire migliaia di connessioni client concorrentemente. Il broker è il primo responsabile per la ricezione di tutti i messaggi, li filtra, e decide in base alle iscrizioni a chi destinare i messaggi. Un'altra responsabilità del broker è l'autenticazione e l'autorizzazione dei client. E nella maggior parte dei casi è possibile estendere il broker creando autenticazioni personalizzate e integrazioni con un sistema di backend.

MQTT è un protocollo di comunicazione asincrono che lavora in cima allo stack TCP/IP.

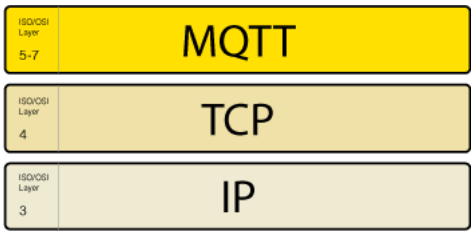

Figura 1.4: Stack MQTT.

La connessione MQTT è sempre tra un client e il broker, un client non è mai connesso ad un'altro client direttamente. La connessione è instanziata quando un client invia un messaggio CONNECT al broker, il broker deve poi rispondere con un CONNACK e lo status della connessione. Una volta che la connessione è stata stabilita, il broker la tiene aperta per tutto il tempo fino a che il client non si disconnette o cade la connessione.

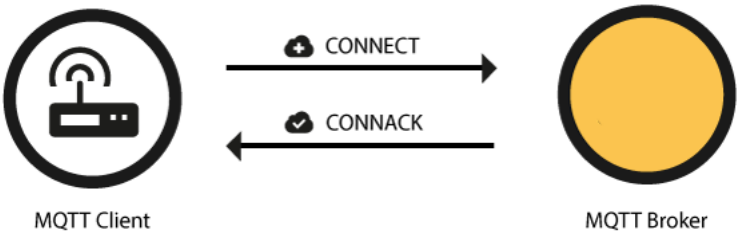

Figura 1.5: Esempio di connessione MQTT.

Nella *Figura 1.6* possiamo dare un'occhiata al messaggio di connessione che viene inviato dal client al broker.
Le opzioni in dettaglio del messaggio:

- **ClientId**, è l'identificativo del client al broker, e deve essere univoco per ogni broker.
- **Clean Session**, indica se il broker deve istanziare questo tipo di connessione con una sessione persistente oppure no, questo vuol dire che se CleanSession è false allora è una sessione persistente e il broker salverà tutte le sottoiscrizioni e i messaggi persi per questo client, altrimenti se è true il broker non salverà nessuna informazione a riguardo ma sarà uno scambio volatile di informazioni, non ne rimarrà traccia.

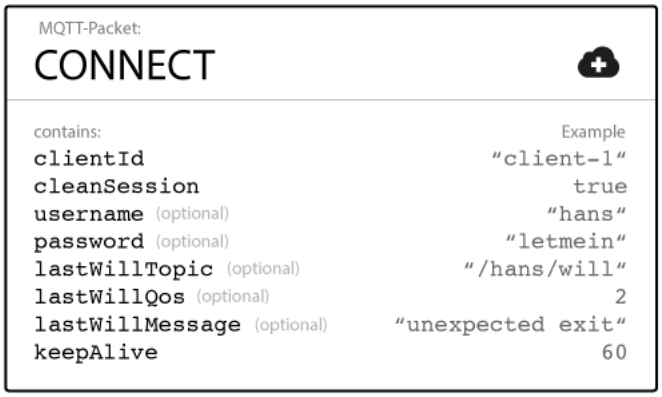

Figura 1.6: Pacchetto messaggio di connessione

- **Username/Password**, i dati di autenticazione del client per poter aumentare la sicurezza e l'affidabilità della connessione.
- **Will Message**, è il messaggio "di testamento", permette di notificare gli altri client quando avviene una disconnessione non voluta.
- **KeepAlive**, è l'intervallo di tempo in cui il client effettua un messaggio di PING al broker e il broker risponde per tenere attiva la connessione.

Il broker nello stesso modo risponderà con un messaggio di CONNACK:

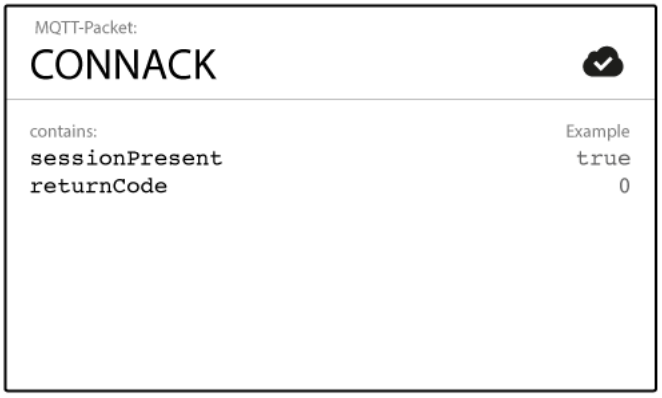

Figura 1.7: Pacchetto messaggio di CONNACK.

Le opzioni in dettaglio del messaggio:

- **Session Present**, indica se il broker ha già una connessione persistente per questo client dovuta a interazioni precedenti, collegata all'opzione CleanSession vista precedentemente.

- **Return Code**, è un valore interno che può assumere un valore da 0 a 5 in base all'esito della richiesta di connessione, per esempio 0 se è stata accettata, da 1 in poi invece se è stata rifiutata e il motivo per cui è stata negata.

Una volta che il client MQTT si è connesso al broker, può iniziare a pubblicare messaggi. Il broker MQTT filtra i messaggi, quindi ogni messaggio deve contenere il topic, che sarà utilizzato dal broker per poter indirizzare il messaggio ai client interessati. Ogni messaggio ha un pyload che contiene l'informazione che si intende spedire, in formato byte. MQTT è data-agnostic, quindi è una scelta del mittente decidere se il dato inviato sarà in binary-data, textual data, XML o JSON.
Questo è il tipo di messaggio di publish nel dettaglio:

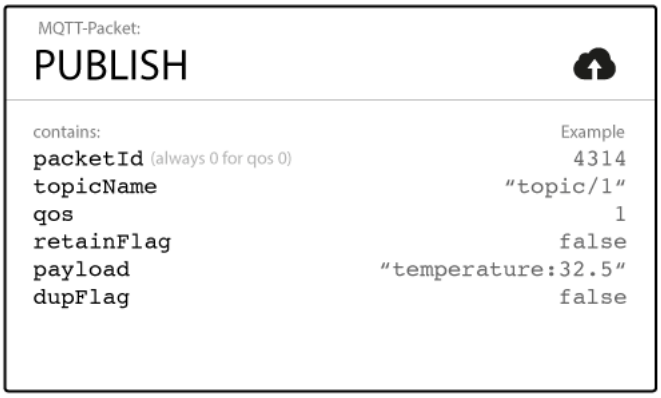

Figura 1.8: Pacchetto messaggio di publish.

Le opzioni in dettaglio sono:

- **Topic Name**, è una semplice stringa che indica la gerarchia del topic, gli slash vengono usati come delimitatori.
- **QoS**, garantisce l'affidabilità del servizio, e può essere di tre tipi:
  - 0 - "Fire and forget": il messaggio è inviato una volta sola senza nessuna conferma di ricezione.
  - 1 - "Delivered at least once": Il messaggio è mandato almeno una volta e richiede una conferma di Ack di ricezione.
  - 2 - "Delivery exactly once": Viene utilizzato il meccanismo "four-way handshake" per assicurarsi che il messaggio sia consegnato esattamente una volta, è il più lento ma anche il più sicuro.

- **Retain-Flag**, questo flag indica se il messaggio deve essere salvato dal broker per questo topic, come ultimo valore conosciuto, in questo modo se un client si iscrive a un topic riceve subito l'ultimo data disponibile.
- **Payload**, è il contenuto vero e proprio del messaggio.
- **Packet Identifier**, è un identificatore unico tra il client e il broker per identificare il messaggio.
- **Dup Flag**, indica che questo messaggio è un duplicato perchè per l'altro messaggio originale non è stato ricevuto un messaggio di conferma.

Quindi quando un client pubblica un messaggio il broker lo riceve, e lo spedisce alle parti interessate, che si sono iscritte a quel determinato topic. Il client che ha inizialmente pubblicato il messaggio non deve più preoccuparsi di niente, perchè da li in poi è il broker che si occuperà del messaggio.

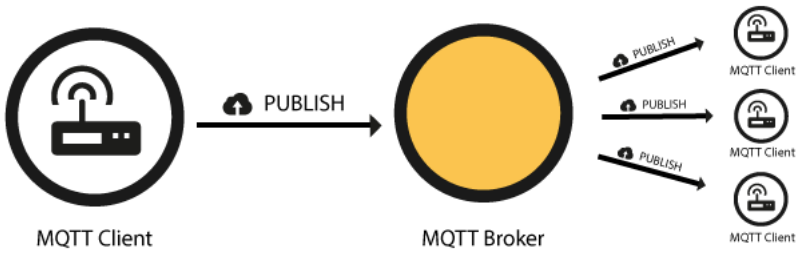

Figura 1.9: Esempio di publicazione di un messaggio.

Allo stesso modo un client può iscriversi a un determinato topic, tramite il comando subscribe, le infomarzioni del pacchetto sono mostrate in *Figura 1.10*.

Le opzioni in dettaglio del pacchetto sono:

- **Packet Identifier**, è un identificativo univoco per identificare il messaggio.
- **List of Subscription**, un messaggio di subscribe può contenere un numero arbitrario di sottoiscrizioni per un client. Ogni sottoiscrizione è una coppia formata da topic e QoS level. Il topic può anche contenere una wildcard che rende possibile iscrivere per più topic di un certo tipo.

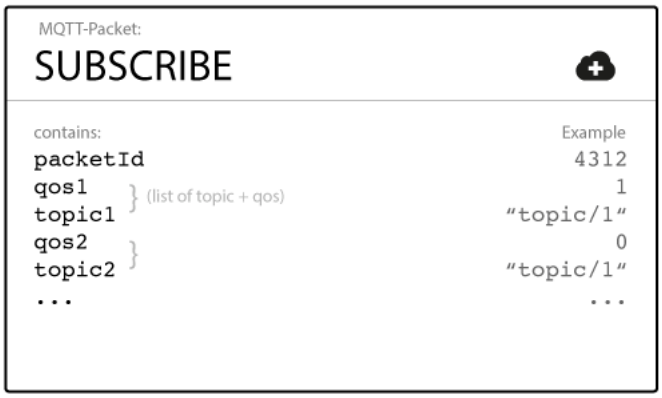

Figura 1.10: Pacchetto messaggio di subscribe.

Oltre a questi messaggi principali esistono anche altri messaggi per esempio per confermare la sottoiscrizione, per annullare la sottoscrizione e la conferma di annullamento della sottoiscrizioni.

Un esempio di comuncazione con un generico sensore di temperatura IoT è questo:

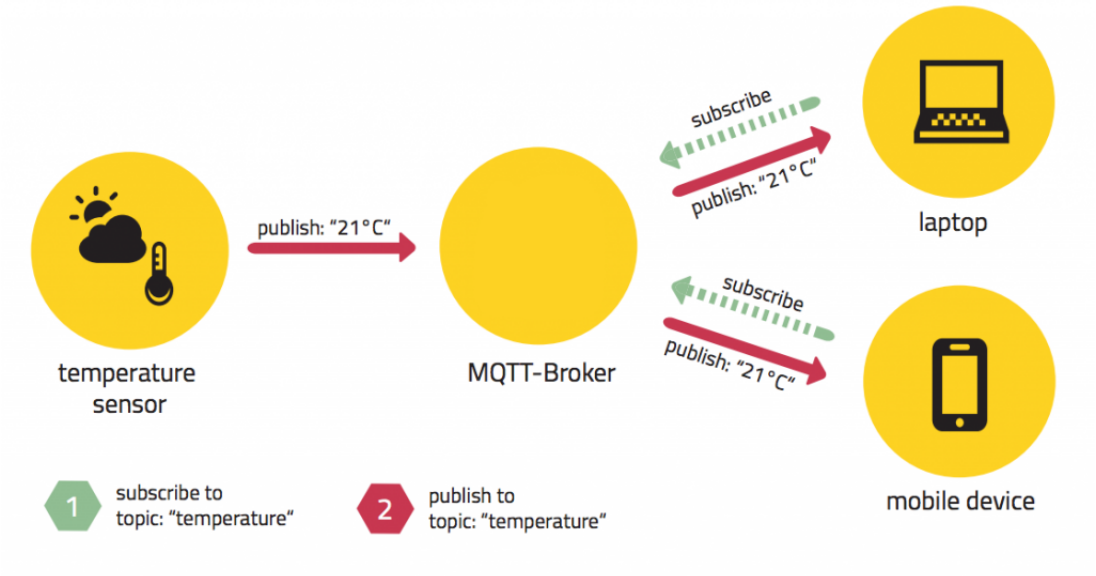

Figura 1.11: Esempio di comunicazione MQTT.

Il protocollo MQTT quindi soddisfa al meglio i requisiti dell'IoT rispetto a un generico protocollo request/response perchè i client non devono richiedere continuamente update, la banda utilizzata è inferiore e così facendo anche il calcolo computazionale si riduce. Inoltre anche se lavora su TCP è progettato per avere un basso "overhead" comparato con altri protocolli a livello applicativo basati su TCP.

4.2 CoAP

Il Constrained Application Protocol (CoAP) è stato progettato dall'Internet Engineering Task Force (IETF) pensato per essere usato in dispositivi con vincoli ristretti come per esempio bassa potenza, poco consumo di banda. E' un protocollo client/server e fornisce un modello di interazione uno-a-uno "request/response" con la possibilità di essere usato per multi-cast.

CoAP è progettato per interoperare con HTTP e RESTful, rendendolo nativamente compatibile con Internet. Lavora sul protocollo di trasporto UDP, che è intrinsecamente e volutamente meno affidabile rispetto al TCP, puntando più sull'invio continuo e ripetitivo di messaggi che sulla qualità di connessione. A differenza del TCP, l'UDP è un protocollo di tipo connectionless, non gestisce il riordinamento dei pacchetti né la ritrasmissione di quelli persi, ed è perciò generalmente considerato di minore affidabilità. In compenso è molto rapido (non c'è latenza per riordino e ritrasmissione) ed efficiente per le applicazioni "leggere" o time-sensitive. Ad esempio, è usato spesso per la trasmissione di informazioni audio-video real-time come nel caso delle trasmissioni Voip. Per esempio un sensore di temperatura può inviare un aggiornamento ogni pochi secondi, anche se nulla è cambiato da una trasmissione all'altra; se un nodo che riceve il valore della temperatura non riceve un aggiornamento, il prossimo arriverà in pochi secondi e sarà probabilmente non molto diverso da quello perso. Le connessioni UDP inoltre abilitano un risveglio rapido della connessione e permettono ai dispositivi di rimanere in uno stato di sleep per lunghi periodo in modo da conservare la batteria.

CoAP è stato progettato per fornire una integrazione trasparente con il Web. Un "cross-proxy" è un elemento della rete che fornisce funzionalità di protocol translation per permettere l'integrazione tra endpoint HTTP e CoAP.

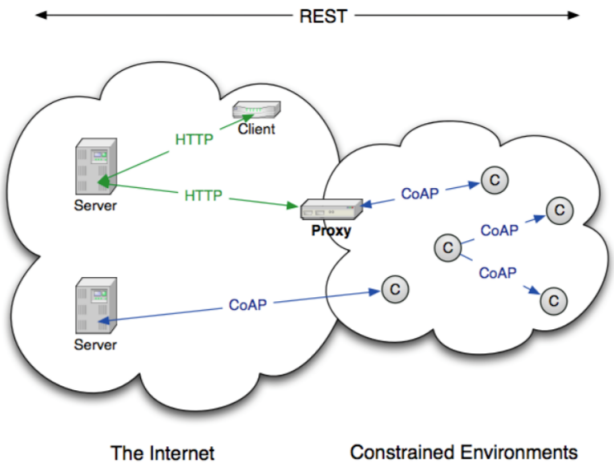

Figura 1.12: Cross-Proxy CoAP.

CoAP ha dovuto colmare alcuni vuoti dovuti dalla natura dell'UDP integrando alcuni meccanismi per raggiungere l'affidabilità. Due byte nell'header di ogni pacchetto indicano il tipo di messaggio e il livello di QoS richiesto. Ci sono quattro tipi di messaggi:

1. **Confirmable**, un messaggio di richiesta che richiede un ack di conferma (ACK). La risposta può essere spedita sia in maniera sincrona quindi insieme all'ACK che se ha bisogno di più tempo di computazione in un secondo momento in maniera asincrona con un messaggio separato.
2. **Non-Confirmable**, un messaggio che non ha bisogno di un ack di conferma.
3. **Acknowledgement**, conferma la ricezione di un messaggio "Confirmable".
4. **Reset**, Conferma la ricezione di un messaggio che non può essere processato.

Un esempio di comunicazione tra un Client e un Server Coap è mostrato in *Figura 1.13*.

CoAP usa l'URI, ossia una stringa di caratteri usata per identificare una risorsa, in modo da identificare i nodi con cui si vuole parlare nella rete.

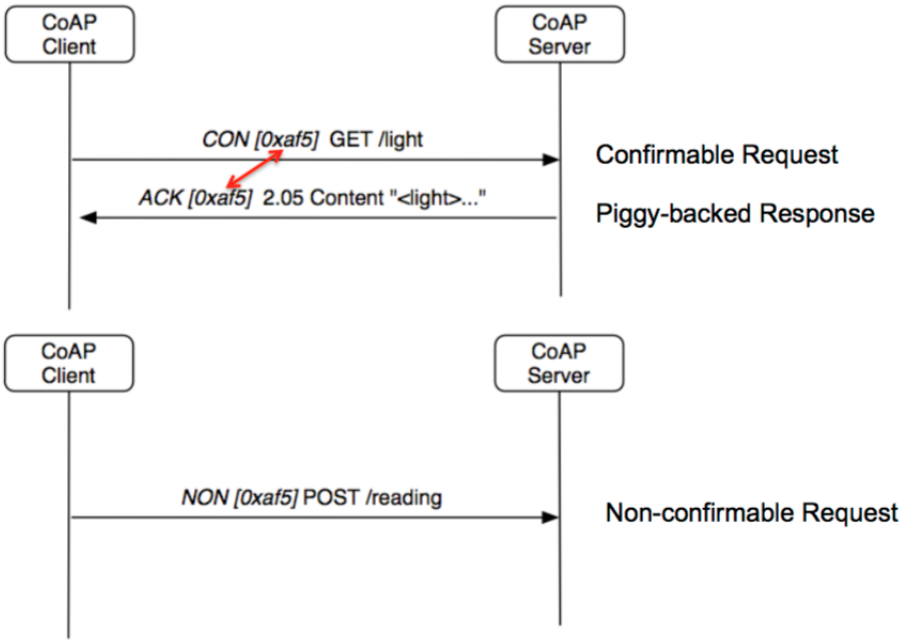

Figura 1.13: Esempio richiesta CoAP.

Questo permette un certo grado di autonomia nei pacchetti dei messaggi che vengono inviati siccome le capacità del dispositivo con cui si vuole parlare viene parzialmente capito dai dettagli dell'URI. In altre parole se abbiamo un sensore alimentato a batteria con un certo URI e un attuatore alimentatore direttamente alla corrente con un altro URI, i nodi nella rete possono essere programmati in modo da aspettarsi lunghi tempi di risposta, informazioni ripetitive e limitate dal sensore alimentato a batteria invece informazioni più ricche e dettagliate e rapidi tempi di risposta dall'attuatore perchè può permettersi avendo più batteria di lavorare senza compromessi.

Nel protocollo CoAP la maggior parte dei messaggi sono spediti e ricevuti attraverso il modello request/response, tuttavia esistono altri modi di operare in modo da permettere ai dispositivi nella rete di essere più disaccoppiati tra di loro. Per esempio CoAP ha semplificato il meccanismo di "observe" simile a quello di publish/subscribe di MQTT, abilitando i nodi nella rete di osservare altri nodi impegnandoli attivamente. Questo tipo di meccanismo è simile all'MQTT broker, a differenza che CoAP non ha nessun broker, quindi non c'è nessuna certezza che possa tenere il messaggio e metterlo in coda per

gli osservatori.

Infine CoAP è stato creato per comunicazioni IoT e M2M, non include al suo interno nessuna caratteristica di sicurezza. Il protocollo proposto a fornire sicurezza è il Datagram Transport Layer Security (DTLS). DTLS lavora in cima allo stack UDP e è analogo al TLS per il TCP. Fornisce autenticazione, integrità dei dati, gestione automatica delle chiavi e algoritmi di criptazione. DLTS però non è stato pensato per l'IoT, perchè non supporta il multicast che è uno dei primi vantaggi di CoAP comparati con altri protocolli di comunicazione. L'handshake del DTLS richiede pacchetti addizionali che incrementano il traffico di rete, e richiedono maggior computazione, così facendo consumano anche più batteria per dispositivi che avrebbero bisogno il minor consumo possibile. Inoltre anche se è HTML- compatibile, CoAP con DTLS può creare una confusione addizionale ai server HTTP a causa della diversa struttura dei pacchetti.

4.3 XMPP

Extensible Messaging and Presence Protocol (XMMP) è stato progettato per essere usato nello scambio di messaggi in chat. E' stato standardizzato dalla IETF più di 10 anni fa, ed è un protocollo che è stato molto usato in Internet nel corso del tempo. Essendo ormai diventato un vecchio protocollo ha fatto fatica a soddisfare le richieste dei nuovi dati delle applicazioni moderne, per questa ragione Google ha smesso di utilizzarlo per il poco supporto. Tuttavia ha riguadagnato attenzione come protocollo di comunicazione nell'IoT. XMPP lavora sul protocollo TCP e fornisce un modello publish/subscribe (asincrono) e anche request/response (sincrono). E' stato progettato per comunicazioni real-time e supporta lo scambio con poca latenza di piccoli messaggi. Supporta il protocollo di sicurezza TLS/SSL, ma non fornisce un livello di qualità QoS che lo rende impraticabile per comunicazioni M2M. XMPP però supporta l'architettura publish/subscribe che è più adatta per l'IoT rispetto a CoAP, e inoltre è un protocollo che è supportato già da tempo quindi è un vantaggio rispetto a MQTT, tuttavia usa messaggi in XML che crea una difficoltà in più per fare il parsing del messaggio e anche capacità di computazione maggiore.

4.4 RESTful Service

Il Representational State Transfer (REST) non è un vero è proprio protocollo ma più uno stile di architettura, è stato introdotto da Roy Fielding nel 2000 ed è stato largamente usato da allora. REST utilizza i metodi HTTP come GET, POST, PUT e DELETE per fornire un sistema di messaggistica orientato alle risorse in cui tutte le azioni posso essere effettuate usando comandi sincroni request/response.

Utilizza l'header di HTTP per indicare il formato di dati che contiene, il contenuto può essere in XML o JSON e dipende dalla configurazione del server HTTP con cui si sta comunicando. REST è già una parte importante dell'IoT perchè è supportato da tutte le piattaforme cloud M2M. Inoltre può essere implementato in applicazioni smarpthone, tablet perchè richiede solo una libreria HTTP che è disponibile in tutti i sistemi operativi. Utilizza TLS/SSL per la sicurezza, ma non è tanto supportato dai sistemi commerciali che a volte forniscono un metodo di autenticazione tramite chiavi da mettere nell'header di ogni richiesta. Il protocollo request/response inoltre utilizza molto la batteria a causa del continuo polling per chiedere nuovi aggiornamenti, un problema risolto dal modello publish/subscribe di MQTT o XMPP. CoAP invece essendo una versione più leggera di REST porta con se alcuni svantaggi dell'architettura request/response. Anche se REST è utilizzato largamente a livello commerciale, è improbabile che diventi un protocollo dominante in applicazioni IoT a causa dei vincoli stretti di comunicazione che hanno i piccoli dispositivi. Però questo tipo di comunicazione potrebbe essere usata insieme ad altri protocolli per esempio in un modello di comunicazione Device-to-Gateway.

4.5 AMQP

Advanced Message Queuing Protocol (AMPQ) è un protocollo che nasce dal settore finanziario. Può utilizzare differenti protocolli di trasporto ma è più affidabile utilizzando il TCP. AMQP fornisce un modello di comunicazione asincrono publish/subscribe, il più grande vantaggio è quello di salvare il

messaggio e spedirlo assicurando una forte affidabilità anche con problemi di connessione. L'affidabilità della consegna del messaggio può essere di tre tipi:

1. Al massimo una volta, il messaggio è spedito una volta sola consegnato oppure no.
2. Almeno una volta, il messaggio verrà sicuramente consegnato una volta, forse più.
3. Esattamente una volta, il messaggio verrà consegnato una sola volta.

Recenti ricerche hanno dimostrato che AMQP ha una bassa percentuale di successo con larghezze di banda ridotte, ma aumenta con l'aumentare della larghezza di banda. Inoltre è stato dimostrato che può spedire una grande quantità di messaggi al secondo, per esempio in uno scenario di 2000 utenti sparsi in 5 continenti può processare 300 milioni di messaggi al giorno. Infine la sicurezza è garantita con TLS/SSL grazie al protocollo TCP.

4.6 Websocket

Il protocollo Websocket è stato sviluppato come parte di HTML 5 per facilitare i canali di comunicazione con il TCP. Websocket non è ne un protocollo request/response ne publish/subscribe, il client inizializza un handshake con il server per stabilire una connessione e iniziare la sessione. In maniera simile a una connessione HTTP, una volta instanziato il canale di comunicazione client e server possono scambiare messaggi in maniera asincrona e bilaterale senza vincoli. Websocket lavora sull'affidabile protocollo TCP, la sessione può essere resa sicura grazie a TLS/SSL. Websocket non è progettato per dispositivi con risorse limitate e poco adatto ad applicazioni IoT, tuttavia è stato progettato per creare una comunicazione real-time, sicura, e con un overhead minimo.

4.7 Considerazioni sui protocolli di comunicazione a livello applicativo

Sono stati presentati i protocolli a livello applicativo che hanno guadagnato più attenzione per l'IoT aggiungendo anche qualche piccolo confronto per

cercare di capire quale possa essere meglio per questo genere di applicazioni. CoAP è l'unico che lavora con UDP, questo lo rende molto leggero, seguito da Websocket che riduce significativamente l'overhead della comunicazione. Ma va presa in considerazione anche la capacità di computazione e comunicazione di questi dispositivi, perchè molti sono vincolati dal tipo di comunicazione e dalla poca computazione e consumo energetico. Considerando quanto detto MQTT che è quello che sta guadagnando più interesse per il modello publish/subscribe che è più adatto per piccoli dispositivi con poca computazione, è stato utilizzato anche per Facebook Messenger.

Per riassumere ci sono diversi fattori che condizionano la scelta del protocollo da utilizzare, per questo penso che il protocollo vada scelto in base alla natura dello scenario che potrebbe avere bisogno delle caratteristiche di uno specifico protocollo e questo non esclude che in scenari complessi possano essere utilizzati più protocolli insieme.

5 Sicurezza dei dispositivi IoT

I dispositivi IoT come appunto dice il nome stesso, lavorano con Internet, e per farlo un utente si aspetta di avere un alto livello di fiducia, nell'applicazione, nel dispositivo, nello scambio di informazioni, quindi in tutto l'ambiente che circonda il dispositivo IoT. Con l'aumentare di dispositivi collegati a Internet, ci sono sempre più possibilità di sfruttare potenziali vulnerabilità di sicurezza, scarsa sicurezza anche in un solo dispositivo IoT potrebbe creare un cyber attacco permettendo di riprogrammare per esempio dispositivi IoT e condizionarne il normale funzionamento.

10 anni fa se avessero detto che sarebbe stato possibile rubare la password dell'account email o altri dati sensibili attraverso il tostapane sarebbe stata un'eresia, adesso invece chiunque staccherebbe la presa del tostapane perchè è la realtà. Questa è l'era dell'Internet of Things in cui anche la sicurezza di quei dispositivi che pensavamo "innocui" può essere un problema.

I dispositivi IoT tendono a differire dai computer tradizionali e dispositivi informatici in tanti modi che mettono in discussione la sicurezza, le sfide a cui possono andare incontro questi dispositivi sono:

- Molti dispositivi IoT, come sensori o oggetti di consumo sono progettati per essere distribuiti su larga scala in maniera molto più grande rispetto ai tradizionali dispositivi collegati a Internet. Questo crea una quantità impredicibile di connessioni tra questi dispositivi, e anche con altri dispositivi di altro tipo, quindi gli strumenti, i metodi, e le strategie associate alla sicurezza dell'IoT devono essere riconsiderate.
- Molti dispositivi IoT saranno un insieme di dispositivi identici o molto simili. Questa omogeneità amplificherà il potenziale impatto di una singola vulnerabilità di sicurezza da tutti quei dispositivi che hanno le stesse caratteristiche. Per esempio la vulnerabilità di un protocollo di comunicazione di una azienda che produce lampadine smart può andare a danneggiare tutti i dispositivi di quel tipo ma anche altri dispositivi che per esempio condividono chiavi di sicurezza.
- Molti dispositivi IoT saranno distribuiti specificando un tempo di vita in anni molto superiore al loro equipaggiamento tecnologico. Inoltre saranno utilizzati in situazioni in cui sarà difficile o impossibile da riconfigurare o aggiornare, oppure possono avere una vita maggiore rispetto a quella dell'azienda che l'ha creato, lasciando il dispositivo senza più supporto. Uno scenario del genere dimostra come il meccanismo di sicurezza implementato potrebbe essere adeguato nel momento in cui è stato rilasciato il prodotto ma non adeguato per tutta la durata di vita del dispositivo. Di conseguenza potrebbe creare delle vulnerabilità che potrebbero persistere per tanto tempo. Questo va in contrasto con i tradizionali computer che sono normalmente aggiornati con update del sistema operativo per tutta la vita del computer per affrontare minacce di sicurezza. Quindi il supporto a lungo termine per i dispositivi IoT è un problema di sicurezza significativo.
- Molti dispositivi IoT sono progettati intenzionalmente senza nessuna possibilità di upgrade, o gli upgrade sono complicati o impraticabili. Per esempio, consideriamo il richiamo di 1.4 milioni di Fiat Chrysler nel 2015 per sistemare un problema di vulnerabilità che ha permesso

un attacco hacker via wireless al dispositivo. Queste macchine devono andare in un concessionario Fiat Chrysler per un upgrade manuale, oppure il proprietario deve effettuare un aggiornamento via chiavetta USB. La realtà è che un'alta percentuale di queste auto non verrà mai aggiornata lasciandola vulnerabile ad attacchi di sicurezza.

- Molti dispositivi IoT operano in un modo in cui l'utente ha una piccolissima visione o nessuna visione di quello che succede al suo interno e dei dati che produce. Questo crea una vulnerabilità di sicurezza quando l'utente crede che un dispositivo IoT esegue certe funzioni quando in realtà potrebbe eseguire funzioni non volute o collezionare più dati di quelli che l'utente vorrebbe. Questi dispositivi potrebbero anche cambiare senza preavviso quando il produttore rilascia un update, lasciandolo vulnerabile a qualsiasi cambiamento il produttore ha eseguito.
- Alcuni Dispositivi IoT sono rilasciati in posti dove la sicurezza fisica è difficile o impossibile da raggiungere. Gli assalitori possono avere un accesso fisico diretto al dispositivo, per questo vanno progettate caratteristiche anti-manomissione per renderli sicuri.
- I futuri dispositivi di IoT potrebbero essere costruiti da privati, vista la continua espansione di Arduino e Raspberry Pi, questi nuovi dispositivi dovrebbero applicare gli stessi standard di sicurezza applicati dalle aziende.

Tutti questi aspetti andranno presi in considerazione dalle aziende e dai privati per la realizzazione di dispositivi IoT.

6 Panorama Tecnologico IoT

Di seguito verranno analizzati i più importanti servizi e prodotti pensati per i dispositivi IoT che si stanno espandendo sempre più in tanti scenari, da quello dell'Home Automation, alle Smart City, alla gestione dell'energia o ha servizi rivolti verso i consumatori. Uno scenario tra i più importanti è quello delle piattaforme o prodotti di Home Automation dove ancora non c'è un vero e proprio standard, e questo rende fertile il terreno a colossi come Intel, Apple, Google, Samsung che vogliono fissare i propri criteri per la Smart

Home cercando di vincolare e influenzare produttori a fare parte del loro sistema.
Oltre questi sistemi progettati per l'Home Automation, Amazon e Microsoft hanno deciso di entrare in questo mercato fornendo una piattaforma cloud che da la possibilità agli sviluppatori e alle aziende di poter realizzare un'intera infrastruttura appoggiata sui loro sistema.

6.1 Home Kit

HomeKit è un framework che immette Apple nel mondo dell'IoT, è stato introdotto da Apple durante la WWDC del 2014, e fornisce agli sviluppatori le API per controllare dispositivi IoT all'interno delle loro app. HomeKit permette di far comunicare differenti dispositivi di differenti produttori tra di loro con un'unica interfaccia grafica. Questo permette di agevolare notevolmente l'utente finale quando ha a che fare con prodotti differenti, per esempio una lampadina con un produttore X ha la sua app per poter essere utilizzata, e un termostato di un produttore Y ha un'altra app proprietaria, tramite HomeKit è possibile realizzare una singola app che gestisca sia la lampadina che il termostato, questo agevola notevolmente l'utilizzo di questi prodotti e incentiva l'utente che non conosce questo mondo a farne parte vista la semplicità di utilizzo.

Il protocollo di HomeKit quindi ha due scopi: quello di creare App terze non direttamente sviluppate dal produttore del dispositivo e quella di integrare diversi prodotti compatibili dalla provenienza più disparata con il risultato di creare un mercato più ampio, con più possibilità di scelta per l'utente finale e senza la necessità che i singoli produttori si accordino tra loro sull'interoperabilità dei propri prodotti.

In pratica Homekit si occuperà di:

- Scoprire gli accessori raggiungibili nell'ambiente circostante e aggiungerli ad un database di configurazione legato all'ambito casalingo chiamato cross-device home configuration database.
- Configurare ed interagire con i dati dei dispositivi disponibili nel database.

- Comunicare con gli accessori configurati per far eseguire dei compiti: ad esempio accendere le luci in una determinata area della casa con un livello di illuminazione prestabilito e con colore specifico.

Il database della configurazione domotica sarà reso disponibile anche a Siri che potrà rispondere a comandi del tipo... "Siri vado a dormire, occupati delle luci" che corrispondono ad azioni preordinate e coordinate in cui vengono assegnati dei valori prestabiliti ai dispositivi distribuiti in zone o stanze specifiche della casa con un coordinamento e raggruppamento logico gestito dall'utente. In pratica Homekit vede la casa come una collezione di accessori - Accessory - per l'automazione a cui possono essere abbinati etichette e raggruppamenti anche su suggerimento non cogente delle App stesse.

La gerarchia supportata da Homekit è la seguente: al livello più alto ci sono le "Homes" che rappresentano le unità principali che possono essere anche multiple (casa principale e casa delle vacanze oppure casa principale e secondaria per gli ospiti), poi ci sono le "Rooms" e cioè la suddivisione opzionale in parti coincidenti con le stanze della casa che non hanno caratteristiche fisiche preordinate ma definiscono gli oggetti che vi appartengono. C'è il livello "Zones" anche questo opzionale che rappresenta un raggruppamento delle "Rooms": si possono aggiungere stanza alle zone come "Piano Terrra" o "Piano Primo" per comandare a Siri di attivare ad esempio un allarme solo in una determinata area della casa. Ad un livello più basso della gerarchia ci sono gli "Accessories" che sono i dispositivi installati in casa e assegnati alle "Rooms" e corrispondono ai prodotti "fisici" per la domotica come sensori, attuatori, interruttori che se non configurati vengono censiti in una stanza indefinita della casa. Infine troviamo i "Services" che corrispondono ai servizi forniti dagli accessori, sia che vengano controllati dagli utenti attraverso Homekit che attraverso delle procedure di aggiornamento autonome. Ad ogni accessorio possono essere attribuiti molteplici servizi come ad esempio, ad un attuatore di apertura di garage si può attribuire l'apertura e la chiusura della basculante e l'accensione o spegnimento delle luci dell'area garage e dei corridoi che vi afferiscono. All'interno dell'API Homekit troviamo diverse classi che permettono di individuare e comandare il singolo accessorio, scoprirlo nella rete, comandare una azione singola o un insieme di azioni, rilevare il

suo stato, programmare una azione con un ritardo temporale o una ripetizione prestabilita.

Quindi Apple non ha creato un proprio Hub, ma ha creato un framework per far interagire questi dispositivi tra di loro in una applicazione unica. HomeKit però come tutti i prodotti Apple non è libero da controllo, infatti se si vuole creare un dispositivo IoT per esempio utilizzando Arduino non è possibile se non si è iscritti al programma MFI Program, e per farlo bisogna essere una azienda registrata nel settore con le dovute licenze.
Esistono già alcuni modi per ovviare questo problema della licenza ma questo dal mio punto di vista è un grosso punto a sfavore in un epoca con tecnologie aperte e accessibili a tutti come Arduino.

6.2 Google Brillo e Weave

Google ha annunciato Brillo nella conferenza per sviluppatori a Maggio 2015, ma non è stata ancora rilasciata e quindi le informazioni a riguardo sono veramente poche.
Brillo è una piattaforma IoT con tre elementi: un embedded OS Android-based, una piattaforma per servizi base e un kit per sviluppatori. Brillo può essere montato tramite codice sorgente su architettura ARM, Intel e MIPS. Il sistema si adatta alle specifiche schede su cui viene installato per supportare la specifica architettura. Il sistema operativo può andare su dispositivi con almeno 128MB di memoria e 32MB di RAM, quindi richiede veramente pochissima computazione. Google sta lavorando con partner per la costruzione di schede che siano compatibili con Brillo e con le sue future versioni.

La piattaforma di servizi base include Weave, che aiuterà i dispositivi a collegarsi in modo sicuro nella rete, e abiliterà gli utenti a collegare questi dispositivi con sistemi mobili o desktop. Metrics è un componente che fa parte dei servizi base della piattaforma e permette di collezionare i dati di utilizzo dai vari dispositivi sempre tramite il permesso dell'utilizzatore. Questi dati poi possono essere verificati e analizzati in una console per capire e scoprire pattern di utilizzo da parte dei consumatori. Gli amministratori inoltre potranno rilasciare aggiornamenti del loro software tramite OTA. Il kit per gli

sviluppatori inoltre fornirà dei tools per lo sviluppo che sono famigliari con l'attuale ADB di Android per realizzare le attuali applicazioni.

Weave è la spina dorsale di comunicazione dei sistemi che saranno alimentati con Brillo per permettere di comunicare tra di loro. Google ha definito Weave come un protocollo cross-platform e una infrastruttura che abilita l'impostazione di un dispositivo tramite uno smartphone e la comunicazione tra questi dispositivi IoT tramite il Cloud, e l'utilizzo invece tramite dispositivi mobile e tramite il web.

E' quindi un protocollo per il rilevamento dei dispositivi, l'autenticazione e l'interazione che è fornita attraversi una libreria client, una mobile tramite un SDK e una web basata sul cloud.

La libreria client è parte del dispositivo che implementa il protocollo, e se il dispositivo è alimentato da Brillo, allora Weave è già presente al suo interno. Gli altri dispositivi invece possono usufruire di queste funzionalità tramite le librerie client. Ogni dispositivo che utilizza il protocollo Weave è automaticamente collegato al servizio cloud. I dispositivi che non sono nella stessa rete o nelle vicinanze utilizzano il cloud per comunicare tra di loro e questa connettività assicura che ogni dispositivo trasmetta i suoi dati al cloud. L'SDK Mobile potrà essere seguito sia su Android che su iOS, e collegherà i dispositivi IoT agli smartphone tramite Weave. Naturalmente Google darà una esperienza di utilizzo superiore su Android rispetto a iOS. Una volta che sarà raggiunta la connessione con un dispositivo IoT, l'app per smartphone potrò utilizzare le API nella stessa rete o tramite cloud per controllare il dispositivo IoT.

Google non ha ancora annunciato tutte le aziende che produrranno schede che saranno alimentate da Brillo, ma per adesso sappiamo che la scheda Edison di Intel è stata già verificata e accettata da Google. Intel Edison è una scheda low-power con una CPU dual core e con Wi-Fi e Bluetooth integrato. Ma non escludiamo che anche Raspberry Pi possa essere supportato.

Non è ancora chiaro però come Weave supporterà gli standard esistenti come Bluetooth Low Energy, ZigBee e Z-Wave.

6.3 Smarthings Hub

A differenza di Google e Apple, Samsung sta partecipando in questo nuovo mercato della Smart Home e dei dispositivi IoT non solo tramite software ma anche tramite hardware. Infatti Samsung ha deciso di gestire la vasta quantità di dispositivi IoT che continuano ad arrivare sul mercato con un Hub che possa connettere tutti questi dispositivi tra di loro in un'unico punto collegato alla rete domestica e a sua volta collegato al cloud.

Supporta dispositivi con connessioni Wi-Fi, ZigBee, Z-Wave e Bluetooth con la nuova versione dell'Hub uscita recentemente.

Non tutti i prodotti in commercio però sono supportati dall'Hub, anche se Samsung sta cercando di integrare sempre più prodotti possibile, tantissime persone in maniera non ufficiale nel mondo open source ci stanno lavorando cercando dei workaround.

Nello store SmartThings di Samsung è possibile anche acquistare una scheda chiamata SmartThings Shield per Arduino, che permette tramite le API di poter realizzare qualsiasi cosa ed estendere ancora di più le possibilità di Smart Home.

Questo da un grosso punto a favore a questa piattaforma, soprattutto per tutte le persone che si stanno cimentando in questa nuova realtà.

6.4 Amazon IoT

Amazon a differenza di Google, Samsung e Apple ha scelto una strada diversa per entrare nel mercato dell'IoT, e l'ha fatto realizzando una piattaforma cloud che consente di connettere i dispositivi IoT in maniera semplice e permette a questi dispositivi di poter interagire con applicazioni cloud e con altri dispositivi in maniera sicura. AWS IoT può supportare miliardi di dispositivi e trilioni di messaggi, e può processare e indirizzare questi messaggi sia ad ad endpoint AWS che ad altri dispositivi in maniera efficiente e sicura. AWS IoT rende inoltre semplice il collegamento con i tanti servizi AWS, come Lambda, Kinesis, S3, Machine Learning, DynamoDB per poter realizzare un'intera infrastruttura che permette di processare e analizzare i dati generati da questi

dispositivi e poter interagire con loro.

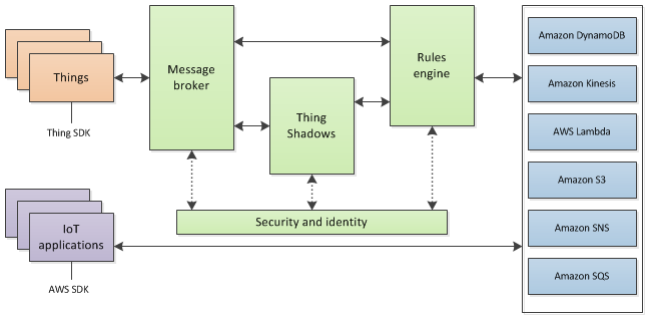

Figura 1.14: Struttura AWS IoT.

Il concetto principale di questa piattaforma IoT è lo "**stato**" del dispositivo. I dispositivi vengono chiamati "**things**" e hanno la possibilità di inviare il proprio stato **pubblicando** un messaggio al **message broker** attraverso i **topic**. Il broker consegna i messaggi ricevuti a tutti i clienti che si sono **iscritti** al topic specifico. In questa descrizione è facile riconoscere la logica del protocollo MQTT e il pattern publish/subscribe, infatti AWS IoT è fortemente basata su questo protocollo.

Lo stato del dispositivo è strettamente collegato all'oggetto **thing shadow** che ha il compito di salvare e fornire quando viene richiesto lo stato del dispositivo. Un'applicazione può richiedere di cambiare lo stato del dispositivo, la richiesta quindi arriverà alla thing shadow e sarà replicata alla thing connessa dal message broker.

Dal punto di vista implementativo, il thing shadow è un documento JSON e significa che il payload per lo scambio di messaggio è anch'esso in JSON. AWS IoT permette ai dispositivi di accedere al cloud utilizzando un protocollo standard come HTTP e MQTT, inoltre fornisce alcuni SDK per semplificare la vita agli sviluppatori. Prima di tutto un SDK C per sistemi embedded che può essere usato su differenti hardware e sistemi operativi, un SDK per NodeJS e infine anche uno per Arduino Yun.

La sicurezza in AWS IoT è fornita dal protocollo TLS quindi la comunicazione tra il message broker e il client sono autenticati usando la mutua autenticazione. Il certificato può essere creato, attivato e revocato utilizzando AWS CLI oppure la console AWS online. Una volta creati i certificati è necessario collegare il certificato a una e una sola Thing e il servizio non può essere utilizzato senza i certificati.
Come detto sopra MQTT è il protocollo ufficiale supportato da AWS IoT, che però utilizza con alcune limitazioni, per esempio non supporta la conservazione dei messaggi, le sessioni persistenti e il QoS di livello 2. E' supportato anche HTTP anche se limitato alla sola pubblicazione dei messaggi usando le API REST (solo con il metodo POST).
Infine AWS IoT è un servizio a pagamento come tutti i servizi AWS, ma offre un servizio di prova gratuito limitato a 250.000 messaggi scambiati.

6.5 Hub IoT Azure

Come Amazon anche Microsoft ha deciso di entrare nel mercato dell'IoT offrendo una piattaforma cloud, e ha aggiunto alla lista dei suoi servizi Azure anche l'Hub IoT. E' un servizio che abilita una comunicazione bidirezionale tra due dispositivi e un sistema di back-end cloud. Il canale di comunicazione è affidabile e sicuro e l'autenticazione è per dispositivo usando le credenziali di accesso.

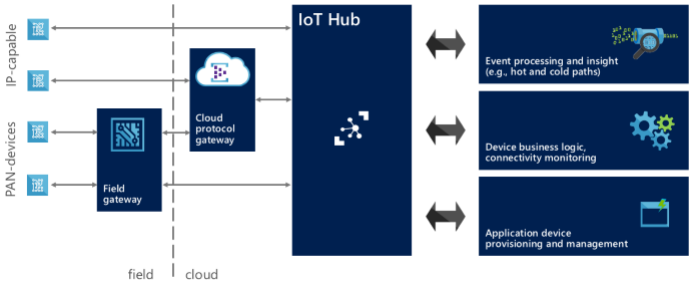

Figura 1.15: Struttura Azure Hub IoT.

Grazie alla sua natura bidirezionale, i messaggi tra i dispositivi e il cloud viaggiano in entrambe le direzioni nel canale di comunicazione creato. Ogni dispositivo ha due endpoint per interagire con IoT Hub:

- D2C (device to cloud), il dispositivo utilizza questo endpoint per spedire i messaggi al cloud, utilizzando dati di telemetria sia come risultato di un comando ricevuto sia come richiesta di esecuzione.
- C2D (cloud to device), il dispositivo riceve i comandi da questo endpoint per eseguire le azioni richieste. L'Hub IoT genera un feedback a livello applicativo per confermare che il comando è stato acquisito dal dispositivo e sta per essere eseguito.

Nel lato cloud, l'IoT Hub ha due endpoint simili:

- C2D (cloud to device), il sistema di backend può usare questo endpoint per spedire messaggi (per esempio comandi) ai dispositivi. L'endpoint agisce come una coda e ogni messaggio ha un TTL (Time to Live) quindi sarà rimosso dalla code se il timeout scade (è utile per avere comandi che siano eseguiti in un breve periodo di tempo e non che siano eseguiti troppo tardi quando un dispositivo offline ritorna online e l'esecuzione non è più necessaria perchè potrebbe essere dannosa). Il sistema di backend può ricevere un messaggio di conferma o di mancata consegna per capire se il dispositivo ha ricevuto o no il comando.
- D2C (device to cloud), è un Event Hubs compatibile endpoint usato dal sistema backend per recuperare i messaggi dai dispositivi. "Event Hubs compatibile", significa che possiamo usare un Event Hub client per ricevere messaggi da questo endpoint.

L'IoT Hub ha un **identity registry** dove vengono memorizzate tutte le informazioni fornite dai dispositivi. Queste informazioni non sono collegate ai dati dei dispositivi, ma all'identità e all'autenticazione. Fornisce informazioni di monitoraggio come stati di connessione (connesso/disconnesso) e l'ultimo tempo di attività, è possibile anche abilitare o disabilitare i dispositivi con questo registro. Per questo l'IoT Hub espone un'altro endpoint per permettere la gestione di queste informazioni.
E' possibile utilizzare l'IoT Hub utilizzando protocolli standard open come HTPP e AMQP e di recente è stato aggiunto il supporto anche a MQTT. Come Amazon anche Azure IoT Hub fornisce SDK per semplificare il lavoro degli sviluppatori, per .Net, Java, NodeJS e anche C.

La connessione stabilita tra i dispositivi e l'IoT Hub è basata su TLS, quindi la comunicazione è criptata per garantire la sicurezza dei dati, e il server è autenticato grazie al suo certificato che viene passato al dispositivo durante il TLS handshaking. Poi tramite l'access control l'IoT Hub può definire i permessi di scrittura, lettura ecc. per i vari dispositivi tra i vari endpoint. Inoltre l'autenticazione è fornita dall'IoT Hub verificando un token spedito dal dispositivo.
Anche questo servizio è a pagamento, e come AWS IoT fornisce un servizio di prova, in cui è possibile collegare fino a 10 dispositivi e permette in tutto uno scambio di 8000 messaggi al giorno. Poco inferiore quindi alla quota gratuita di AWS IoT.

Capitolo 2

Open IoT Middleware

La tesi riguarda la costruzione di un sistema ibrido aperto e non proprietario in grado di miscelare tecnologie cloud-based con tecnologie fortemente situate per la gestione dei dispositivi IoT applicabile in diversi scenari come per esempio quello di Home Automation, aziendale, o anche di Smart City.

Il termine non proprietario indica che non sono stati utilizzati tecnologie o protocolli o sistemi ai quali non è possibile apportare modifiche, il termine aperto è stato usato perchè c'è stata l'intenzione fin dall'inizio di creare un sistema sempre espandibile con nuove tecnologie o dispositivi in grado di aggiungere nuove funzionalità. Inizialmente però è stato testato il servizio proprietario di Amazon, AWS IoT per vedere cosa offriva e come sarebbe cambiato lo sviluppo della piattaforma con e successivamente senza rendendo la piattaforma open e non proprietaria.

L'obbiettivo è stato quello di realizzare una piattaforma ibrida che potesse svolgere non solo funzioni di monitoraggio e controllo per i dispositivi IoT, ma che desse la possibilità di poter interagire con questi dispositivi, e di effettuare computazioni disembodied in modo da estendere le funzionalità che il singolo sensore/attuatore da solo non supporta.

In fase di realizzazione progettuale sono stati inizialmente valutati i requisiti funzionali e non che una struttura di questo tipo dovrebbe avere come funzionalità base per un corretto funzionamento in diversi scenari, successi-

vamente è stato valutato il modello di comunicazione migliore e il protocollo di comunicazione livello applicativo da utilizzare per la comunicazione tra dispositivi IoT.

Essendo parte della piattaforma potenzialmente realizzabile con tecnologie cloud-based sono stati presi in considerazione servizi disponibili in commercio come Amazon AWS IoT e Microsoft Azure IoT Hub per poterli utilizzare e accelerare lo sviluppo della piattaforma. Infine fatte queste considerazioni è stata progettata l'architettura hardware e software e sono stati presi i componenti hardware necessari.

1 Requisiti funzionali e non

Per individuare le funzionalità generali del sistema sono stati analizzati i caratteri che il sistema avrebbe dovuto avere per essere utilizzato in diversi scenari IoT. Sono quindi stati definiti i seguenti punti:

- **Possibilità del controllo da parte dell'utente tramite interfaccia web**, l'utente potrà aggiungere o rimuovere dispositivi IoT in maniera totalmente dinamica, e le sue modifiche saranno subito prese in carico dal sistema e quindi il cambiamento sarà in run-time.
- **Possibilità di collegamento dei sensori/attuatori al cloud**, i sensori e gli attuatori saranno collegati in maniera diretta o indiretta tramite gateway alla piattaforma cloud in modo da poter scambiare informazioni in run-time.
- **Possibilità di trasmissione dei dati generati dai sensori nel cloud**, i sensori invieranno i dati al cloud in maniera diretta o tramite un gateway.
- **Possibilità di interagire con gli attuatori**,l'utente tramite interfaccia grafica potrà inviare comandi agli attuatori aggiunti nel suo profilo.
- **Possibilità di avere una cronologia dei dati per specifico sensore**, i dati inviati dai sensori al cloud saranno memorizzati, in modo da fornire per ogni sensore una cronologia e poter essere analizzati.

- **Possibilità di avere i dati dei sensori in real time in una dashboard**, sarà presente una pagina principale in cui viusalizzare in real-time il valore più recente inviato da ogni sensore.

2 Scelte Progettuali

In questo paragrafo verranno spiegate le scelte progettuali fatte dopo aver analizzato lo stato dell'arte nel mondo dell'IoT in ogni sua parte, dal modello di comunicazione ai protocolli di comunicazione a livello applicativo, alle soluzioni valide attualmente sul mercato utilizzabili e per concludere a possibili idee aggiuntive che è possibile realizzare.

2.1 Modello di comunicazione scelto

In base ai componenti recuperati e alle funzioni che dovrebbe avere il sistema, è stato possibile scegliere il modello di comunicazione da utilizzare tra i tre modelli di comunicazione presentati nel primo capitolo. Il modello scelto è quello device-to-gateway, in cui i dispositivi IoT si collegano a un gateway locale che si occupa di spedire le informazioni ricevute al cloud.

E' stato scelto questo tipo di modello di comunicazione perchè i dispositivi recuperati per il progetto non hanno alcun tipo di connessione internet e soprattutto non hanno capacità di computazione quindi hanno bisogno di essere collegati a un gateway che si occuperà di spedire le informazioni al cloud e di proteggere questi dispositivi dalla rete pubblica in modo da isolarli solo nella rete locale in cui operano.

2.2 Protocollo di comunicazione scelto

Dopo aver analizzato nel capitolo precedente i protocolli di comunicazione a livello applicativo più usati, è stato scelto di utilizzare il protocollo MQTT. In seguito all'analisi fatta nel capitolo precedente su questo protocollo, possiamo aggiungere che negli ultimi anni è stato il protocollo che ha avuto maggiore crescita e popolarità nel mondo dell'IoT, grazie appunto alle sue

caratteristiche e al modello publish/subscribe. E' stato specificatamente progettato per essere utilizzato in dispositivi con risorse limitate e basso utilizzo di banda.

La forza principale di MQTT è la semplicità, ha solo 5 metodi API, perfetto per scenari in cui bisogna inviare la temperatura o eseguire comandi rapidi, è anche molto utile per collegare dispositivi tra di loro come per esempio collegare Arduino a un servizio web con MQTT.

Oltre a questo protocollo si è scelto anche di utilizzare il protocollo WebSocket per un aspetto di comunicazione dell'interfaccia web.

2.3 Amazon AWS IoT vs Microsoft Azure Hub IoT

Amazon AWS IoT e Microsoft Azure Hub IoT sono due piattaforme cloud che permettono agli sviluppatori/aziende di poter creare una infrastruttura dinamica senza dover progettare da zero il sistema. Hanno sviluppato le loro piattaforme con scelte differenti a partire dai protocolli di comunicazione usati, AMQP per Microsoft e MQTT per Amazon (solo di recente anche Microsoft ha deciso di supportare MQTT) forse per la grande ascesa che questo protocollo sta avendo nel mondo dell'IoT.

Le due piattaforme a parte la loro struttura e il loro modo di gestire i dispositivi IoT mostrato nel capitolo precedente, offrono in generale un servizio simile, infatti entrambe forniscono agli sviluppatori degli SDK per diverse piattaforme di sviluppo, ed entrambe offrono un livello di sicurezza TLS.

Una differenza invece molto importante e fondamentale riguarda i costi della piattaforma. Amazon permette di utilizzare il servizio gratuitamente fino ad un massimo di 250.000 messaggi inviati per 12 mesi, dopo di che il traffico viene calcolato in base al volume di messaggi scambiati, per esempio 5$ per un milione di messaggi, quindi si paga solo quello che si consuma. Microsoft invece fornisce 8000 messaggi gratis al giorno ma una volta superata quella soglia il servizio a un costo di 50$ al mese fino a un massimo 400.000 messaggi al giorno, quindi anche se si consumano meno di 400.000 messaggi o se un

solo giorno si sfora quella cifra si è comunque costretti l'intero importo. Come si può vedere quindi la piattaforma di Amazon è molto più dinamica nei costi perchè ti permette di pagare effettivamente per quello che consumi, e inoltre AWS IoT può essere integrato con tutti i servizi AWS di Amazon che possono estendere le funzionalità del sistema in maniera rapida e semplice.

Concludendo credo che Amazon AWS IoT sia al momento la scelta migliore.

2.4 Regole

Le funzioni che dovrà avere il sistema elencate in breve precedentemente sono la base del sistema, ossia sono tutte le funzioni che un sistema di questo tipo dovrebbe avere per un normale funzionamento. Ma come dice appunto il titolo della tesi questo studio ha il compito di valutare la computazione Embodied e Disembodied, perchè la prossima generazione di sistemi computazionali sarà un mix tra scenari pervasivi e cloud computing, con componenti intelligenti e non che potrebbero non avere informazioni o capacità di calcolo sufficienti per il corretto funzionamento in questi scenari, e da soli sarebbero inutilizzabili.

Proviamo ad immaginare la creazione di un semplice scenario di Home Automation in cui all'esterno dell'abitazione è posto un sensore di luminosità, che riesce a rilevare la quantità di luce nell'ambiente, in giardino sono presenti delle luci per l'illuminazione nottuarna e infine ogni finestra è collegata con dei servomotori che possono aprire o chiudere le fessure delle persiane. Lo scenario prevede che quando il sensore di luminosità rileva una forte quantità di luce tutti i servo motori posizionati sulle persiane devono aprire le fessure per far entrare la luce in casa o viceversa chiuderle per non farla entrare, invece quando il sensore di luminosità rileva una bassissima quantità di luce deve accendere le luci in giardino.

In questo scenario il sensore di luminosità ha il solo compito di inviare il valore di luce rilevato al sistema, quindi non ha nessuna conoscenza di dove si trova o di dove sono posizionate le luci in giardino o i servo motori sulle persiane, e non sa i suoi dati che rileva a cosa servono, non ha nessuna capa-

cità di computazione l'unica cosa che sa è di inviare i dati che rileva al cloud. Lo stesso discorso vale per i servo motori o le lampadine sono solo in attesa di ricevere il comando.

Questo esempio appena descritto ha fornito gli spunti per progettare una funzione che potesse rendere il sistema più dinamico e inoltre questo tipo di funzioni sta diventando sempre più necessaria in questi scenari. Ossia l'inserimento di regole, che permettono di eseguire certe operazioni automaticamente se vengono soddisfatte certe condizioni, così da avere una computazione disembodied nella logica del sistema.

3 Architettura

Dopo aver analizzato lo stato dell'arte dell'IoT e fatto le dovute scelte progettuali per un IoT Middleware, di seguito vengono mostrate le scelte architetturali hardware, di sistema e software.

3.1 Architettura Hardware

Nelle soluzioni IoT bisogna avere a che fare con una vasta quantità di dispositivi disposti nel mondo fisico, la loro diversa natura rende difficile l'integrazione di tutti questi dispositivi in un unico sistema, e l'unico modo è quello di avere una architettura intermediaria che fa da proxy tra il mondo di dispositivi sul campo e il data center con cui devono interagire. Quello di cui c'è bisogno è un IoT Gateway. Proviamo ad esporre alcuni aspetti importanti che ci spingono ad avere un architettura con un Gateway.

Principalmente questo genere di dispositivi ha capacità molto limitate in termini di connessione di rete. I sensori o gli attuatori possono avere in genere una connessione Bluetooth Low Energey (BLE), oppure altri potrebbero utilizzare per esempio il protocollo ZigBee. Ma tutti questi protocolli hanno una solo cosa in comune, non possono connettersi a reti più grandi come le reti WAN o Internet. C'è bisogno di un Gateway che fornisce a questi dispositivi un singolo punto di contatto con il mondo esterno, via WiFi, GSM o altri

tipi di connessioni.

Bisogna tenere in considerazione che un gateway non è solo un proxy vuoto che ha l'unico compito di spedire i dati al servizio di backend. Il gateway può effettuare computazioni situate, quindi embodied, prima di spedire i dati al cloud, e si occupa anche di avere un singolo punto di monitoraggio nel sistema senza dover monitorare ogni sensore/attuatore singolarmente.

L'architettura nella figura sottostante è quella più comune in cui il gateway scambia informazioni con i sensori/attuatori tramite connessioni wireless come Bluetooth o Zigbee:

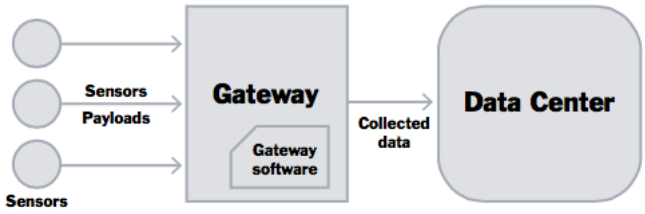

Figura 2.1: Architettura IoT Gateway.

Bisogna tenere in considerazione però che ci sono molte variazioni di questa architettura, infatti una di queste variazioni è quella usata in questo progetto, in cui i sensori/attuatori sono posizionati direttamente sul gateway.

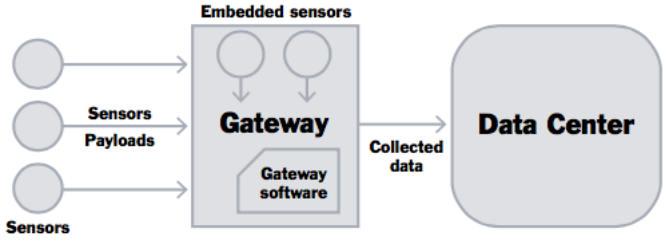

Figura 2.2: Architettura IoT Gateway con sensori/attuatori embedded.

Questa soluzione architetturale alternativa permette di evitare eventuali problemi di comunicazione con ZigBee o Bluetooth per ricevere e inviare le informazioni ai sensori/attuatori, è quindi una implementazione più semplice e

immediata, però non sempre è utilizzabile. Per esempio quando i sensori/attuatori non possono essere posizionati nello stesso posto fisico del gateway oppure per collegarli è necessario un cablaggio troppo lungo che l'ambiente non permette allora l'unica soluzione rimane quella del collegamento wireless. Questa tipologia architetturale non implica che i sensori/attuatori siano fisicamente nella stessa "scatola" ma che il collegamento viene effettuato tramite i pin della scheda per esempio con una scheda Raspberry Pi, i collegamenti avvengono sui 20 pin GPIO.

Quindi queste due strutture sono molto simili e la scelta tra le due dipende solo dalla posizione dei sensori/attuatori nell'ambiente e come si vede nella *Figura 2.2* una implementazione non esclude l'altra, infatti possono coesistere nella stessa architettura.

L'IoT Gateway è un termine e una architettura utilizzata soprattuto nelle aziende, ma negli ultimi anni questo tipo di architettura sta prendendo sempre più piede nel mondo dell'IoT e nell'Home Automation, infatti con un nome diverso il gateway prende il nome di Smart Hub. Lo Smart Hub viene così definito: "E' essenzialmente un modo per centralizzare in un unico punto il controllo wireless di luci, termostati, allarmi di fumo, sensori di movimento, e qualsiasi altro dispositivo o elettrodomestico smart. Uno Smart Hub agisce come "middleman" ossia come intermediario nel sistema facilitando la comunicazione tra tutti i vari dispositivi e abilitare il controllo di essi in un unico punto."

Questo tipo di architettura IoT Gateway/Smart Hub è una delle soluzioni commerciali più usate nell'Home Automation, tra le tante spicca quella di Samsung con SmartThings Hub, in cui come spiegato nel capitolo precedente permette di collegare dispositivi di marche differenti e poterli controllare tutti tramite una singola interfaccia.

Hardware necessario

Dopo aver definito l'architettura e le scelte progettuali sono stati recuperati nel laboratorio dell'università i componenti adatti per questo tipo di archi-

tettura.

Raspberry Pi, è uno dei più popolari computer di piccole dimensioni che offre un completa piattaforma Linux ad un costo veramente basso. Raspberry Pi è stato realizzato per stimolare l'insegnamento di base dell'informatica e della programmazione nelle scuole, ma è diventata nel corso del tempo vista la potenza, il costo e le dimensioni ridotte un punto di riferimento per molti progetti, tra i quali l'IoT. Per il collegamento di sensori o attuatori vengono utilizzati i connettori GPIO sulla scheda, ed è stata fornita insieme alla scheda anche un connettore Adafruit Pi Cobbler per agevolare le connessioni sui pin GPIO con una breadboard così da evitare saldature per velocizzare la prototipazione. Sono state rilasciate diverse versioni di questa scheda nel corso degli anni, quella che useremo è la versione Raspberry Pi 2 Model B.

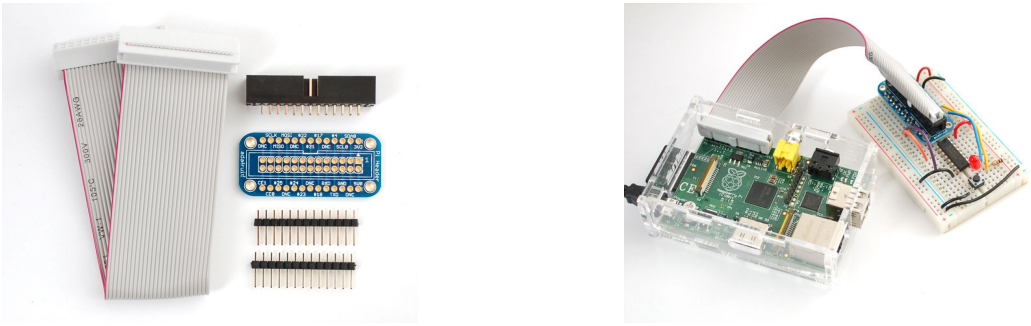

Figura 2.3: Raspberry Pi e Adafruit Pi Cobbler.

Arduino, è la scheda elettronica più famosa al mondo, ha rivoluzionato il modo in cui creare prototipi per scopi hobbistici didattici e professionali. Per la facilità di utilizzo, per le piccole dimensioni e per il costo basso è ideale per progetti IoT. Sono state rilasciate diverse versioni nel corso degli anni quella che useremo è Arduino Uno, e vista la mancanza di una porta di Ethernet o di una connessione Wireless nella scheda aggiungeremo l'Ethernet Shield, che aggiunge all'Arduino funzionalità di rete.

Figura 2.4: Arduino Uno e Ethernet Shield.

Sensori, sono stati recuperati semplici ma molto utili sensori, che è possibile trovare nei più comuni scenari di Home Automation, Smart City, Smart Factories. I sensori utilizzati sono un sensore di Temperatura LM35, una fotoresistenza in grado di rilevare l'intensità di luce che lo colpisce, e un sensore ad ultrasuoni HC-SR04 che riesce a rilevare oggetti all'interno di un raggio di azione.

Figura 2.5: Rispettivamente, sensore di Temperatura LM35, Fotoresistenza, Sensore ad ultrasioni HC-SR04

Attuatori, sono stati recuperati dei semplici Led, paragonabili a lampadine all'interno di una abitazione per esempio, e un servo motore HS-53.

Figura 2.6: Led e Servomotore

3.2 Architettura di Sistema

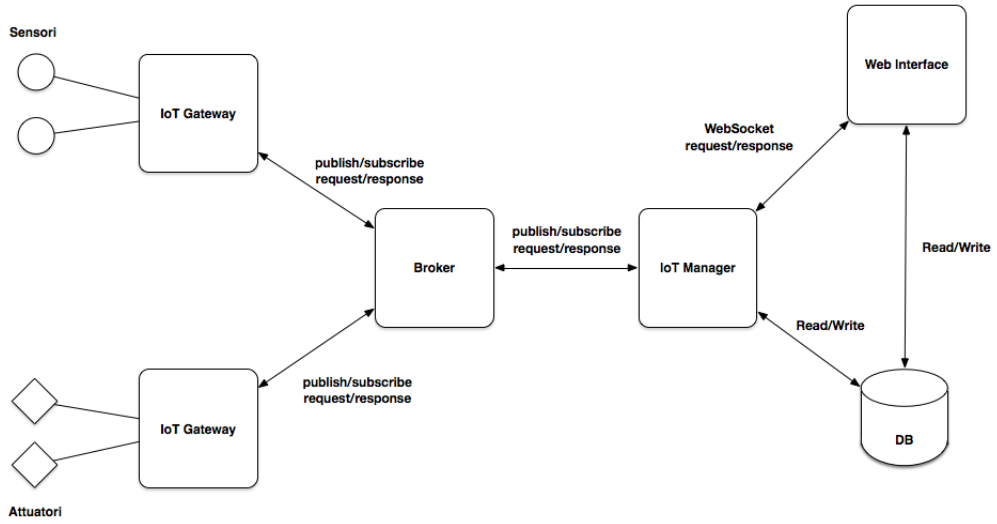

Figura 2.7: Architettura di sistema.

Nella *Figura 2.7* è possibile vedere l'architettura di sistema, ora verranno analizzate nel dettaglio tutte le parti che lo compongono.

- **Sensori**, sono piccoli dispositivi hardware che possono misurare e percepire l'ambiente che li circonda. Ne esistono di diversi tipi come sensori di temperatura, umidità pressione dell'aria, di movimento ecc.
- **Attuatori**, a differenza dei sensori che possono essere di piccoli dimensioni, gli attuatori sono generalmente più grandi, ricevono i comandi

dal gateway ed eseguono quello che gli viene chiesto, un esempio più classico di attuatore sono le luci, o la serratura elettrica di una porta.

- **IoT Gateway**, il software del gateway è il cuore della parte client, è colui che si occupa di collezionare i dati dai vari sensori/attuatori processarli e se necessario arricchirli con informazioni e infine spedirli al cloud. Il software del gateway può essere progettato con proprietà di fault tolerance che abilitano il sistema a continuare la sua normale attività o in maniera ridotta anche all'avvenimento di errori o problemi. Un esempio pratico è quello realizzato da Samsung per lo SmartThings Hub, dove la nuova versione dell'Hub uscita da poco in commercio è stato attrezzata con un vano batterie che permette all'Hub di lavorare per circa 10 ore anche in mancanza di alimentazione elettrica, e permette all'app per tablet/smartphone di potersi connettere con una rete locale e continuare il suo funzionamento. I sensori/attuatori sono predisposti per generare dati con alta frequenza, è importante quindi che il gateway decida quando richiedere il dato allo specifico sensore in modo da non generare più informazioni di quelle che servono, un esempio classico potrebbe essere quello del sensore di temperatura, che può rilevare la temperatura ogni millisecondo, ma è veramente necessario avere una così alta frequenza di rilevamento? Nella maggior parte dei casi, a parte in scenari in cui c'è bisogno di un altissimo livello di precisione, è più che sufficiente una lettura ogni qualche secondo. Quindi è il gateway che tramite polling richiede i dati ai sensori con le frequenze impostate in base alla precisione richiesta dallo scenario. Infine il gateway comunica con un broker che fa da intermediario con i tanti gateway che ci possono essere in un scenario e gestisce le connessioni con il cloud.
- **Broker**, è colui che si occupa di gestire tutte le connessioni, facendo da intermediario tra i gateway e il cloud.
- **IoT Manager**, è il componente del sistema che si occupa di disaccoppiare la parte embodied e disembodeid, riceverà i dati dei gateway dal broker e avrà la possibilità di leggere e scrivere queste informazioni nel database. Infine ha una connessione con l'interfaccia web per permettere all'utente di comandare gli attuatori.

- **Database**, è un generico database che permetterà la memorizzazione delle informazioni spedite dai sensori, e gestirà i sensori/attuatori aggiunti nel profilo utente.
- **Web Interface**, interfaccia web per permettere all'utente di gestire i dispositivi IoT e poterli controllare.

3.3 Architettura Software

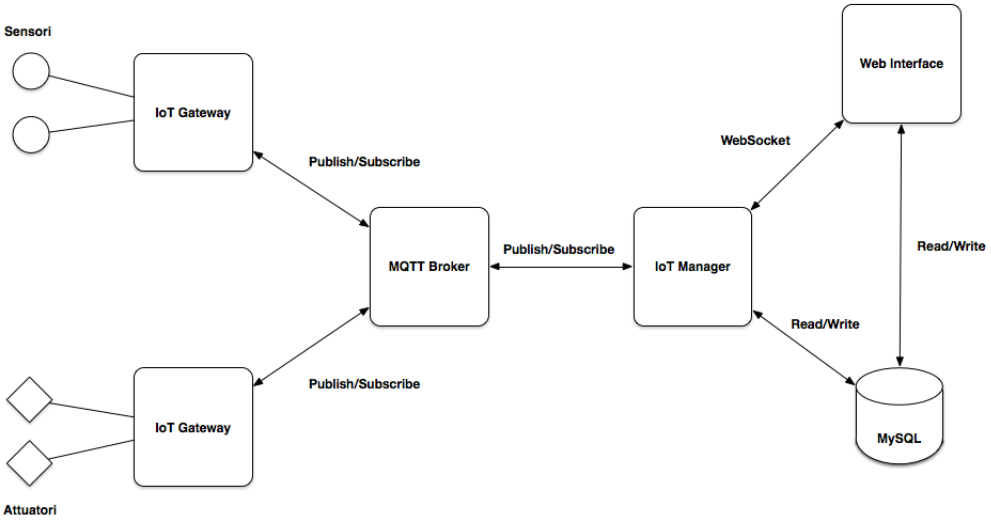

Figura 2.8: Architettura Software.

Nella *Figura 2.8* è possibile vedere l'architettura software, ora verranno analizzate nel dettaglio tutte le parti che lo compongono e come sono state implementate.

- **IoT Gateway**, il gateway può essere implementato in diversi modi in base alla scheda utilizzata, per esempio utilizzando il Raspberry Pi verrà usato Python mentre utilizzando Arduino viene utilizzato il linguaggio C in versione "alleggerita". Indipendentemente dalla scheda hardware utilizzata per la realizzazione del gateway, in qualunque caso il protocollo di comunicazione livello applicativo scelto è MQTT che è possibile trovarlo per tantissimi linguaggi di programmazione tra cui i due appena citati. Il gateway attraverso il protocollo MQTT potrà

connettersi all'MQTT broker e utilizzando il modello publish/subscribe potrà pubblicare i dati generati dai sensori e iscriversi per specifici topic per esempio per il controllo dell'accensione e lo spegnimento dei led. In base allo specifico linguaggio di programmazione utilizzato come detto precedentemente il gateway interrogherà i sensori tramite polling per avere l'ultimo dato disponibile con una frequenza scelta in base alle necessità.

- **MQTT Broker**, il broker è il cuore del protocollo MQTT, è colui che si occupa di gestire le connessioni tra i vari client connessi, e di gestire le pubblicazione e le sottoiscrizioni ai topic. Nel corso degli anni e vista la continua ascesa di questo protocollo di comunicazione sono stati implementati diversi broker MQTT sia open source che commerciali, altri invece sono nati come servizio, come quello di Amazon AWS IoT. Per accellerare i tempi di sviluppo e vista l'opportunità di sfruttare il servizio di prova gratuito di AWS IoT si è deciso di affidare il broker MQTT al servizio di Amazon. L'implementazione è piuttosto semplice, per poterla utilizzare dal pannello di controllo è necessario creare una Thing, creare un certificato di sicurezza per questa Thing e attribuirgli i permessi necessari allo scopo. Ogni Thing ha bisogno del proprio certificato di sicurezza e non è possibile condividere tra diverse Thing. Il certificato verrà utilizzato per effettuare una connessione sicura ai server di Amazon dai client, senza il certificato non è possibile instaurare una connessione con il servizio.
- **IoT Manager**, realizzato utilizzando Node JS è il cuore dell'IoT Middleware, e ha diversi ruoli:
  - Connessione con il database MySQL, tramite l'utillizzo di una libreria MySQL per NodeJS, verrà instaurato un collegamento che permettere all'IoT Manager di leggere e scrivere le informazioni, come per esempio salvare il valore dei dati in arrivo dai sensori.
  - Connessione con il broker MQTT, tramite l'utilizzo di una libreria client MQTT verrà instaurata una connessione con il broker MQTT, al quale si effettueranno iscrizioni o pubblicazioni in base ai sensori/attuatori aggiunti dall'utente nel suo profilo.
  - Connessione WebSocket, tramite l'utilizzo di una libreria websocket per Node JS verrà aperto un socket server e rimarrà in attesa

della richiesta di collegamento con l'interfaccia web. Quando arriverà la richiesta di connessione verrà aperto un canale di comunicazione bidireazionale per permettere all'utente di poter effettuare richieste in run-time nel sistema e per permettere al sistema di notificare l'utente di eventuali cambiamenti senza dover ricaricare la pagina.

- **Database MySQL**, il database realizzato in MySQL verrà utilizzato per la memorizzazione dei sensori/attuatori aggiunti dall'utente collegati al suo profilo, e per il salvataggio di tutte le impostazioni inerenti.
- **Interfaccia Web**, realizzata in HTML 5, CSS 3, Javascript, PHP, permette all'utente di poter interagire con il sistema, gestendo i dispositivi e potendo controllare i dati da loro generati.

Rimozione di Amazon AWS IoT

Come detto all'inizio di questo capitolo, lo scopo era quello di realizzare un middleware open e non proprietario, per questo dopo aver provato Amazon AWS IoT come broker per la piattaforma e averne analizzato ogni singolo aspetto si è deciso di rimuovere il servizio per alcuni motivi:

- Il primo motivo è dovuto al fatto che siccome AWS IoT accetta solamente connessione certificate da Amazon tramite certificati creati dal pannello di controllo limita l'utilizzo di alcuni dispositivi. Un esempio è quello della scheda Arduino Uno o precedenti, che Amazon tramite i suoi SDK di AWS IoT non supporta, supporta però una scheda Arduino più potente e più costosa come Arduino Yun. E' stato chiesto ad Amazon come mai non avesse supportato anche altre schede Arduino, e la risposta è stata che l'SDK per Arduino Yun è stato costruito utilizzando la sua architettura dual-processor, questo vuol dire che i loro SDK hanno bisogno di più potenza rispetto a schede più economiche come Arduino Uno. Inoltre non esistono librerie MQTT che supportano l'utilizzo di certificati con Arduino Uno, quindi questo è uno dei motivi per cui si è deciso di rimuovere il servizio.
- Il secondo motivo è dovuto al fatto che dopo l'utilizzo costante di qualche ora del servizio in fase di sviluppo, sono stati scambiati più di 5000

messaggi, contro i 250.000 messaggi offerti dal pacchetto di prova di AWS IoT. Quindi siccome lo sviluppo sarebbe durato molto di più di qualche ora per evitare eventuali costi è stato rimosso.

Capitolo 3

Implementazione

Tutto il progetto è accessibile al seguente indirizzo: `https://github.com/Piero87/IoTMiddleware`, in questo capitolo però, verranno descritte nel dettaglio alcune parti fondamentali tra cui il lato client che non avendo una interfaccia utente hanno bisogno di una descrizione dettagliata per poter utilizzare il sistema. Successivamente verrà analizzato l'IoT Manager che si occupa di gestire i dispositivi lato cloud e infine una piccola parte dell'interfaccia web.

1 Gateway

Il gateway è colui che fa da intermediario tra i sensori/attuatori e la piattaforma cloud, è possibile implementarlo in diverse schede elettroniche due tra le più famose sono Raspberry Pi e Arduino. La scelta tra le due schede si potrebbe fare in base alla potenza che si vuole dare al gateway, il Raspberry Pi essendo molto più potente rispetto all'Arduino permette una maggiore computazione e fornisce anche maggiore memoria, le schede Arduino invece sono più adatte come Gateway per pochi sensori o attuatori vista il poco spazio disponibile per poter eseguire il codice. Questo non esclude che in alcuni scenari sia possibile utilizzarle entrambe per esempio l'Arduino può essere usato collegando sensori che forniscono informazioni che non necessitano di tanta computazione, mentre il Raspberry Pi potrebbe essere utilizzato con sensori/attuatori che necessitano di più potenza e capacità di calcolo. Vedremo ora nel dettaglio come implementare un semplice Gateway con sensori e attuatori sia su Arduino che su Raspberry Pi.

1.1 Gateway su Arduino

Di seguito viene mostrata l'implementazione di un semplice Arduino Gateway a cui è connesso un sensore di temperatura e un led, il codice che segue è quello che dovrebbe scrivere l'utente/sviluppatore per implementarlo e per potersi interfacciare con la piattaforma cloud.

```
1  byte mac[]    = {  0x90, 0xA2, 0xDA, 0x0D, 0x1E, 0x97 };
2
3  char servername[]="ec2-52-33-1-162.us-west-2.compute.
       amazonaws.com";
4
5  int lastTemperature;
6  unsigned long lastTime;
7  int sensorPin = 0;
8  int ledPin = 7;
9
10 EthernetClient ethClient;
11 PubSubClient client(servername, 1883, callback, ethClient);
12
13 void setup()
14 {
15   Serial.begin(9600);
16   pinMode(ledPin, OUTPUT);
17
18   if (Ethernet.begin(mac) != 0)
19     {
20        if (client.connect("ArduinoGateway")) {
21           lastTemperature=0;
22           lastTime=0;
23           client.subscribe("/home/light");
24        }
25     }
26 }
```

Dopo aver connesso fisicamente il sensore di luminosità al pin analogico 0 e il led al pin digitale 7 vengono inizializzati gli stessi valori all'interno del codice. Il metodo `setup()` è il primo metodo chiamato di default in Arduino, e qui viene inizializzata l'Ethernet Shield utilizzando l'indirizzo MAC scritto sulla scheda, e solo nel caso in cui ci sia una connessione attiva, viene effettuato il collegamento del client MQTT verso il broker all'jndirizzo specificato

nella variabile `servername`. Il nome `ArduinoGateway` scelto per il client, è un identificativo per l'MQTT broker quindi non ci possono essere due client con lo stesso nome collegati allo stesso broker. Infine vengono inizializzate due variabili temporanee che vedremo successivamente e il client si iscrive al topic `/home/light` per essere avvisato tramite il metodo di callback dal broker quando ci sono dei cambiamenti su quel topic.

```
1  void loop()
2  {
3    int reading = analogRead(sensorPin);
4    int temperature = reading * 0.48875;
5    if (temperature != lastTemperature) {
6
7      if(millis()>(lastTime+3000)) {
8
9        StaticJsonBuffer<200> jsonBuffer;
10       JsonObject& root = jsonBuffer.createObject();
11       root["temperature"] = temperature;
12       char buffer[256];
13       root.printTo(buffer, sizeof(buffer));
14       client.publish("/home/sensor/temperature",buffer);
15       lastTemperature = temperature;
16       lastTime = millis();
17     }
18   }
19   client.loop();
20 }
```

Il metodo `loop()` è il secondo metodo chiamato di default da Arduino e viene ripetuto ciclicamente. Qui viene letta tramite polling la temperatura del sensore, e solo se il valore è cambiato rispetto al precedente e solo se sono passati almeno 3 secondi dall'ultimo publishing viene pubblicato il nuovo valore al broker al topic `/home/sensor/temperature` dopo averlo codificato in un pacchetto JSON. Questo controllo evita di saturare la rete con connessioni non necessarie, molto importante per dispositivi IoT che possono avere poca batteria e banda. Un'altra soluzione per preservare la batteria del dispositivo potrebbe essere quella di eseguire un comando di `sleep()` tra le varie letture del sensore.

```
1  void callback(char* topic, byte* payload, unsigned int length
       ) {
2
3    Serial.println(topic);
4    String msg = toString(payload, length);
5    Serial.println(msg);
6    StaticJsonBuffer<200> jsonBuffer;
7    JsonObject& root = jsonBuffer.parseObject(msg);
8
9    if (root.containsKey("on")) {
10     Serial.println("on recevied");
11     digitalWrite(ledPin, HIGH);
12   } else if (root.containsKey("off")) {
13     Serial.println("off recevied");
14     digitalWrite(ledPin, LOW);
15   }
16 }
```

Il metodo `callback()`, è il metodo di default della libreria MQTT utilizzata su Arduino in cui riceveremo messaggi dal broker MQTT. Questo metodo ha tre parametri, il primo è `topic`, che ci permette di sapere da quale topic è arrivato il messaggio. Il secondo parametro è `payload` che contiene l'informazione ricevuta in byte, dopo averlo trasformato in un formato di semplice lettura viene verificato se contiene la chiave `on`, in quel caso il led collegato alla scheda Arduino viene acceso, altrimenti se contiene la chiave `off` viene spento. Per parlare ulteriormente del caso disembodied anche qui possiamo vedere che se si volesse aggiungere un nuovo led, basterebbe solo collegare il pin alla scheda Arduino, e aggiungere una riga di accensione e una spegnimento insieme a quelle appena viste, così facendo rientrerebbe in automatico nel gruppo già interfacciato con il cloud al topic `/home/light`.

Questo piccolo esempio mostra in maniera chiara come l'utente/sviluppatore deve implementare un gateway per poter interfacciare i suoi sensori/attuatori alla piattaforma cloud su Arduino. Infine andranno creati gli stessi dispositivi anche nel profilo dell'interfaccia utente nel cloud, per permettere all'IoT Manager di interagire con questi dispositivi.

1.2 Gateway su Raspberry Pi

Di seguito viene mostrata l'implementazione di un semplice Raspberry Pi Gateway a cui è connesso un Led, il codice che segue è quello che dovrebbe scrivere l'utente/sviluppatore per implementarlo e per potersi interfacciare con la piattaforma cloud.

```
1 ledPin = 25
2 gpio.setmode(gpio.BCM)
3 gpio.setup(ledPin, gpio.OUT)
4 gpio.output(ledPin, gpio.LOW)
```

Il led viene collegato al Pin 25 utilizzando l'Adafruit Pi Cobbler, e viene impostato inizialmente a `LOW`, quindi spento.

```
1 mqttc = mqtt.Client(client_id="RaspberryPi_Led_Blu",
    clean_session=True)
2 mqttc.on_connect = on_connect
3 mqttc.on_message = on_message
4 mqttc.connect("ec2-52-33-1-162.us-west-2.compute.amazonaws.
    com", port=1883)
5 mqttc.loop_forever()
```

Qui viene mostrato come l'MQTT client si collega all'indirizzo dell'MQTT Broker utilizzando un id univoco.

```
1 def on_connect(client, userdata, flags, rc):
2       print("Connected with result code "+str(rc))
3       client.subscribe("home/light",1)
4       client.subscribe("home/led_blue_switch",1)
```

Appena collegato il led si iscrive a due topic, il primo è `home/light`, che visto il nome possiamo vederlo come tutte le luci all'interno della casa, quindi rafforzerà la parte disembodied in cui non siamo a conoscenza ne di quante luci sono collegate ne di dove sono. Il secondo topic è più specifico e per esempio se è l'unica luce blu ci permette di accendere o spegnere solo quella luce.

```
1 def on_message(client, userdata, msg):
2
3   json_data = msg.payload.decode('utf-8')
4   print("Message received: "+json_data)
5   parsed_json = json.loads(json_data)
```

```
6
7     if "on" in parsed_json:
8           gpio.output(ledPin, gpio.HIGH)
9     elif "off" in parsed_json:
10          gpio.output(ledPin, gpio.LOW)
```

Il metodo `on_message` verrà chiamato ogni volta che riceviamo un messaggio dal broker, siccome indifferentemente dal topic a cui ci siamo iscritti la lampadina si può accendere o spegnere in base alla chiave *on* oppure *off* non andiamo a differenziare il topic, ma controlliamo solo la chiave e accendiamo o spegniamo la lampadina di conseguenza.

2 IoT Manager

L'IoT Manager è implementato in Node.js ed è il cuore dell'IoT Middleware, è colui che si occupa di gestire i dispositivi aggiunti dall'utente attraverso l'interfaccia web nel cloud e di tenere attive le connessioni cruciali per il funzionamento dell'infrastruttura. Vediamo nel dettaglio alcune parti chiave del codice.

```
1  var mysql_client = require('./node_modules/mysql');
2
3  var mysql = mysql_client.createConnection({
4    host     : 'localhost',
5    user     : 'username',
6    password : 'password',
7    database : 'db'
8  });
9
10 mysql.connect(function(err) {
11   if (err) {
12     console.log('Error Connecting MySQL: ' + err);
13     return;
14   } else {
15     console.log("MySQL Connected.");
16   }
17 });
```

Qui viene mostrato il collegamento dell'IoT Manager con il database utilizzando una libreria di MySQL per Node.js.

```
var mqtt    = require('mqtt');
var mqtt_client  = mqtt.connect('mqtt://localhost:1883');

mqtt_client.on('connect', function() {
    console.log('MQTT Connected.');
    mysql.query('SELECT name FROM topics WHERE `type` = "
       publish"', function (error, results, fields) {

      results.forEach(function(row){
      mqtt_client.subscribe(row['name']);
    });
  });
});
```

Qui viene mostrato innanzi tutto il collegamento all'MQTT Broker che in questo caso per semplicità è stato installato sulla stessa macchina EC2 che hosta l'IoT Manager quindi l'indirizzo di collegamento al broker è `localhost`, nulla vieta che in una diversa implementazione il broker possa essere su un'altra macchina o affidato ad un'altro servizio come era la prima implementazione con Amazon AWS IoT spiegata nel capitolo precedente. Quando il collegamento con il broker è instaurato l'IoT Manager recupera con una query al database tutti i topic di tipo `publish` aggiunti ai sensori/attuatori attraverso l'interfaccia web e si scrive tramite il comando di `subscribe()`, in questo modo quando un sensore/attuatore effettuerà un publish l'IoT Manager riceverà i dati e li salverà nel database.

```
var io = require('./node_modules/socket.io').listen(5000);
io.sockets.on('connection', function (socket) {

  console.log('User connected to Socket.io.');

  socket.on('publish_key', function (data) {
        console.log('Publishing to '+data.topic+' only Key: '
           +data.key);
    var post = {};
    post[data.key] = "";
    publish(data.topic,post);
    });

    socket.on('publish_key_with_value', function (data) {
```

```
14            console.log('Publishing to '+data.topic+' with Key:
                  '+data.key+' and Value: '+data.key_val);
15      var post = {};
16      post[data.key] = data.key_val;
17            publish(data.topic,post);
18      });
```

Questo è l'ultimo collegamento che effettua l'IoT Manager dove apre un Socket server utilizzando la libreria *socket.io* e attende che l'utente acceda all'interfaccia web, non appena questo accade tramite la WebSocket si instaura un canale di comunicazione bidirezionale in cui l'utente può tramite l'interfaccia HTML eseguire comandi di `publish` per determinati topic in run-time.

```
1  mqtt_client.on('message', function(topic, payload) {
2
3    console.log('message', topic, payload.toString());
4    var obj = JSON.parse(payload);
5
6    mysql.query('SELECT K.id,K.name FROM topics as T,
        table_keys as K WHERE T.type = "publish" AND T.name = ?
        AND K.id_topic = T.id',topic, function (error, results,
        fields) {
7       results.forEach(function(row){
8       if (row['name'] in obj) {
9         var post  = {id_key: row['id'], value: obj[row['name'
              ]]};
10        var query = mysql.query('INSERT INTO
              sensors_actuators_data SET ?', post, function(err,
               result) {
11        });
12      }
13    });
14   });
15 });
```

Questo metodo viene chiamato ogni volta che si riceve un messaggio con il protocollo MQTT dal broker, qui viene salvato il valore inviato per esempio dal sensore per il relativo topic e chiave anche questi decisi tramite l'interfaccia grafica web.

2.1 Regole

Un aspetto molto importante all'interno dell'IoT Manager da vedere nel dettaglio sono le Regole. Attraverso l'interfaccia grafica è possibile aggiungere una regola collegandola a un topic, per una certa chiave, e una certa condizione per esempio nell'immagine sottostante viene mostrato il risultato di una regola aggiunta.

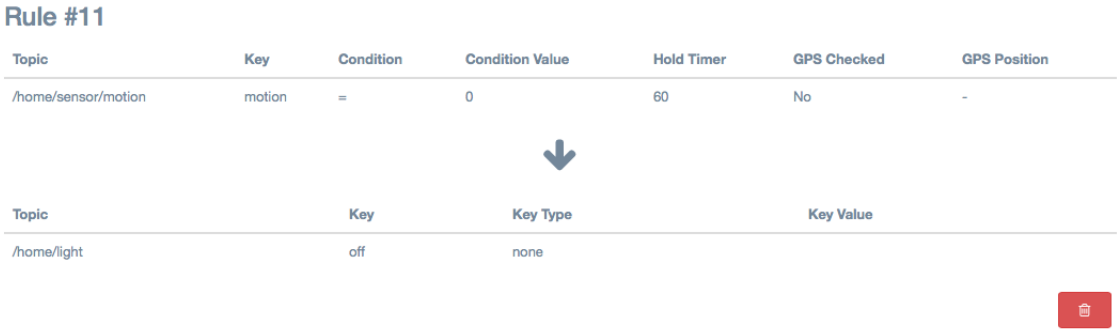

Figura 3.1: Regola.

Come è possibile vedere la regola aggiunta nell'interfaccia dice che quando la chiave `motion` ricevuta tramite il topic `/home/sensor/motion` mantiene un valore pari a 0 per 60 secondi, viene pubblicata la chiave `off` al topic `/home/light`. Una regola come questa per esempio potrebbe essere applicata in uno scenario di Home Automation in cui quando il sensore di movimento per 60 secondi non rileva nessuno all'interno della stanza spegne le luci. Vediamo ora come è stato possibile implementare una regola di questo tipo.

```
1 mqtt_client.on('message', function(topic, payload) {
2 ...
3 mysql.query('SELECT T1.id as id_topic, T1.name as topic_name,
    K1.id as id_key, K1.name as key_name,T2.id as
    id_topic_result, T2.name as topic_name_result,K2.id as
    id_key_result, K2.name as key_name_result, R.id as id_rule
    , R.condition_type, R.condition_value, R.hold_timer, R.
    key_type_result, R.key_value_result, R.gps_checked, R.
    gps_value, R.enabled FROM topics as T1, topics as T2,
    table_keys as K1, table_keys as K2, rules as R WHERE T1.id
     = R.id_topic AND K1.id = R.id_key AND T2.id = R.
    id_topic_result AND K2.id = R.id_key_result AND T1.type =
```

```
    "publish" AND T1.name = ?',topic, function (error, results
    , fields) {
4       results.forEach(function(row){
5         check_condition_rule(obj, row);
6     });
7   });
8 });
```

Sempre all'interno del metodo visto precedentemente che riceve i messaggi dal broker MQTT, verrà controllato nel database se per il topic appena arrivato è associata una regola, in quel caso viene chiamato il metodo `check_condition_rule()`.

```
1 function check_condition_rule(obj, row) {
2
3     var value_to_check = 0;
4       if (row['gps_checked'] == 0) {
5         value_to_check = parseFloat(obj[row['key_name']]);
6       } else {
7         value_to_check = haversineDistance(row['gps_value'].
              split(";"), obj[row['key_name']].split(";"));
8       }
9
10    if (row['condition_type'] == '<') {
11        if (value_to_check < parseFloat(row['condition_value'
              ])) {
12          if (row['enabled'] == 1) {
13            if (row['hold_timer'] == 0) {
14              var post = {};
15            post[row['key_name_result']] = row['
                  key_type_result'] == 'none' ? "" : row['
                  key_value_result'];
16            publish(row['topic_name_result'],post);
17            enableDisableRule(0,row['id_rule']);
18            } else {
19              addTimerRule (obj,row);
20            }
21          }
22        } else {
23          enableDisableRule(1,row['id_rule']);
24          if (row['hold_timer'] != 0) {
25            deleteTimerRule (obj,row);
```

```
26          }
27          }}
28  ...
29  }
```

Una regola può avere tre tipi di condizioni: “<”, “ >”, “=”. Il codice qui sopra dimostra l’implementazione della regola tramite la condizione “<”. Il primo controllo è stato verificare se la condizione è stata soddisfatta o meno, se è stata soddisfatta viene effettuato un’altro controllo per verificare se la regola è abilitata. L’abilitazione della regola è un aspetto fondamentale che è stato aggiunto per evitare che una regola venisse risolta infinite volte. Facciamo un esempio pratico, se viene aggiunta una regola che accende il ventilatore quando viene rilevata una temperatura maggiore di 25°e il sensore di temperatura pubblica una temperatura ogni 5 secondi di 26°, implica che venga inviato un comando di accensione al ventilatore ogni 5 secondi, e questo non va bene. La soluzione a questo problema è stata trovata aggiungendo un valore aggiuntivo alla regola chiamato `enabled` che viene abilitata di default quando viene aggiunta la regola, poi viene disabilitata quando la regola viene risolta *riga 17* e viene riabilitata quando la condizione non è più soddisfatta, *riga 23*. Così facendo questa alternanza tra abilitato e disabilitato riesce a risolvere il problema.

Continuando la descrizione del codice, se la regola non prevede un timer allora viene eseguita immediatamente, altrimenti se lo prevede come si può vedere alla riga *19* verrà chiamato il metodo `addTimerRule` che vedremo nel dettaglio successivamente. Infine se la condizione non è stata soddisfatta ed è presente un timer viene annullato tramite il metodo `deleteTimerRule` e riabilitata la regola, *riga 23*.

```
1  var timers = {};
2
3  function addTimerRule (obj,row) {
4    if (!(row['id_rule'] in timers)) {
5         timers[row['id_rule']] = setTimeout(function(){
6         var post = {};
7         post[row['key_name_result']] = row['key_type_result']
              == 'none' ? "" : row['key_value_result'];
8         publish(row['topic_name_result'],post);
9         enableDisableRule(0,row['id_rule']);
```

```
10        delete timers[row['id_rule']];
11
12      }, row['hold_timer'] * 1000);
13    }
14  }
```

Innanzi tutto come si può vedere nella riga *1* i timer vengono salvati in un array, in particolare in un array chiave valore, in cui la chiave è l'id della regola salvata nel database in modo da non aggiungere due volte lo stesso timer già attivo. Infatti la prima cosa all'interno del metodo `addTimerRule()` è quella di controllare se esiste già un timer all'interno dell'array con quella chiave, se non esiste viene creato e aggiunto all'array.

```
1  function deleteTimerRule (obj,row) {
2    if (row['id_rule'] in timers) {
3        clearTimeout(timers[row['id_rule']]);
4        delete timers[row['id_rule']];
5      }
6  }
```

Qui solo se esiste una regola all'interno dell'array con quell'id viene annullato il timer e eliminata la regola.

3 Web Dashboard

In questo paragrafo verrà analizzata la dashboard dell'interfaccia utente, invece nel capitolo successivo con i casi di studio verranno analizzate tutte le funzionalità nel dettaglio dell'interfaccia utente. La dashboard permette all'utente di avere una visione completa di tutti gli attuatori e sensori aggiunti nel suo profilo, ogni sensore o attuatore è incorporato in un widget ed è possibile per i sensori osservare il dato più recente in real-time, mentre per gli attuatori poterli controllare in maniera diretta.

Partiamo dai widget per i sensori, come è possibile vedere nell'immagine sottostante il widget fornisce in real-time l'ultimo dato disponibile rilevato dal sensore e il nome del sensore che l'ha pubblicato.

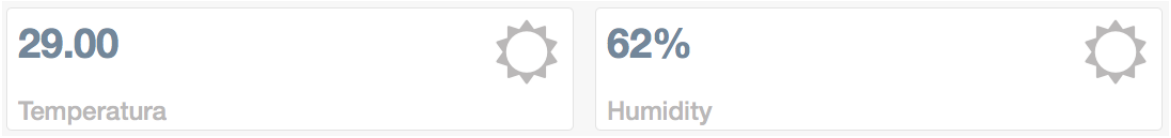

Figura 3.2: Widget Sensori.

Invece per quanto riguarda i widget degli attuatori, abbiamo due tipi di visualizzazioni in base alla scelta effettuata nel form di aggiunta dell'attuatore:

- Button View, il widget conterrà un pulsante per ogni chiave aggiunta al topic di subscribe nel relativo form. I pulsanti serviranno per inviare quella chiave come comando all'attuatore.
- Field View, il widget conterrà un campo di testo per ogni chiave aggiunta al topic di subscribe nel relativo form. I campi di testo verranno utilizzati per decidere quale valore inviare insieme alla chiave all'attuatore.

Nell'immagine sottostante a sinistra è possibile vedere come sia stata usata la Field View per esempio per poter settare un valore all'intensità di luce di una lampadina, mentre in quella di destra avremo dei pulsanti per poter inviare dei comandi di on e off pre-impostati.

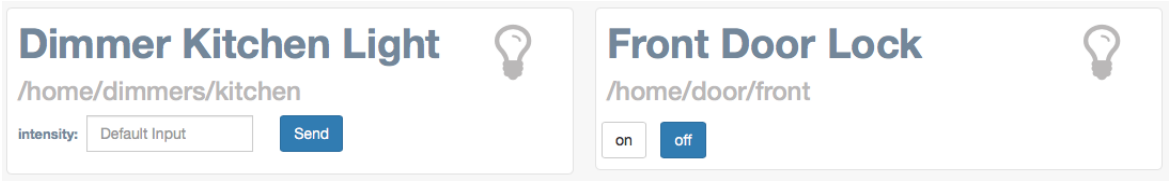

Figura 3.3: Widget Attuatori.

Infine nell'immagine che segue è possibile visualizzare la dashboard completa.

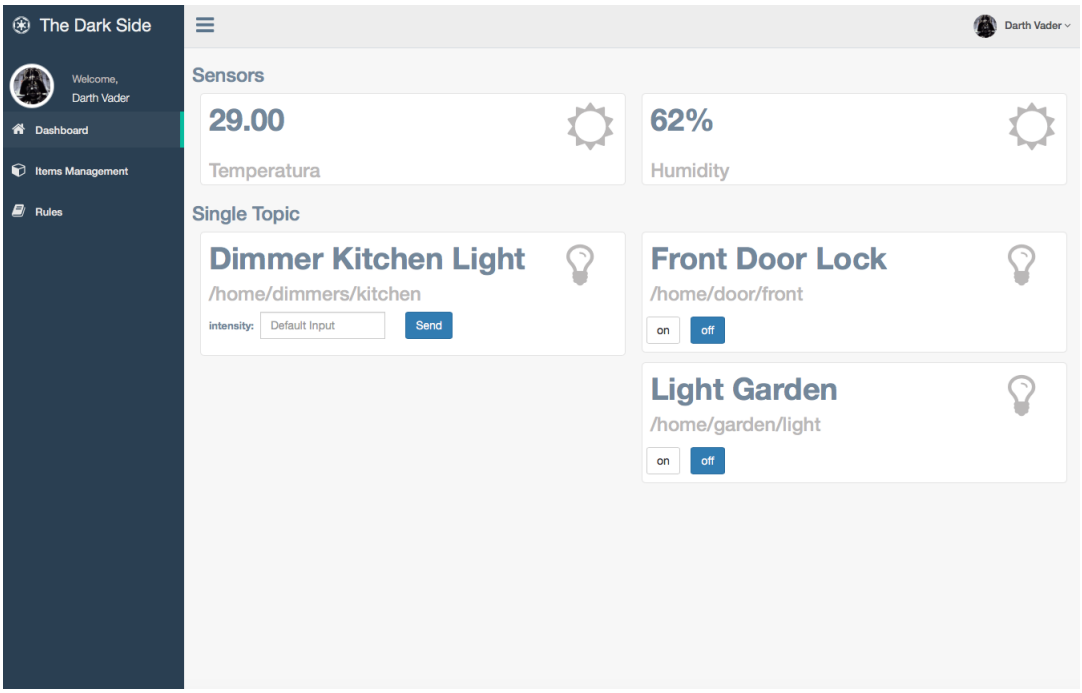

Figura 3.4: Dasboard.

Capitolo 4

Casi di studio

In questo capitolo verranno descritti quattro casi di studio applicati all'Home Automation utilizzando l'IoT Middleware realizzato. Nel primo caso di studio verrà mostrato come tramite l'utilizzo di una semplice regola è possibile avere una soluzione embodied o disembodied. Nei successivi due verrà prima analizzato un problema del primo caso di studio e di come possa essere risolto abbinando una condizione con un timer alla regola e infine come utilizzare le coordinate gps con l'utilizzo delle regole. L'ultimo caso di studio invece dimostrerà come l'utente in maniera semplice possa modificare la configurazione di uno scenario già attivo senza alterare il suo funzionamento.

1 Disaccoppiamento Embodied-Disembodied

Come è possibile vedere nella figura sottostante questo caso di studio prevede l'uso di sensori di luminosità posizionati in ogni lato della casa in cui sono presenti delle finestre. In ogni finestra è presente una tapparella automatizzata da un motore, infine nel giardino dell'abitazione sono presenti delle luci per l'illuminazione notturna. I sensori di luminosità, i motori e le luci sono collegati al gateway installato nell'abitazione e collegato alla rete che fa da intermediario con la piattaforma cloud-based.

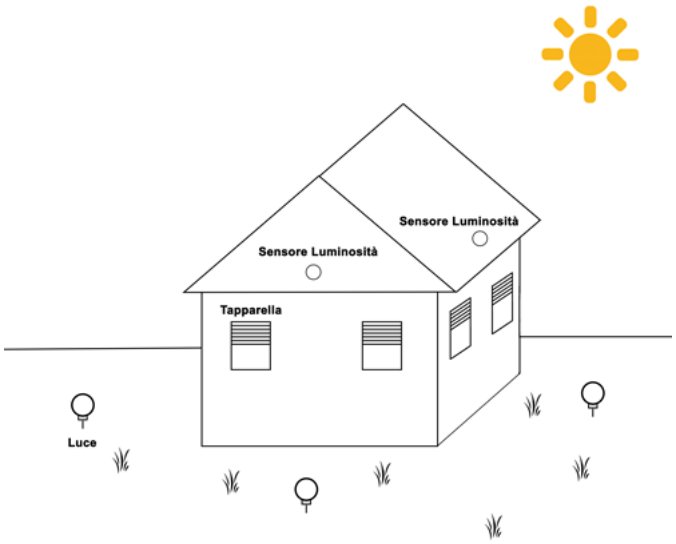

Figura 4.1: Scenario 1.

Questo caso di studio ha due obbiettivi, il primo è quello di abbassare le tapparelle elettriche quando il sensore di luminosità posizionato nello stesso lato rileva una quantità di luce superiore di quella impostata, una soluzione di questo tipo potrebbe essere impiegata per preservare il legno delle finestra quando c'è troppa luce solare. Il secondo obbiettivo è quello di accendere le luci posizionate in giardino quando il sensore di luminosità rileva una quantità di luce inferiore di quella impostata, per avere un'illuminazione notturna.

Vediamo ora nel dettaglio come implementare questo scenario appena descritto tramite l'utilizzo di regole implementate nell'IoT Middleware che ci permettono di disaccoppiare la parte embodied da quella disembodied. Innanzi tutto l'utente deve collegare fisicamente i sensori e gli attuatori al gateway e scrivere il codice necessario per poter utilizzare questi dispositivi e abilitarli allo scambio di messaggi. Visto che era al di fuori dello studio della tesi non è stata implementata alcuna interfaccia grafica utente nell'IoT Gateway, lato client. Quindi per poter utilizzare questa piattaforma è necessaria da parte dell'utente qualche competenza tecnica sia per il collegamento fisico dei dispositivi che per il codice per gestirli. Questo non esclude che in futuro possa

essere completata la piattaforma per la gestione totale del sistema tramite interfaccia grafica.

Per una maggiore comprensione del funzionamento dello scenario in questo paragrafo possiamo dire che nell'implementazione del lato client le luci seguendo il modello publish/subscribe del protocollo a livello applicativo MQTT iscriveranno al topic: `/home/garden/light`.

Nello stesso modo i motori che automatizzano le tapparelle elettriche si iscriveranno a due topic: `/home/window/motor` e `/home/window/motor/south`. Il primo topic permette di abbassare tutte le tapparelle all'interno dell'abitazione, per esempio attraverso un pulsante nell'interfaccia grafica utente, invece il secondo topic permette soddisfare lo scenario abbassando le tapparelle in base alla quantità di luce rilevata dal sensore di luminosità abbinato.

Questi topic da soli dimostrano l'aspetto più importante dello scenario ossia il disaccoppiamento della parte embodied e disembodied del sistema. Partiamo dall'aspetto disembodied, tutte le luci sono iscritte allo stesso topic ciò dimostra che il sistema non ha nessuna concezione o conoscenza di quali/quante luci siano o dove siano posizionate, sa solo che le deve spegnere o accendere. E' possibile fare lo stesso discorso per il topic generale a cui si sono iscritti tutti i motori, diversamente il secondo topic determina la posizione del motore posizionato sulla finestra, questo va a caratterizzare l'aspetto embodied del sistema, perchè si ha conoscenza di quali sono e dove sono i motori con cui vogliamo interagire.

L'iscrizione ai rispettivi topic permette in questo caso agli attuatori di poter ricevere comandi codificati in formato JSON, per esempio nel caso delle luci riceveranno pacchetti JSON con all'interno una chiave `on` o `off` rispettivamente per accendere o spegnere le luci.

Per quanto riguarda invece i sensori di luminosità siccome lo scenario prevede che ogni sensore controlli il proprio lato della casa, come per i motori avremo topic separati in base alla loro posizione, e andranno a pubblicare informa-

zioni tramite il comando publish, per esempio: `/home/sensor/light/south`. Bisogna sottolineare come già detto in fase di implementazione che la lettura dei sensori avviene tramite polling da parte del Gateway, quindi la frequenza di lettura viene scelta in base alla criticità dello scenario. Per uno scenario di questo tipo dove il sensore di luminosità deve percepire il valore della luce solare che non ha cambiamenti drastici, non è necessaria una lettura ogni millisecondo o ogni secondo, ma possiamo scegliere una frequenza di 3 secondi o anche maggiore.

Dopo aver impostato la parte client, bisognerà andare ad aggiungere gli stessi sensori e attuatori nell'interfaccia utente della piattaforma cloud-based, partendo dalla creazione del gateway e proseguendo con i sensori e attuatori. Questo ci permetterà di modellare in maniera astratta la stessa struttura fisica realizzata nel lato client. Qui di seguito viene mostrata un'immagine che mostra come in maniera semplice e immediata è possibile aggiungere un gateway al profilo utente nella piattaforma cloud.

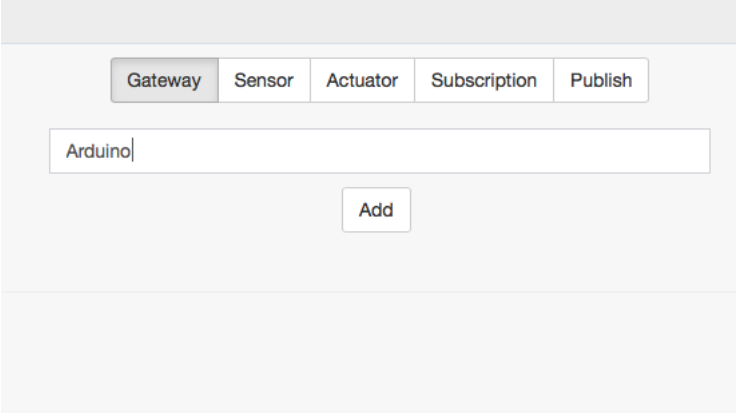

Figura 4.2: Creazione Gateway tramite interfaccia web.

Nello stesso modo sarà possibile aggiungere i sensori di luminosità per ogni lato in cui sono presenti le finestre, e collegarli con il topic di publish in cui verranno pubblicati i dati percepiti. In questo modo ogni volta che un sensore pubblicherà un dato, l'IoT Manager assocerà lo specifico dato allo specifico sensore.

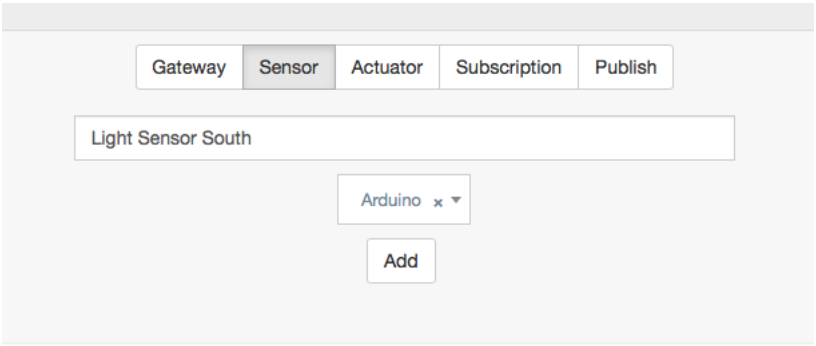

Figura 4.3: Creazione del sensore collegato al gateway.

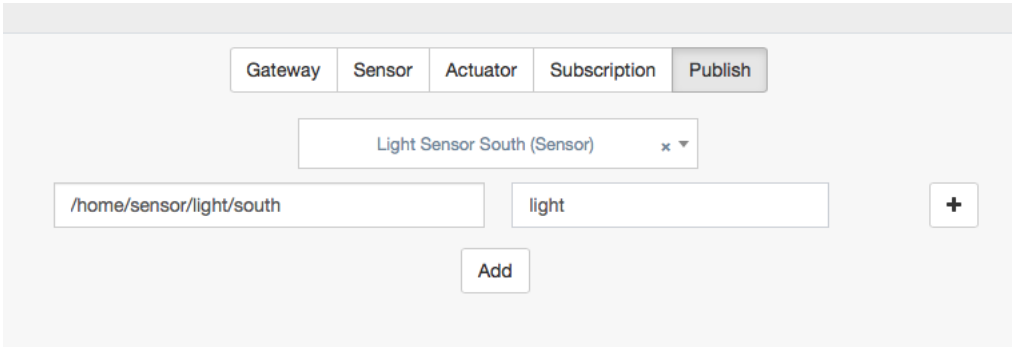

Figura 4.4: Collegamento del sensore con il topic in cui pubblicherà i dati.

Infine anche qui con lo stesso procedimento sarà possibile aggiungere gli attuatori e collegarli ai topic a cui si iscriveranno in attesa di ricevere informazioni.

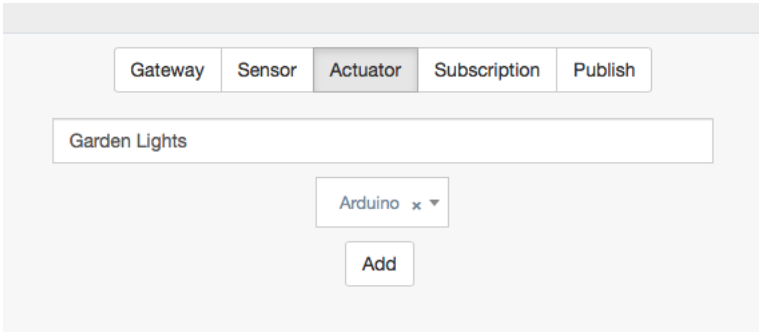

Figura 4.5: Creazione dell'attuatore collegato al gateway.

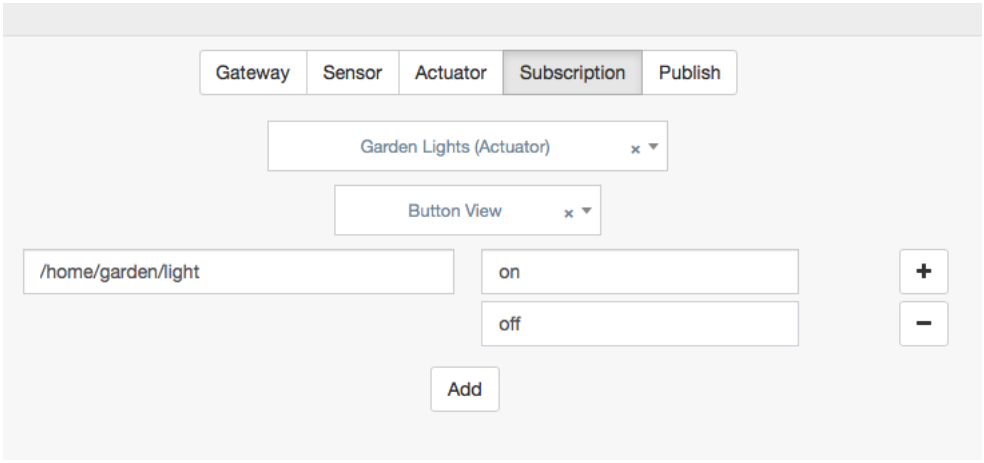

Figura 4.6: Collegamento dell'attuatore con il topic a cui si iscriverà in attesa di informazioni.

In questo modo sarà possibile aggiungere tutti i componenti IoT, ed avere una panoramica di cosa è stato aggiunto al gateway come è possibile vedere nell'immagine sottostante.

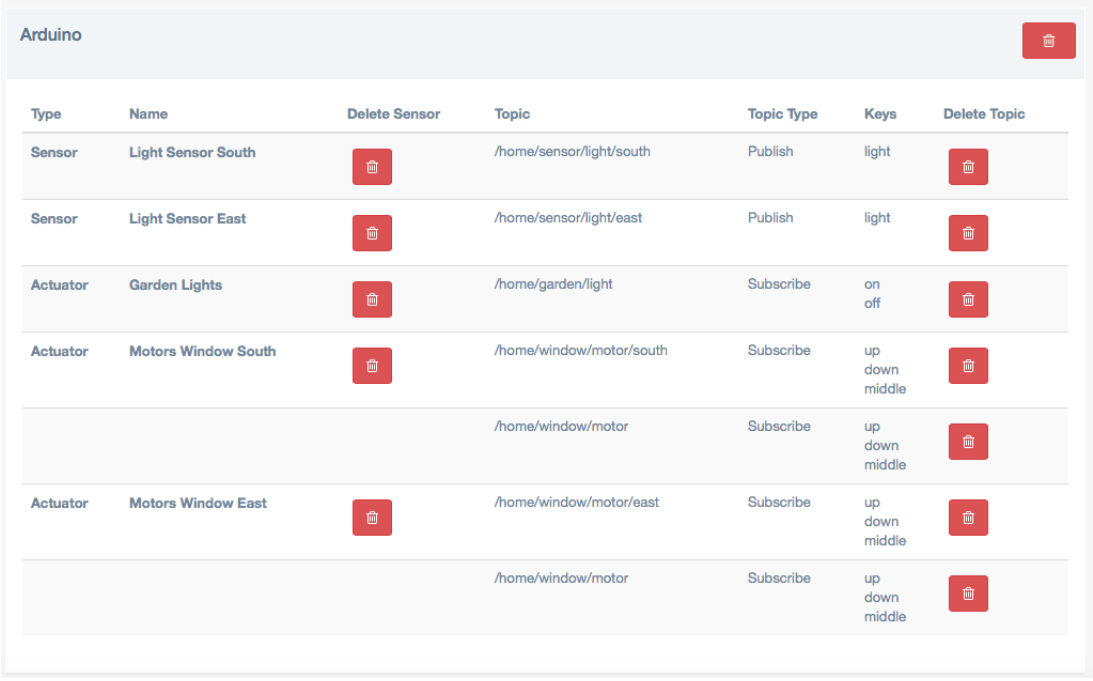

Figura 4.7: Panoramica dei componenti aggiunti.

Per concludere la configurazione dello scenario tramite l'interfaccia web, l'u-

tente avrà la possibilità di aggiungere le regole anche qui utilizzando un semplice form, vediamo nel dettaglio due regole che ci permettono di capire meglio la parte embodied e quella disembodied.

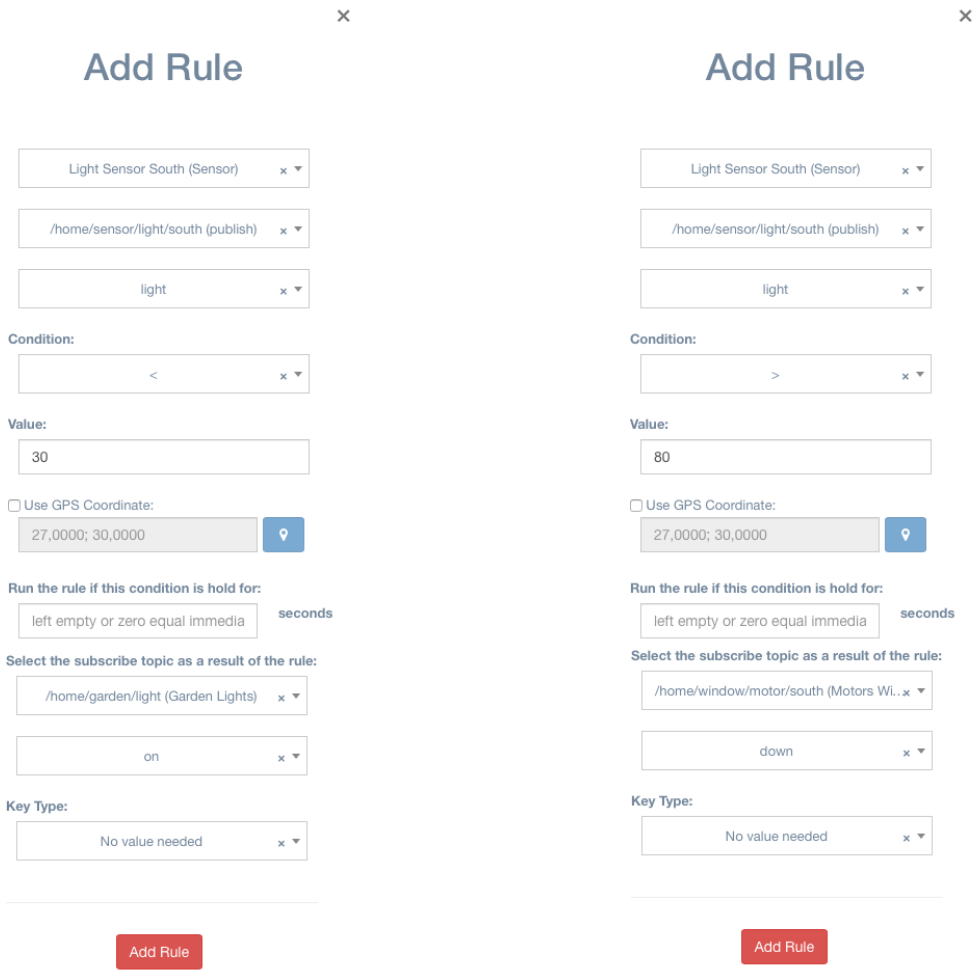

Figura 4.8: Regole.

L'immagine di sinistra mostra chiaramente il caso disembodied in cui non appena l'IoT Manager riceve il valore dal sensore di luminosità (in questo esempio è stato mostrato quello posto a sud della casa, ma si potrebbe posizionare un'altro in cima al tetto in modo da avere una percezione a 360°della quantità di luce per evitare zone d'ombra) e questo valore è inferiore di 30, allora la piattaforma cloud in automatico pubblicherà un messaggio JSON con chiave `on` al topic `/home/garden/light`. Così facendo tutte le luci in giardino verranno accese, questo mostra chiaramente la parte disembodied

perchè il sistema non sa dove sono posizionate le luci o quante luci ci sono sa solo che deve inviare un messaggio.

L'immagine di destra invece mostra il caso embodied, in cui non appena l'IoT Manager riceve dal sensore di luminosità posizionato a Sud, un valore maggiore di 80, allora il sistema pubblicherà un messaggio JSON con chiave `down` ai motori posizionati a Sud. Nel lato client il comando `down` sarà stato implementato in modo che le tapparelle collegate ai motori si abbassino completamente, questo dimostra come il sistema ha conoscenza di chi riceverà il messaggio.

Di seguito mostriamo il diagramma delle interazioni, che mostra come le varie parti che compongono il sistema comunicano tra di loro per gestire lo scenario, in particolare per la regola 1.

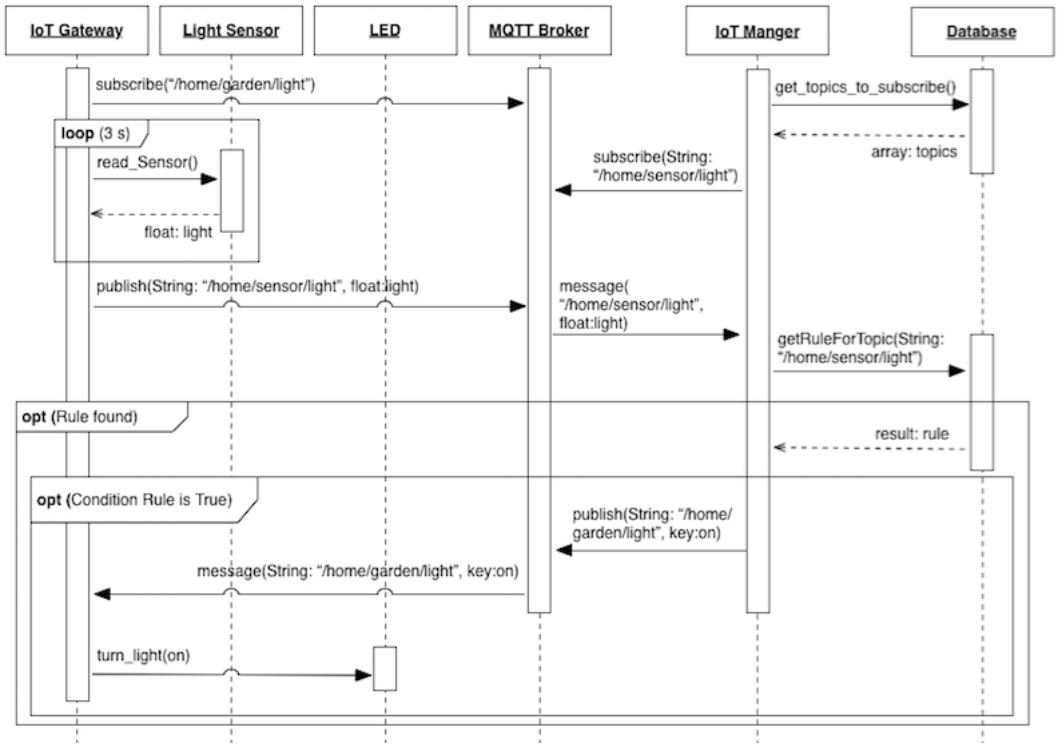

Figura 4.9: Diagramma delle interazioni per lo scenario 1 e la regola 1.

In questo caso di studio è stato possibile vedere quindi come l'utente possa utilizzare l'interfaccia web dell'IoT Middleware per poter realizzare compu-

tazioni embodied e disembodied in maniera semplice.

2 Situatedness temporale

Dopo aver analizzato il primo caso di studio è stato trovato un problema che potrebbe compromettere il risultato dello scenario. Questo perchè la regola creata per accendere le luci in giardino quando il sensore di luminosità rileva un valore inferiore di 30 è immediata, quindi se il sensore di luminosità viene coperto per un attimo da qualcosa le luci vengono accese anche se non dovrebbero. E' stata trovata una soluzione a questo problema implementando nell'IoT Middleware dei Timer nell'utilizzo delle Regole. L'utente quando aggiungerà una regola utilizzando il form dell'interfaccia web può scegliere tra due casi:

- Il primo è quello di lasciare il campo inerente al timer vuoto, questo equivale a scriverci 0 quindi la regola verrà risolta immediatamente non appena la condizione sarà soddisfatta (Come è stato mostrato nel primo caso di studio).
- Il secondo caso è quello di scrivere un valore in secondi, che indica per quanto tempo la condizione deve essere soddisfatta affinchè venga eseguita la regola.

Vediamo qui di seguito la soluzione al problema del caso di studio 1 con la nuova regola.

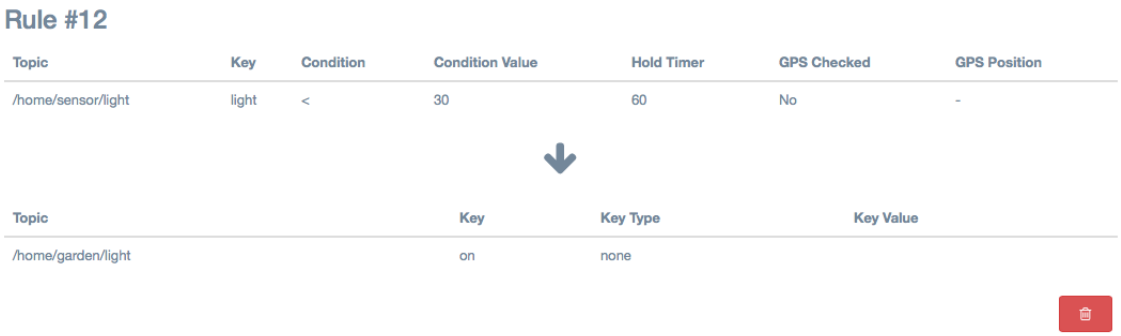

Figura 4.10: Regola con Timer.

La nuova regola quindi dice che non appena viene rilevato un valore minore di 30, viene fatto partire un timer con durata di 60 secondi, se in questo lasso

di tempo continuano ad arrivare valori minori di 30 allora il timer continuerà ad esistere e allo scadere verrà eseguita la regola, altrimenti se arriva un valore maggiore di 30 il timer viene annullato.

Il diagramma delle interazioni a parte alcune modifiche ai nomi delle richieste è uguale a quello in *Figura 4.9*.

Questa semplice soluzione ci permette di poter realizzare tantissimi scenari nell'ambito dell'Home Automation, come per esempio potrebbe essere quello in cui posizionando un sensore di movimento all'interno di una stanza, se nella stanza non viene rilevato nessuno per qualche minuto allora si può spegnere la luce per risparmiare energia.

3 Situatedness spaziale

La situatedness temporale è stata completata con quella spaziale, infatti è possibile scegliere se utilizzare una coordinata geografica come condizione di controllo nell'utilizzo di una regola. Spieghiamo nel dettaglio questa funzione attraverso la descrizione di uno scenario di Home Automation, lo scenario prevede che quando l'automobile dell'utente si avvicina a una distanza scelta per esempio 100m dall'abitazione venga in automatico aperta la porta del garage. Questo scenario prevede l'utilizzo di un sensore GPS posizionato sull'automobile oppure potrebbe anche essere realizzata un'app per Smartphone utilizzando il sensore GPS integrato, che possa spedire la posizione all'IoT Middleware ogni intervallo di tempo.

Per poter realizzare uno scenario di questo tipo l'utente deve aggiungere una regola fatta così:

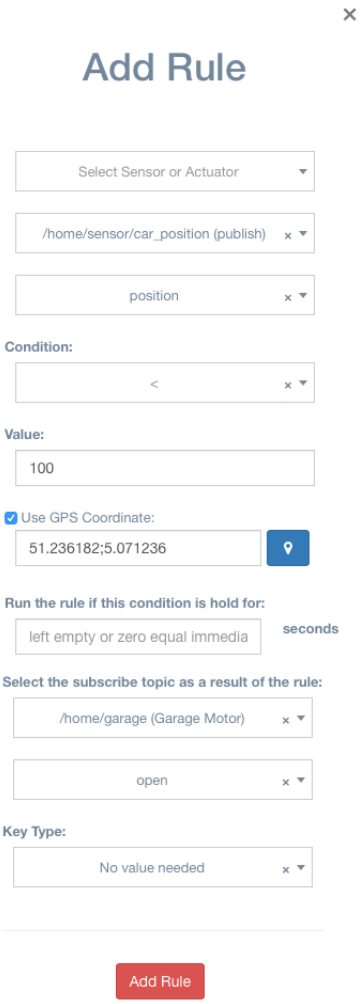

Figura 4.11: Regola con Posizione GPS.

La prima cosa da fare è spuntare la checkbox *Use GPS Coordinate* per abilitare il form, dopo di che tramite l'utilizzo del pulsante blu è possibile recuperare la posizione corrente tramite browser in cui si trova il dispositivo che sta aggiungendo la regola, altrimenti è possibile inserire i valori di latitudine e longitudine separati da punto e virgola manualmente, per esempio: `51.236182;5.071236`.

Per verificare se la condizione è stata soddisfatta viene calcolata la differenza in metri tra la coordinata spedita dal sensore GPS e la coordinata di riferimento inserita nel form della regola, il valore risultante sarà utilizzato per il controllo della condizione, per esempio in questo scenario se è minore di

100m viene aperta la porta del garage.

Il diagramma delle interazioni a parte alcune modifiche ai nomi delle richieste è uguale a quello in *Figura 4.9*.

4 Flessibilità ed estendibilità

In questo caso di studio mostreremo la semplicità con cui un utente potrà aggiungere componenti che fanno già parte di un gruppo di sensori/attuatori senza fare modifiche nell'interfaccia web. In particolare facendo riferimento al primo caso di studio descritto precedentemente spiegheremo come fare per aggiungere una luce in giardino in aggiunta a quelle esistenti, così facendo verrà mostrata ulteriormente la parte disembodied e la dinamicità dell'IoT Middleware.

Grazie a come è stato progettato il middleware, l'utente non deve effettuare modifiche nell'interfaccia web, perchè vogliamo aggiungere una lampadina in un gruppo di lampadine che hanno già le proprie regole e sono già presenti all'interno del profilo. Le uniche due cose da fare da parte dell'utente lato client sono:

- Aggiungere fisicamente la lampadina in giardino con gli opportuni collegamenti al gateway.
- Effettuare una piccola modifica al codice per aggiungere la lampadina nel software del gateway in modo che si possa accendere e spegnere con le altre quando arriveranno gli opportuni messaggi relativi al topic a cui sono già iscritte tutte le lampadine.

Fatte queste due piccole modifiche la lampadina aggiunta al sistema è già funzionante e verrà accesa con le altre al calare dell'illuminazione come descritto nel primo caso di studio. Questo rafforza ulteriormente la potenza di questo middleware e di questa strategia e come la parte disembodied possa in maniera semplice gestire lo scenario.

4.1 Sviluppi Futuri

Siccome era al di fuori del focus della tesi non è stata implementata alcuna interfaccia grafica utente lato client, quindi l'unico modo per poter modificare le informazioni nel gateway e il comportamento dei componenti è tramite la modifica del codice. Questo non permette a tutti di utilizzare la piattaforma, perchè sono necessarie competenze tecniche per il collegamento hardware e per la modifica del software nel gateway. Come sviluppo futuro però si potrebbe modificare l'architettura del gateway e dell'interfaccia web in modo che il comportamento del gateway possa essere deciso tutto dall'interfaccia utente cloud-based e inviato al gateway. In questo modo si modificherebbe il sistema senza scrivere alcuna linea di codice anche nel lato client, così facendo la piattaforma diventerebbe più user-friendly.

Conclusioni

Questa tesi ha avuto il compito di analizzare come i dispositivi IoT si potessero interfacciare con il Cloud e come e dove la computazione potesse essere distribuita lato Embodied e Disembodied, per farlo è stata realizzata una piattaforma che facesse da Middleware tra i dispositivi IoT e il Cloud. Per realizzare questa piattaforma è stato utilizzato il modello di comunicazione Gateway-to-Cloud che al momento sembra essere uno dei modelli più utilizzati in ambito commerciale, questo perchè vista la mancanza di uno standard per il protocollo di comunicazione nell'IoT, il gateway permette di poter interfacciare più dispositivi con protocolli differenti che altrimenti non potrebbero comunicare tra di loro. La piattaforma ci ha permesso sopratutto attraverso l'utilizzo di regole di capire come si può differenziare la computazione Embodied e Disembodied in scenari come quello dell'Home Automation.

La piattaforma realizzata solo per studio ha ampi margini di miglioramento, uno tra tanti è quello della sicurezza nell'IoT e nel Cloud che è stato solo analizzato nel capitolo 1, ma siccome è al di fuori del focus di questa tesi non è stato implementato nell'IoT Middleware. Il problema della sicurezza e della privacy è qualcosa che potrebbe rallentare la crescita di questa nuova tecnologia integrata con il Cloud, perchè le informazioni che vengono trasmesse tra questi dispositivi sono molte volte informazioni personali dell'utente o dell'azienda. Bisogna quindi valutare tutti gli aspetti e cercare una soluzione comune a questi problemi in modo da realizzare degli standard e delle linee guida che sviluppatori e aziende possano seguire.

Dopo questo studio possiamo dire che combinare i computer, i sensori, e le reti per monitorare e controllare dispositivi è stato pensato da decenni, ma

il recente rilascio di alcune tecnologie chiave e l'andamento del mercato ha fatto si che l'Internet of Things sia ora una nuova realtà. L'IoT ha iniziato una rivoluzione, in un mondo che sta diventando sempre più "smart", i collegamenti tra oggetti e l'ambiente e tra oggetti e persone stanno diventando sempre più legati tra di loro. Il prospetto dell'Internet of Things come un insieme onnipresente di dispositivi legati a Internet potrebbe radicalmente cambiare ciò che la gente pensa su cosa significa essere "online".

Infatti non tutti hanno conoscenza del significato di essere sempre "online" che ha portato il Cloud grazie alla continua crescita dell'ubiquità di calcolo, questo fa si che tutto sia connesso a Internet e i dati elaborati vengano usati per vari scopi, creando non solo informazione ma anche conoscenza e saggezza. Così facendo l'IoT e il Cloud sembrano fatti apposta per lavorare in simbiosi, i piccoli dispositivi IoT con poca capacità di calcolo hanno bisogno della potenza di calcolo che il Cloud può offrire in modo da gestire, memorizzare, elaborare e analizzare il continuo flusso di dati generati da questi dispositivi.

Bibliografia

[1] Vasileios Karagiannis, Periklis Chatzimisios, Francisco Vazquez-Gallego, Jesus Alonso-Zarate, *"A Survey on Application Layer Protocols for the Internet of Things"*;

[2] Danilo Candiotti, *"IoT e progettazione di Sistemi di Home Automation: un caso di studio reale basato su framework e standard open"*;

[3] Stefano Mariani and Andrea Omicini, *"TuCSoN on Cloud: An Event-Driven Architecture for Embodied / Disembodied Coordination"*;

[4] Internet Society, *"The Internet of Things: An Overview"*;

[5] Node JS MySQL, *"https://github.com/mysqljs/mysql"*;

[6] Socket.io, *"http://socket.io/"*;

[7] MQTT client for Node.js, *"https://github.com/mqttjs/MQTT.js"*;

[8] A client library for the Arduino Ethernet Shield that provides support for MQTT, *"http://pubsubclient.knolleary.net/"*;

[9] MQTT Python client, *"https://pypi.python.org/pypi/paho-mqtt/1.1"*;

[10] MQTT Message Broker, *"http://mosquitto.org"*;

Ringraziamenti

Voglio ringraziare i miei genitori che mi hanno sempre sostenuto e che mi hanno permesso, con i loro sacrifici, di raggiungere questo obbiettivo, altrimenti irraggiungibile.

Voglio ringraziare la Giulia che mi ha sempre spronato e dato la forza di continuare, Ti Amo.

Voglio ringraziare mia sorella, anche se ormai la vedo due volte l'anno mi ha sempre aiutato anche a distanza ed è sempre nei miei pensieri e nel mio cuore.

Voglio ringraziare Zack, Busca e Richard per tutto il tempo trascorso insieme su progetti e esami che sembravano insormontabili, ma ragazzi possiamo dire che c'è l'abbiamo fatta!

Grazie Zack per tutti i momenti passati insieme a lavorare ai nostri progetti alternativi, le notti insonni e per le piccole soddisfazioni che ci siamo tolti, me le porterò sempre nel cuore nonostante quello che decideremo di fare con Whale{TRUE}.

Grazie al Prof. Omicini per la sua disponibilità e continua simpatia.

Grazie al Dott. Stefano Mariani per l'amicizia e per l'aiuto nello sviluppo della tesi.

Grazie a Tea per il suo affetto.