Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Ingegneria e Scienze Informatiche

PROGETTAZIONE E SVILUPPO DI UN PROTOTIPO DI PIATTAFORMA PER WEB OF THINGS

Relatore:
Chiar.mo Prof.
ALESSANDRO RICCI

Presentata da:
MATTIA VANDI

Correlatore:
Dott. Ing.
ANGELO CROATTI

Sessione II
Anno Accademico 2016 - 2017

PAROLE CHIAVE

Pervasive computing
Sistemi Embedded
Internet of Things
Internet delle Cose
Web
Web of Things
Web delle Cose

A mio nonno Angelo

Introduzione

Con l'avvento dell'"Internet delle Cose" (o Internet of Things, utilizzando il termine anglosassone, in breve IoT) sempre di più alcuni dispositivi "intelligenti" sono stati integrati nelle "cose" che noi tutti utilizziamo quotidianamente.

Tali dispositivi si rendono riconoscibili attraverso tecnologie per l'identificazione (RFID, NFC, iBeacon, etc.) e acquisiscono intelligenza grazie al fatto di poter comunicare dati ed accedere ad informazioni aggregate rese disponibili da altri dispositivi in rete.

Un obiettivo generale dell'Internet delle Cose (IoT), è di poter tracciare una mappa digitale del mondo reale, dando così un'identità elettronica alle cose e ai luoghi dell'ambiente fisico.

I campi di applicabilità sono molteplici: dalle applicazioni industriali (processi produttivi), dalla logistica all'infomobilità, fino all'effcienza energetica, all'assistenza remota e alla tutela ambientale.

In questa tesi si andranno ad esplorare le possibili strategie per integrare dispositivi sprovvisti di un accesso diretto alle rete internet, attraverso un'attività di identificazione e di una successiva trasformazione per rendere questi dispositivi gestibili mediante l'utilizzo di tecnologie già presenti (Web 2.0) e di tecnologie elettroniche abilitanti (IoT).

L'integrazione di dispositivi elettronici nel Web ha assunto il nome di Web of Things come descritto nel "Towards the WOT Manifesto" [27] pubblicato il 10 Aprile 2009 da un gruppo di ricercatori, tra i quali Dominique Guinard e Vlad Trifa.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Internet of Things

1.1 Cos'è Internet of Things

L'avvento dell'era digitale ha sancito la creazione di nuove abitudini nello stile di vita delle persone: metodi di comunicazione moderni e accesso sempre maggiore a dispositivi tecnologici portatili sono diventati, a partire dai primi anni 2000, parte integrante della vita di chiunque. Questi fattori uniti allo sviluppo di Internet, ora accessibile praticamente da qualsiasi parte del pianeta e per mezzo di una miriade di dispositivi diversi, offrono la garanzia di poter disporre di una capacità elaborativa distribuita in grado di soddisfare praticamente qualsiasi richiesta. È partendo da questa considerazione che si può introdurre il concetto di *ubiquitous computing*, esso sta ad indicare il trend sempre crescente di incorporare microcontrolli e, nei casi più recenti o quando serve capacità elaborativa maggiore, anche microprocessori all'interno di oggetti di uso quotidiano (everyday objects), ubiquitous computing, detto anche a volte pervasive computing, significa infatti "existing everywhere" cioè la computazione ovunque. Un paradigma computazionale come quello appena introdotto nasce grazie allo sviluppo di Internet, ed è proprio la rete globale che meglio si offre a qualsiasi tipo di oggetto dotato di capacità di calcolo come miglior interfaccia per la comunicazione con il mondo o con qualsiasi uomo o macchina che abbia, come lui, accesso alla

stessa rete.

Pervasive computing e rete Internet danno vita a un paradigma che viene definito come Internet of Things, questo termine venne probabilmente usato per la prima volta da un gruppo di ricerca del MIT (Massachussetts Institute of Technology) nel 1999 mentre i suoi componenti si trovavano al lavoro per lo sviluppo della tecnologia RFID (Radio-Frequency Identification) al servizio di una rete di sensori, le cosiddette Wireless Sensor Network (WSN).

Essendo un termine coniato recentemente non ne esiste una definizione precisa e condivisa in letteratura, vorrei quindi proporne alcune: la prima non è una vera e propria definizione ma credo sia tra le più esplicative in quanto data dall'IoT Council [**?**] che così si esprime quando definisce il proprio compito:

> *"The Internet of Things (IoT) is a vision. It is being built today. The stakeholders are known, the debate has yet to start. In hundreds of years our real needs have not changed. We want to be loved, feel safe, have fun, be relevant in work and friendship, be able to support our families and somehow play a role however small in the larger scheme of Things. So what will really happen when things, homes and cities become smart. The result will probably be an tsunami of what at first looks like very small steps, small changes. The purpose of Council is to follow and forecast what will happen when smart objects surround us in smart homes,offices, streets, and cities."*

All'interno di queste poche righe è descritta quella che si auspica essere una visione futura del mondo e viene fatto ponendo l'uomo davanti a un bivio come quasi per metterlo in guardia davanti a uno sviluppo tecnologico, figurato come uno tsunami che dà l'idea di un qualcosa che parte e non può essere fermato, con cui si dovrà per forza di cose provare a convivere.

Un'altra definizione che mette l'accento sul rapporto che esiste fra l'uomo e lo sviluppo della tecnologia è quella data dal CISCO IBSG [17] (Internet Business Solution Group):

> *"IoT is simply the point in time when more 'things or objects' were connected to the Internet than people."*

Facendo rifermento sempre allo studio condotto da questo gruppo nel 2003, 4 anni dopo che il termine IoT venne usato per la prima volta, esisteva un rapporto di 0,08 dispositivi connessi in rete per persona, il che indicava l'impossibilità dell'esistenza al tempo di questa teconologia.

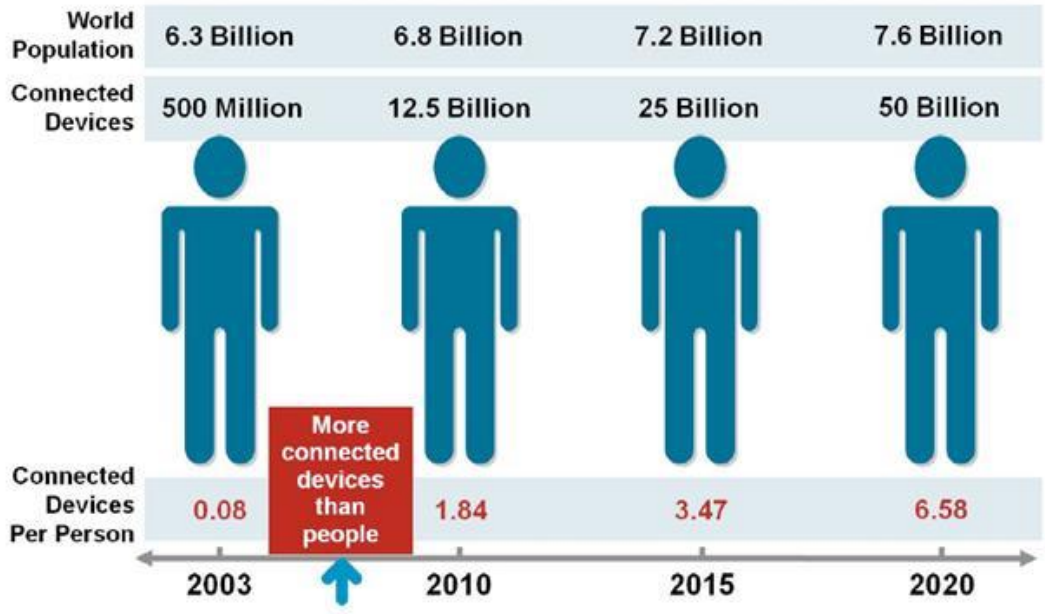

Figura 1.1: Crescita del numero di dispositivi collegati ad internet in proporzione alla popolazione globale dal 2003 ad oggi ed una previsione della sua crescita entro il 2020.

Seguendo la nota legge di Moore, questo rapporto di 0,08 nel 2003 è aumentato fino a superare la quota 1 nel 2010, sempre secondo CISCO: è quindi attorno a questo periodo (considerando anche il fatto che solo parte della popolazione mondiale ha accesso ad Internet tutt'ora) che qualsiasi tipo di applicazione basata su IoT può cominciare effettivamente ad esistere se si tiene conto che anche solo i primi smartphone, dispositivi fondamentali in tale scenario, sono stati presentati fra il 2007 e il 2008.

Passando a definizioni che più si avvicinano al mondo della tecnologia mi piacerebbe citare quella fornita dall'IIT [1] (Istituto di Informatica e Telematica) del CNR (Consiglio Nazionale delle Ricerche) che dell'Internet of Things dice:

> *"[...] rappresenta il superamento dei classici limiti della rete che, fuoriuscendo dal mondo virtuale si collega al mondo reale. Tag e sensori infatti, associati a un oggetto, possono identificarlo univocamente e raccogliere informazioni in tempo reale su parametri del suo ambiente [...]"*

Potrei elencarne a decine ancora ma preferisco soffermarmi a definire quelli che sono i tratti comuni che tutte quante le definizioni date contengono, per iniziare cos'i a delineare i vari argomenti che saranno discussi in dettaglio nel proseguo dell'elaborato.

- Presenza di oggetti dotati di interfacce di comunicazione (schede di rete);
- Portabilità elevata di questi oggetti intelligenti e dimensioni ridotte come caratteristica fondamentale;
- Possibilità di localizzazione e identificazione univoca;
- Possibilità di interazione con l'uomo bidirezionale;
- Preponderanza di inserimento capillare di oggetti, sensori e attuatori all'interno dell'ambiente spinta al punto tale che i primi utilizzatori della rete in termini numerici siano macchine e oggetti inanimati;
- Raccolta dati e informazioni sistematica con la possibilità di poter centralizzare l'elaborazione;
- Possibilità di proporsi come nuova rivoluzione sociale con risvolti sulle abitudini di vita non ancora ben definiti.

Il perché l'idea dell'Internet of Things sia così recente, è in primis dovuto al fatto che il supporto tecnologico necessario non è banale: se pensiamo che Internet nasce fra gli anni '60 e '70, che le dimensioni delle unità di calcolo dell'epoca erano imponenti e non di certo nell'ordine dei pochi centimetri odierni, tutto quello che oggi ci accingiamo a descrivere come una tecnologia consolidata (o per lo meno teoricamente delineata) poteva sembrare un'utopia. E invece in questo arco di tempo sono stati studiati e perfezionati protocolli di comunicazione, poi diventati degli standard, e delle tecnologie che hanno permesso di concentrare l'hardware necessario all'elaborazione delle informazioni e alla loro trasmissione in dispositivi che possiamo tranquillamente maneggiare, come per esempio gli stessi smartphone, oppure inserire nel contesto ambientale senza che si percepiscano neppure. La cosa più interessante è che questi protocolli, studiati inizialmente per altri scopi, si prestano in modo ottimale ad essere utilizzati, con qualche piccolo perfezionamento, anche dai moderni dispositivi terminali che compongono la parte periferica dell'Internet of Things. Questo è il caso del protocollo TCP/IP e di tutta la famiglia di protocolli dello standard IEEE 802, supportati per l'instradamento e il trasporto delle informazioni all'interno praticamente di qualsiasi rete compresa Internet, XML o JSON per la rappresentazione delle informazioni in formati ben interpretabili da qualsiasi dispositivo ed infine tecnologie come Bluetooth o ZigBee per la comunicazione fra dispositivi; ed è appunto a questo che si riferiva l'IoT council nella visione proposta quando si spiegava come tutto quello necessario per l'implementazione già esiste e va semplicemente assemblato.

La vera e propria sfida tecnologica risiede in altro. Esistono una serie di aziende diverse che hanno la possibilità di imporsi sulla scena mondiale con l'avvento di questa nuova era dell'informatica, non esistendo veri e propri standard per IoT, ognuno di questi producer ne potrebbe applicare di proprietari, e questo sarebbe certamente una sconfitta considerando l'obiettivo iniziale prefisso che riguarda l'integrazione totale fra mondo fisico e mondo virtuale. Un'altra considerazione a riguardo si potrebbe fare tenendo

conto degli enti privati o pubblici che potrebbero mettere a disposizione finanziamenti per lo sviluppo, questo credo sia un punto centrale per cercare di prevedere la direzione di crescita del settore, se si diriga più verso il business e la produzione industriale oppure verso il controllo, il monitoraggio e la salvaguardia dell'ambiente o della cittadinanza. Sarebbe in ogni caso ottimale definire, per consentire una crescita più veloce e uniforme di questa tecnologia, in che modo essa si debba "costruire" e come debba lavorare in sintonia con ciò che già esiste, con le altre macchine, con il web e quali debbano anche essere i limiti di uno sviluppo che rischia di andare in un verso che può portare a un controllo troppo invasivo. In particolare una standardizzazione della gestione di aspetti di sicurezza e privacy, introducendo opportuni modelli per la gestione dell'identità e del controllo degli accessi, è sicuramente un punto fondamentale che va regolato.

Esistono inoltre tutt'ora problematiche relative alla tecnologia non trascurabili che possono rallentare lo sviluppo dello scenario applicativo di IoT, prima fra tutte e non trascuarabile in mondo in cui ogni oggetto voglia essere all'interno di Internet è l'indirizzabilità, ad oggi infatti il protocollo utilizzato è l'IPv4 che, sfruttando 32 bit per l'indirizzamento, non ha più disponibilità di indirizzi validi. Il passaggio al protocollo IPv6 è quindi una delle prime sfide tecnologiche che si dovranno vincere per sognare effettivamente un mondo in cui ogni dispositivo sia realmente smart. La scalabilità è un'altra sfida per cui il progresso tecnologico dovrà proporre soluzioni che permetteranno di soddisfare un numero che diverrà sempre più elevato di richieste per ogni servizio presente in un mondo dominato da dispositivi intelligenti, la strada che si sta percorrendo oggi guarda verso l'inserimento di middleware che si frapporranno nella comunicazione tra utente e dispositivo, in modo che l'overhead provocato da un numero di connessioni molto elevato non gravi sul dispositivo in sé dotato di risorse limitate bensì su un componente che si inserirebbe nell'architettura con funzionalità (considerando per ora la scalabilità di comunicazione come caso analizzato) simili a quelle di un server web. Anche permettere l'inserimento dinamico di dispositivi all'interno di un'ar-

chitettura già operativa può non essere banale se si vuol evitare una fase di configurazione iniziale con cui si stabiliscano i modi e i tempi di comunicazione; riferendosi nuovamente al middleware, in questo caso come ponte delle comunicazioni tra ogni singolo componente dell'infrastruttura, l'espansione o la riduzione del numero di dispositivi in maniera dinamica sfruttandone le funzionalità può essere molto meno difficoltosa. Un discorso molto importante va infine fatto sulla parte di tracciabilità e scoperta dei servzi presenti nella rete globale: per esempio un utilizzatore di smartphone vuole avere accesso a un sensore di temperatura situato in una precisa località, tralasciando la parte di comunicazione che avviene fra i due, come fa questo utente a trovare ciò che possa soddisfare la sua richiesta. Questi ed altri temi saranno analizzati nella parte dedicata al Web of Things ovvero l'integrazione degli oggetti facenti parte di Internet of Things all'interno non solo della più grande e importante rete al mondo, ma anche all'interno del web.

1.2 Dispositivi hardware

Sono spesso stati nominati i cosiddetti dispositivi intelligenti (smart thing o smart device), questi rientrano all'interno della categoria dei sistemi embedded [29] cioè:

> *"una combinazione di componenti sia hardware che software, programmabile o meno, che è progettato per svolgere una funzione specifica o funzioni specifiche all'interno di un sistema più grande."*

Caratteristiche:

- Questi dispositivi sono accessibili a prezzi non molto elevati e offrono capacità di elaborazione contenute ma sono allo stesso tempo ottimali per essere integrati come dispositivi terminali all'interno di un'architettura IoT.

- La caratteristica di implementare al loro interno un software che ciclicamente viene eseguito li rende programmabili in maniera non troppo complessa, non richiedono un sistema operativo in quanto la gestione di tutte le risorse è fatta dall'unico software in esecuzione.

- Essendo di dimensioni ridotte e non avendo sempre disponibile una fonte di energia, necessitano di batterie o di riuscire ad immagazzinare energia dall'ambiente, in ogni caso un tema molto importante nella programmazione dei dispositivi embedded è l'efficienza nello sfruttamento dell'energia, e in generale l'efficienza nello sfruttamento di qualsiasi risorsa che è a loro disposizione perché quasi sempre limitata.

- Essendo molte volte immersi nell'ambiente necessitano processi industriali di produzione che utilizzino materiali non nocivi e non inquinanti, da considerarsi come inquinamento anche quello visivo o acustico.

- Non essendo continuamente monitorati, caratteristiche hardware e software quali l'affidabilità e la garanzia del corretto funzionamento sono necessarie.

- Possono essere richieste delle caratteristiche di real-time (o anche hard real-time) per dispositivi che assolvono allo svolgimento di compiti che hanno un rigore temporale molto forte. Questo può essere il caso di oggetti intelligenti al servizio di processi industriali di precisione, al servizio di strumenti medicali o per la segnalazione di pericoli.

Il sorprendete sviluppo dell'IoT ha modificato il trend di produzione e di richiesta dell'industria dei microprocessori e microcontrollori. Infatti anche per i sistemi embedded il componente più importante rimane sempre questo ma abbiamo una differenza molto grande rispetto al mercato dei tradizionali calcolatori, questi ultimi usano processori con architetture CISC (Complex Instruction Set Computer) che non sono adatte a sistemi con potenza di calcolo ridotta, infatti i dispositivi embedded utilizzano processori con architetture RISC (Reduced Instruction Set Computer) in virtù della

buona relazione che lega costo, consumo energetico e prestazioni. Esempi di microcontrollori per sistemi embedded sono: Atmel AVR, Texas Instruments MSP430, Microchip PIC16C84, ARM a 32 bit, Hitachi H8, PowerPC, Espressif ESP8266; tutti nati da produttori diversi a sottolineare il fatto che esiste una frammentazione nella progettazione e nella produzione. Sono tutti comunque, indipendentemente dall'architettura, a 8, 16 e raramente a 32 bit, posseggono dei pin per il collegamento diretto dei dispositivi di input (sensori) ed output (attuatori) e hanno dei pin dedicati alla gestione degli interrupt, molto importanti in sede di progettazione del software, soprattutto nei sistemi non dotati di sistema operativo, per la segnalazione di eventi esterni non trascurabili e da processare con urgenza.

Di pari passo è in atto anche una rivoluzione nel mercato delle memorie, i sistemi quali smartphone, lettori multimediali e macchine fotografiche digitali richiedono infatti l'utilizzo di unità di memorie veloci, silenziose e con un costo comunque non molto elevato; è quindi in atto un passaggio sempre più massiccio verso tipologie di memorie flash. Queste memorie a stato solido non volatili riescono ad avere costi contenuti per capacità non molto elevate e sono efficienti dal punto di vista dei consumi e dell'affidabilità, nei sistemi embedded vengono usate DRAM (Dynamic Random Access Memory) che garantiscono una densità maggiore e un minore costo a dispetto delle più veloci SRAM (Static Random Access Memory). La caratteristica di integrare su un unico chip tutta la componentistica necessaria per il funzionamento e la programmazione (microprocessore, I/O, generatore di clock, RAM, memoria per contenere il programma, etc.), fa sì che ci si riferisca a questi sistemi spesso con il nome di SoC (System-on-a-chip), dotati di una buona flessibilità, evitano al programmatore qualsiasi tipo di progettazione hardware e si rendono subito disponibili all'implementazione del software; fra questo tipo si sistemi sicuramente il più famoso è Arduino.

Come già detto i microcontrollori utilizzati in questo ambito dispongono di alcune linee di Input/Output direttamente programmabili via software, ed è quello che si fa quando si vogliono gestire sensori e attuatori. I primi

sono dei trasduttori che permettono di misurare diversi fenomeni fisici (temperatura, luminosità, etc.) e convertono la grandezza misurata in segnali analogici o digitali direttamente interpretabili dall'unità di calcolo, i secondi sono invece dispositivi che producono effetti misurabili nell'ambiente generati solitamente in risposta ad eventi che, sempre nell'ambiente, accadono.

Nel contesto di IoT per "cosa" o "oggetto" ci si riferisce ad un'entità o un oggetto fisico che ha un identificativo univoco, un sistema embedded e l'abilità di trasferire dati nella rete.

1.3 Tecnologie abilitanti

La quasi totalità delle tecnologie che implementano la comunicazione fra dispositivi e fra dispositivi e middleware è di tipo wireless e tipicamente a corto raggio , la particolarità risiede nel fatto che nulla è in realtà stato appositamente progettato per Internet of Things quindi focalizzerò l'analisi più su quali siano vantaggi, svantaggi e modalità di utilizzo nel contesto citato che sull'analizzare in modo estremamente dettagliato la tecnologia stessa.

1.3.1 Wi-Fi

Il Wi-Fi è una tecnologia per la creazione di reti senza fili basato sullo standard IEEE 802.11 (quindi sulla famiglia di protocolli che usano TCP/IP a livello di trasporto), la sua portata varia in base al dispositivo e può passare da pochi metri a centinaia rendendolo perfetto per creare infrastrutture di rete in collegamento ad Internet.

Può operare in due modalità:

- Modalità **ad hoc**: non necessita di un punto di accesso e ogni membro della rete può comunicare con gli altri scambiandosi informazioni, questo può creare molti problemi in quanto la larghezza di banda è divisa per il numero di host e la connessione trova un collo di bottiglia

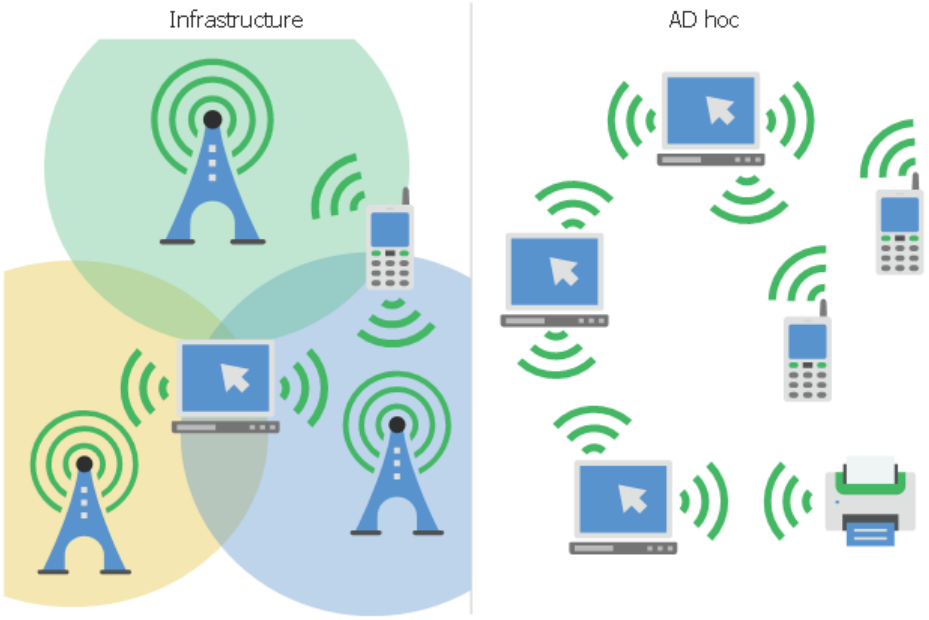

Figura 1.2: Funzionamento della modalità ad hoc ed infrastrutturata del Wi-Fi.

in presenza di un host più lento degli altri in quanto la velocità a cui viaggiano i dati ha un limite superiore.

- Modalità **infrastruttura**: un access point raccoglie tutti i dati trasmessi da ogni singolo host e solo questo access point può mandare informazioni ad un altro host. La larghezza di banda rimane così preservata, e per reti con un numero di membri elevato è possibile utilizzare più di un solo punto di accesso in modo che un dispositivo possa comunicare con l'access point più vicino diminuendo il tasso di interferenze presente nelle connessioni. Aumentando l'affidabilità della rete, si rispetta una di quei principi sui quali qualsiasi buona architettura IoT si deve basare.

In generale questo tipo di tecnologia, come le altre wireless, offre il vantaggio di non essere legata a nessun tipo di cablaggio dei cavi permettendo di inserire sensori in luoghi molte volte non raggiungibili altrimenti, d'altra

parte la richiesta di energia è molto elevata e il sovraffollamento della banda destinata a questo standard può rendere le comunicazioni difficili in ambiti, specie quello cittadino, in cui la presenza di reti Wi-Fi domestiche è cospicua. I costi sono contenuti sia per i moduli che permettono la comunicazione sia per il fatto che le connessioni sullo spettro di banda dedicato allo standard IEEE 802.11 sono libere da licenza.

1.3.2 Bluetooth

Nelle telecomunicazioni Bluetooth è uno standard tecnico industriale per la trasmissione di dati all'interno di WPAN (Wireless Personal Area Network). Utilizza una frequenza radio a corto raggio e ha fra le sue qualità la sicurezza e il costo molto contenuto, un dispositivo può ricercare ogni altro dispositivo che offra lo stesso tipo di comunicazione all'interno del raggio di copertura delle onde radio (qualche decina di metri) ed è la tecnologia abilitante, per esempio, nella comunicazione fra joystick e console, nei computer per la connessione delle periferiche ed è utilizzato dai dispositivi mobile per la connessione all'interno delle moderne automobili con il fine di utilizzare funzioni quali vivavoce e riproduzione di contenuti multimediali.

Anche in questo caso ci sono due tipologie di comunicazione: la *piconet* e la *scatternet*.

Due dispositivi collegati fra di loro formano una piconet, ognuno dei due può essere master o slave, in particolare il master è quello che si occupa della sincronizzazione del clock degli altri slave e della gestione del canale di frequenza alla quale ogni slave si collega. Una piconet pu'o essere di tipo punto-punto oppure punto-multipunto, mentre invece la connessione di più piconet prevede la creazione di una scatternet all'interno della quale ogni dispositivo può essere slave di più piconet contemporaneamente, ma un master può essere slave di al massimo un'altra piconet.

Essendo uno standard di successo, nato nel 1999 grazie a una cooperazione fra le più importanti aziende del settore elettronico tra cui Sony Ericsson, IBM, Intel, Toshiba e Nokia, ha avuto diverse evoluzioni nel tempo che hanno

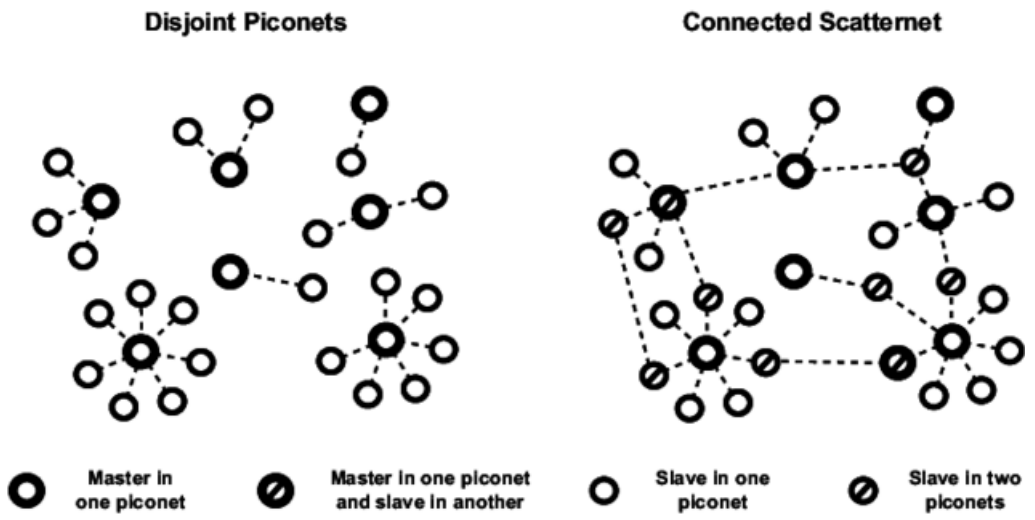

Figura 1.3: Tipologie di rete Bluetooth.

garantito via via sempre prestazione maggiori con un consumo di energia sempre minore. Vista poi l'effettiva adattabilità di questa specifica ai dispositivi di dimensioni e potenza ridotta si è pensato di sviluppare BLE (Bluetooth Low Energy, a volte per questioni commerciali chiamato anche Bluetooth Smart) che opera alle stesse frequenze e con le stesse modalità del normale standard BT ma si rende perfetto per l'integrazione in dispositivi quali beacon, orologi da polso, tastiere, giocattoli o sensori per lo sport. Bluetooh è determinante per l'effettiva realizzazione di Internet of Things [26] e a tal proposito riporto le parole di un'intervista rilasciata da Suke Jawanda, capo marketing del Bluetooth Special Interest Group:

> *"We can credibly say that Bluetooth Smart technology is what has made the Internet of Things a reality. It's no coincidence that the flood of wearables we're seeing come to market today started flowing after Bluetooth Smart was integrated into iOS and iPhone 4s in late 2011. Since then, it's become the de facto standard for fitness devices, smart watches and medical devices from companies as diverse as Nike, Polar, Adidas, Pebble, Samsung, Qualcomm, Sony, iHealth, and Nonin. And with all major mobile operating*

> *systems now offering native Bluetooth Smart APIs, this will only accelerate as manufacturers can create products and associated apps that will run on the smartphones, tablets and PCs that people already own. Going beyond wearables, there are a number of wireless solutions that will help to enable the IoT, but because Bluetooth Smart is standards based, inexpensive and already shipping in billions of devices every year, it's going to be a major part of making the IoT a practical reality. Manufacturers know Bluetooth Smart can help them bring new products to huge markets quickly. We're already seeing this as Bluetooth Smart is being widely adopted in beacons for retailing and payment systems, and growing fast in smart home applications as well."*

Lo standard BLE si basa sulla versione 4.1 del protocollo di cui si possono individuare tre qualità ben precise: *usabilità*, *flessibilità* e *capacità di adattarsi* al mondo IoT. La prima qualità riguarda l'esperienza utente che risulta migliorata grazie a meccanismi di prevenzione verso il mantenimento della connessione e la protezione dalle interferenze con le onde radio emanate, per esempio, dall'uso della telefonia mobile, la seconda si riferisce al fatto che un dispositivo Bluetooth Smart è capace di comunicare con qualsiasi altro tipo di dispositivo bluetooth che implementi anche versioni più vecchie del protocollo. La capacità di adattarsi al mondo IoT fa riferimento a una scelta lungimirante fatta da chi ha progettato la versione del protocollo: infatti esso dispone di un canale di comunicazione IPv6 che permetterà ad ogni singolo dispositivo bluetooth di essere connesso in rete.

1.3.3 ZigBee

ZigBee è il nome utilizzato per indicare una specifica per un insieme di protocolli di comunicazione che basano il proprio funzionamento su antenne radio digitali a bassa potenza, si stima infatti che una rete di sensori che comunica grazie a questa tecnologia possa garantire, con la stessa batteria, le proprio funzionalità per circa due anni; grazie a questa caratteristica, al costo

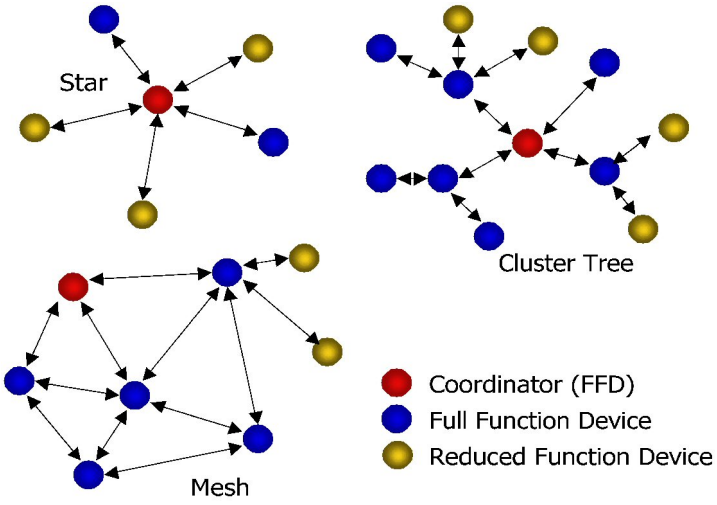

Figura 1.4: Tipologie di rete ZigBee.

contenuto e all'assenza di vincoli nell'utilizzo, ha preso piede soprattuto in ambito industriale e nel settore della domotica. È una tecnologia di tipo wireless, recente e molto utilizzata nei sistemi embedded per creare delle Wireless Personal Area Network (WPAN) in base allo standard IEEE 802.15.4. È una specifica nata nel 2005 per contrastare la diffusione di Bluetooth proponendo un miglioramento delle prestazioni nell'ambito dei dispositivi di dimensioni ridotte costringendo, data la sua riuscita, gli sviluppatori della tecnologia Bluetooth ad adottare miglioramenti che sono poi sfociati nel BLE.

Tipicamente, una rete ZigBee è composta da:

- **Full Function Device**: che possono operare come:
  - **Coordinatore**: detiene il controllo di tutte le informazioni;
  - **Router**: si occupa di instradare correttamente i pacchetti.
- **Reduced Function Device**: sono posti alle estremità periferiche della rete, si preoccupano di agire sull'ambiente circostante o rilevarne dei dati utili.

ZigBee permette di organizzare la rete ed ottimizzarne il controllo. Qualsiasi dispositivo si voglia unire ad essa deve informare il coordinatore, che

fornirà a quest'ultimo un indirizzo di rete valido che verrà impiegato nei pacchetti di rete al posto del MAC Address.

Ci sono nodi router sempre in attesa di ricevere chiamate ed ogni pacchetto che circola per la rete può essere consegnato solo attraversandone uno; quando un router riceve il pacchetto ne rileva la destinazione e verifica se il nodo sia attivo oppure in stand-by. Nella prima ipotesi procede con l'inoltro del pacchetto mentre altrimenti avvia un ciclo di attesa.

L'utilizzo di questo protocollo per connettere in rete dispositivi intelligenti è molto indicata perché si tratta dell'unica tecnologia wireless specificatamente concepita e progettata per fornire soluzioni di rete a basso costo, per sensori a bassa potenza e per il controllo di network che includono una grande varietà di dispositivi.

La possibilità di collegarne fino a 65.000 dispositivi in un'unica rete rende ZigBee, per ora, imbattibile. Grazie a queste qualità ZigBee è l'ideale per collegare le cose che usiamo ogni giorno, per controllare i dispositivi elettronici di consumo, per gestire l'impiego di energia, o per aumentare il nostro confort, ed infine la sicurezza.

1.3.4 RFID

RFID è una tecnologia di comunicazione a bassa potenza importantissima per il mondo IoT, rende infatti possibile l'integrazione di un metodo di comunicazione wireless all'interno di dispositivi a bassissima potenza e di dimensione molto ridotte come per esempio una calamita, determina un consumo di energia praticamente nullo ed è quindi ottimale per essere usato in ogni caso in cui non si abbia accesso a fonti di energia a lunga durata. RFID è acronino di Radio Frequency Identification ed appunto come il nome esteso suggerisce veniva in principio utilizzata per l'identificazione; alcuni esempi sono il tracciamento delle merci, l'utilizzo nei biglietti di accesso ad eventi oppure i sistemi di antitaccheggio. I sistemi RFID sono costituiti da dei tag inseriti dentro gli oggetti che vogliono essere identificati, un lettore e un sistema informatico che, elaborando l'informazione ricevuta dal lettore,

ottiene l'ID del dispositivo, infatti questa tecnologia è caratterizzata da uno standard che permette ad ogni tag di possedere un ID univoco. Dal punto di vista fisico un tag è costituito semplicemente da un chip, un'antenna e un involucro protettivo; si differenziano però in quattro tipologie diverse:

- *attivi*: hanno una batteria al loro interno che ne diminuisce il ciclo di vita, le dimensioni sono maggiori rispetto a quelli passivi ma sono questi tag quelli che possono iniziare una comunicazione.
- *passivi*: sono di piccole dimensioni, non hanno un batteria interna e il costo è molto ridotto. L'energia di cui hanno bisogno in fase di comunicazione riescono a ricavarla dal reader.
- *semi-passivi*: questo tipo di tag, a differenza dei tag solamente passivi, hanno una batteria interna che però alimenta solamente il chip interno e non il trasmettirore, questo ne aumenta il ciclo di vita ma non permette loro di iniziare una comunicazione, rispondono solo alle interrogazioni dei reader.
- *semi-attivi*: dispongono di una batteria ma sono solitamente disattivati e vengono sollecitati nelle comunicazioni solo dal reader.

Il reader è un ricetrasmettitore che contiene una CPU, altre parti digitali di I/O per consentire il collegamento seriale con un computer e componenti di elettronica analogica per trasmissione/ricezione dei dati attraverso un'antenna; è dotato di una fonte di alimentazione autonoma. Le onde radio utilizzate hanno un range molto ampio che va da circa 30 kHz a 3GHz per le microonde, in base alla frequenza di trasmissione si individuano diversi standard definiti da organizzazioni quali l'Auto-ID Center e l'ISO. Per i tag più piccoli si utilizzano frequenze nell'ordine dei kHz (Low Frequency) che permettono delle comunicazioni a distanza di centimetri, da 3MHz a 30MHz si parla di High Frequency che hanno un range di decine di centimetri; questo range di frequenze è quello utilizzato nello standard Near Field Communication (NFC) ISO 14443. Per comunicazioni a più ampio raggio vengono invece utilizzate

delle frequenze molto più elevate come UHF (Ultra-High-Frequency) e Microonde che arrivano fino a 3GHz, qui il consumo di potenza comincia ad essere elevato e vengono utilizzati, per esempio, all'interno dei dispositivi per la lettura dei pedaggi autostradali in velocità. Gli attori nella comunicazione RFID sono: il reader, il tag e un middleware che funge da interfaccia tra applicazione e reader fisico. Il middleware si occupa di interpretare le richieste provenienti dal software applicativo e di comandare il reader utilizzando il protocollo tipico del lettore e la porta di comunicazione a cui è connesso. Viceversa, i dati letti dal reader vengono solitamente riportati verso l'applicazione. Il middleware può utilizzare uno schema di collegamento diretto al reader, fornendo i comandi appropriati direttamente sulla porta di comunicazione oppure si può appoggiare a librerie del produttore dell'hardware che istanziano un reader virtuale con il quale comunicare tramite chiamate API. L'utilità del middleware risiede nella capacità di riuscire a comandare hardware di tipologia differente rendendo quindi indipendente il software applicativo dai componeneti effettivamente utilizzati.

1.3.5 NFC

NFC è una tecnologia di comunicazione radio, proprio come la tecnologia Wi-Fi, Bluetooth e altre. Si contraddistingue da queste perché opera solo a distanza di qualche centimetro (massimo 10). Più precisamente è una serie di specifiche e standard per l'accoppiamento induttivo a radio frequenza (13.56 MHz) per trasferire informazioni tra due dispositivi vicini. Non è una tecnologia nuova, ma sta cominciando a diventare importante in questi ultimi tempi, specialmente perché si sta facendo largo nel mondo degli smartphone. Integrando chip e antenna NFC nello smartphone è infatti possibile realizzare sistemi di pagamento (avvicinando lo smartphone a degli appositi terminali nei negozi), sistemi di condivisione di dati tra dispositivi, accedere a contenuti digitali leggendo smart tags (piccoli chip di sola lettura che si possono trovare in poster e documenti, come badge o passaporti), sistemi di autenticazione (accedere a reti senza dover inserire password e tanto altro, magari

avvicinando lo smartphone al router). NFC è una tecnologia nata dall'evoluzione della tecnologia RFID (quindi un dispositivo NFC può operare con sistemi RFID già esistenti), che permette di trasferire piccole quantità di dati tra due dispositivi che si trovano a pochi centimetri di distanza l'uno dall'altro, senza la necessità di codici di accoppiamento. Il vantaggio su tutte le altre è quindi proprio la semplicità, perché questa tecnologia permette di trasformare lo smartphone in una carta di credito senza fili e tanto altro. Il funzionamento è il seguente: il dispositivo che inizia la comunicazione usa un'antenna per emettere un segnale radio a 13.56 MHz, generando quindi un campo elettromagnetico. Se un dispositivo NFC è presente nel campo, viene attivato e quindi creato un canale di comunicazione. La comunicazione dovrà però avvenire solo con un destinatario, perciò in caso di presenza di più dispositivi si dovrà comunicare solo con uno alla volta. La comunicazione si dice attiva se sia chi inizia la comunicazione sia il destinatario generano il proprio campo, si dice invece passiva se solo chi inizia la comunicazione lo genera. La comunicazione è half-duplex, ovvero che un dispositivo riceve quando l'altro sta trasmettendo. Il bit rate può essere 106, 212 o 424 kbps, quindi non molto veloce, perciò vanno scambiate piccole quantità di dati. I dispositivi NFC si dicono attivi se generano loro il campo (quindi sono alimentati da batteria), e passivi se non lo generano e vengono alimentati dai campi generati da altri dispositivi. Riassumendo, un dispositivo NFC può funzionare in tre modalità diverse:

- **Lettore/scrittore**: in questa modalità, il dispositivo NFC può leggere e modificare dati contenuti in tag NFC passivi (che si possono trovare ad esempio nei poster), permettendo all'utente di recuperare delle informazioni. Per quanto riguarda i consumi energetici, l'energia per generare il campo NFC deve essere fornita dal dispositivo attivo.
- **Emulazione di carta**: un dispositivo NFC può comportarsi come una smart card in questa modalità. Una smart card è una carta tascabile con dei circuiti integrati (solitamente memoria volatile e microprocessore). In questa modalità un lettore RFID esterno non può effettuare

distinzioni tra una carta e un dispositivo NFC. Una smart card può essere emulata sia a livello applicativo sia usando un componente chiamato Secure Element, un dispositivo simile alle vere smart card. Ovviamente questa modalità è utile per i pagamenti elettronici.

- **Peer-to-peer**: questa modalità permette a due dispositivi NFC di stabilire una connessione bidirezionale per scambiare dati. Si distingue qui l'*initiator*, cioè colui che fa la richiesta (client) e il *target*, cioè colui che riceve la richiesta e risponde . L'host non può iniziare la comunicazione se prima non è stato interpellato dall'initiator. Eventuali misure di sicurezza devono essere implementate dallo sviluppatore al livello applicativo, perché gli standard per la comunicazione non speficicano niente da questo punto di vista. Per quanto riguarda i consumi energetici, l'energia richiesta per creare il campo NFC è condivisa tra il targer e l'initiatior.

1.3.6 iBeacon

iBeacon è uno standard definito da Apple che permette alle applicazioni iOS e Android di ottenere dati dai dispositivi iBeacon ricevendo da loro messaggi. È rivolto principalmente a problemi di localizzazione. Questi dispositivi emettono continuamente messaggi in un formato ben preciso, che possono essere captati da smartphone o altri dispositivi con BLE. I messaggi sono composti da:

- **UUID** (*Universally Unique IDentifier*): stringa di 16 bit usata per differenziare tra di loro i gruppi di beacon diversi. Ad esempio se un'azienda ha una rete di beacon, questi avranno lo stesso UUID.

- **Major**: stringa di due bit usata per distinguere sottogruppi all'interno del gruppo più grande.

- **Minor**: stringa di due bit che serve per identificare univocamente il singolo beacon.

Un possibile scenario applicativo può essere il seguente: un utente si trova all'interno di un negozio, e un'applicazione dedicata sul suo smartphone si mette a captare beacon. A seconda dei dati captati l'utente viene poi notificato di informazioni rilevanti che possono tornargli utili.

1.4 Protocolli di rete

Oltre alle tecnologie che permettono la comunicazione tra dispositivi è molto interessante analizzare i protocolli utilizzati al di sopra di esse: a differenza dei terminali nelle normali comunicazioni in rete, il contesto considerato come già detto si compone di dispositivi particolari non adatti all'utilizzo dei tradizionali protocolli di comunicazione come HTTP; quest'ultimo è un protocollo ASCII puramente testuale che insieme alle informazioni utili per ogni richiesta/risposta tende ad inglobare nella sua struttura una quantità di dati elevata e dalle dimensioni non marginali se replicate per ogni singolo messaggio immesso nella rete. Per la comunicazione tra singoli dispositivi o tra dispositivi e un sistema middleware vengono preferiti dei protocolli binari che sono basati sul pattern publish/subscribe e garantiscono un overhead minimo per ogni singolo messaggio.

Il paradigma publish/subscribe permette di creare una sorta di separazione tra il sistema stesso e i soggetti esterni che ne richiedono i servizi, i publisher sono tutte le entità che pubblicano un messaggio o un evento sul sistema mentre i subscriber sono tutti gli altri soggetti che richiedono di ricevere ciò che viene pubblicato e sono sempre a conoscenza dell'intera lista dei publisher e dei loro servizi. Questo disaccoppiamento dei ruoli permette di passare da una centralità del concetto di messaggio nelle comunicazioni alla centralità del concetto di evento e a sistemi event-based. In questo contesto i protocolli di comunicazione maggiormente utilizzati sono:

- **MQTT** (*Message Queue Telemetry Transport*): è un protocollo di comunicazione molto leggero che garantisce affidabilità in reti caratterizzate da una banda limitata e un'instabilità di connessione, utilizza

TCP/IP a livello di trasporto e necessita di un componente intermediaro della comunicazione chiamato broker che distribuisce al client destinatario ogni messaggio.

- **XMPP** (*Extensible Messaging and Presence Protocol*): è un insieme di protocolli aperti di messaggistica istantanea basato su XML. Questo protocollo fa uso di un sistema decentralizzato per la comunicazione molto simile al meccanismo delle mail, chiunque può creare il proprio server e renderlo parte di una rete privata o pubblica, la sicurezza è garantita dalla possibilità di accoppiare ogni trasmissione con delle sessioni SSL o TLS. Essendo uno standard aperto esso può essere personalizzato per renderlo il più adatto possibile ad ogni singola architettura che lo voglia utilizzare, nelle comunicazioni asincrone event-based è stato storicamente alla base di servizi quali Windows Messenger e Yahoo! Messenger; necessita di una codifica in base-64 per i dati prima di essere trasmessi in forma binaria.

- **UPnP** (*Univerasl Plug and Play*): è un protocollo di comunicazione ideato per la semplificazione dell'utilizzo dei dispositivi all'interno della rete, dando la possibilità di utilizzare un dispositivo non appena venga inserito all'interno di essa (o appena venga collegato al computer) tramite la cosiddetta zero-configuration, cioè un meccanismo di configurazione dinamica senza l'aiuto del servizio fornito da un server DHCP. Si presta molto bene al modello di architettura per Internet of Things in cui si introduca un middleware, infatti per garantire un meccanismo leggero di trasmissione indipendente dal mezzo trasmissivo e senza l'ausilio di driver, questo protocollo necessita di una periferica di controllo individuata appunto nel middleware stesso.

- **COaP** (*Constrained Application Protocol*): è un protocollo HTTP-like che mantiene un approcccio request/response eliminando la maggior parte dell'overhead che il protocollo HTTP comporta. Utilizza un for-

mato binario anzich é testuale e si appoggia ad UDP come protocollo di trasporto.

1.5 Middleware per Internet of Things

In informatica il middleware indica un insieme di programmi che fungono da intermediari tra diversi moduli software che compongono una precisa architettura. La sua funzione è quella di dare organizzazione e fornire delle interfacce che nascondano ai livelli superiori l'implementazione e l'organizzazione dei livelli sottostanti. Gli svantaggi dell'assenza di un middleware sono facilmente individuabili: un'inseistente meccanismo di protezione degli accessi, un'assente scalabilità, la non presenza di interoperabilità e l'assenza di un'interfaccia comune per l'accesso agli smart device sono solo alcuni degli inconvenienti che si manifestano senza la presenza di un modulo intermedio nell'architettura. Il Middleware in ambito Internet of Things non viene comunque introdotto solo per garantire un'interfaccia per l'utilizzo, con le stesse metodologie, di risorse differenti, il suo compito va oltre.

Questi sono, raggruppati, i servizi che un buon software di mediazione deve fornire:

- **Connettività e Comunicazione**: il middleware deve garantire interoperabilità a livello di comunicazione fra i dispositivi della rete, per questo in molte architetture si preferisce far passare ogni singola comunicazione attraverso il nodo principale, evitando connessioni dirette fra due peer della rete. Questo serve per agevolare i dispositivi stessi che non devono fornire supporto alle comunicazioni entranti proteggendo sé stessi da attacchi che tentino di compromettere il loro funzionamento sfruttando le porte aperte in ascolto; in secondo luogo nessuno dei peer necessita di conoscere più di un protocollo di rete se tutte le comunicazioni sono gestite dal middleware, infatti esso, conoscendo ogni singolo protocollo usato da ogni dispositivo per la comunicazione, è in grado di fare la traduzione dei messaggi per renderli comprensibili in

modo del tutto trasparente. Se questo non avvenisse invece le possibilità di comunicazione sarebbero limitate oppure bisognerebbe adeguare ogni singolo componente a "parlare la stessa lingua", questo limiterebbe molto le possiblità di espansione delle reti di sensori. Come ultimo vantaggio si garantisce il funzionamento a prescindere dalla topologia che caratterizza la rete di sensori sottostanti.

- **Gestione dei dispositivi**: il middleware deve sempre avere consapevolezza del contesto in cui opera e questo comporta la necessità di gestire ogni minima azione compiuta dai dispositivi. Data la necessità di gestire l'intero ecosistema sotostante, con il termine contesto intendiamo lo stato di ogni dispositivo, le sue funzioni, la sua sicurezza, il suo SW e così via.

  Quindi sicuramente fra le principali funzioni di device managment troviamo:

  - Scoperta dinamica dei dispositivi;
  - Eliminazione dalla rete di dispositivi difettosi;
  - Abilitare e disabilitare funzionalità HW;
  - Aggiornamento firmware e SW da remoto;
  - Localizzazione dispositivi compromessi o smarriti;
  - Reset delle informazioni non più sicure.

  La gestione dei dispostivi e delle risorse fatta a livello di middleware risulta comoda per renderla indipendente da un singolo sistema operativo o da una singola applicazione.

- **Raccolta dei dati, analisi e attuazione**: ogni singola entità che compone la rete di oggetti intelligenti al di sotto del middleware non dispone, per motivi di costi e dimensioni, di supporti adeguati per la memorizzazione dei dati, quindi per evitare che informazioni utili per l'analisi del contesto ambientale nella quale i sensori risiedono vengano

perse, esse sono collezionate in apposite strutture dati di alto livello memorizzate nel middleware oppure, con approcci più moderni, utilizzando un Cloud distribuito con tutti gli ulteriori vantaggi che esso comporta.

- **Scalabilità**: come già anticipato quello della scalabilità per le reti di sensori è un problema molto importante: la capacità elaborativa dei sistemi che si celano all'interno dei piccoli dispositivi che compongono la rete, non permette ad un gran numero di client di porre richieste contemporanee. Esistendo, nei protocolli usati per la comunicazione, dei buffer di dimensione finita (e nel nostro caso anche molto ridotta) eventuali richieste eccedenti il limite smaltibile sarebbero completamente scartate comportando grosse disfunzioni per l'utente finale, il middleware, essendo dotato di HW di fascia elevata, non corre il rischio (a meno di situazioni estreme) di saturare la propria capacità; così, interponendosi come un proxy che permette la gestione di tutte le connessioni entranti smistandole secondo la capacità di ogni singolo sensore, garantisce una QoS sempre elevata. A questo possiamo aggiungere un'ulteriore riflessione: se non esistesse questo livello intermedio, ogni singola scheda di rete di ogni componente della rete dovrebbe essere in grado di interpretare il protocollo HTTP mentre invece frapponendo un nuovo nodo nella comunicazione, questo fa sì che il livello di mediazione si assuma la responsabilità di comunicare con ogni dispositivo, utilizzando il protocollo che esso predilige. Come ultima analisi a livello di scalabilità possiamo considerare che molti dispositivi non sono dotati di sistema operativo, e molte volte neanche di più flussi concorrenti di computazione, quindi garantire un meccanismo come quello appena descritto di separazione delle responsabilità libera loro del tempo utile per rimanere sensibili alle variazioni nell'ambiente, garantendo affidabilità maggiore dell'intero sistema.

- **Sicurezza**: questo è uno degli aspetti più critici quando si parla di

Internet of Things, i problemi che esistono sono sia quelli tradizionali che per ogni rete di calcolatori che espone al mondo i propri dati si vengono a creare, sia problemi specifici che esistono per tecnologie che vengono usate prevalentemente in questo campo. Prendiamo come esempio la tecnologia RFID: essa, tramite il meccanismo dei reader, potrebbe essere facilmente compromessa da lettori non autorizzati o da strumenti che intercettano la comunicazione fra entità autorizzate come ricevitori audio, queste sono solo alcune delle problematiche perch é esistono altri tipi di attachi detti Tag tampering in cui persone malintenzionate potrebbero cambiare il contenuto stesso di uno specifico tag [19]. Inconvenienti simili sono presenti per ogni singola tecnologia abilitante che si usa a livello di interfaccia con il mondo fisico. Oltre a tutte le problematiche particolari per la tecnologia oggetto di studio, ne esistono altre legate ad autenticazione e riconoscimento: sarebbe buona prassi non utilizzare solo username e password per autenticazioni ma gestire la procedura mediante protocolli che si servano di meccanismi di token, cioè di dispositivi hardware personali dotati di generatori pseudocasuali di numeri che si presentano sotto forma di oggetti comuni come portachiavi. Infine il middleware nasconde la rete sottostante e rimane l'unico gateway verso l'esterno della rete come avviene per esempio all'interno di reti domestiche o aziendali con le funzioni offerte dai NAT[1].

1.5.1 Organizzazione

Dopo aver analizzato quelli che sono gli scopi per cui viene introdotto un livello intermedio all'interno dell'architettura procediamo con l'analizzare un modo con cui esso può essere realizzato.

Un'architettura orientata ai servizi (SOA) [12] può essere la scelta migliore per integrare dispositivi di tipo diverso, scelta che può essere lungimirante in ottica di un'integrazione dello stesso middleware, come sarà mostrato nel

[1] `http://www.rfc-base.org/rfc-1631.html`

capitolo successivo, all'interno del web dove tutte le risorse o quasi offrono interfacce RESTful. Insieme a questo la struttura, come per ogni software di dimensioni rilevanti, è organizzata a livelli, e questi sono quelli individuati partendo dall'interfaccia cioè il livello che mette a disposizione tutti i servizi pubblici del middleware:

- **Interface**: a questo livello vengono rese pubbliche l'interfaccia REST e i meccanismi per l'utilizzo delle API fornite per l'interazione con tali servizi. Qui avviene la connessione degli utenti che utilizzeranno i moduli di presentazione delle informazioni oppure la connessione di applicazioni prodotte da terzi che sfruttano l'API, la sua realizzazione è necessaria perché semplifica i livelli inferiori gestendo l'interoperabilità e l'interconnessione dei servizi distruibiti in rete separando lo strato di comunicazione da quello di computazione.

- **Service**: è il livello cruciale del middleware, è qui che vengono implementati i requisiti di scalabilità e interoparabilità. Il programma fornisce una serie di API che possono essere utilizzate dai livelli inferiori per modellare i componenti della rete IoT ed integrare loro in termine di servizi, la modellazione fatta permette la ricerca di precisi servizi all'interno della rete, la connessione di risorse diverse, meccanismi che permettono di utilizzare le informazioni fornite dai servizi in modo sicuro.

- **Networking**: sta alla base di tutto il modello e al di sopra delle sole reti di sensori. Qui deve avvenire la raccolta dei dati e degli eventi che giungono dall'ambiente ed anche quelli che giungono da sistemi esterni come sistemi aziendali, sistemi sanitari o altro in grado a loro volta di influenzare il comportamento dell'intero sistema. A questo livello il middleware deve essere in grado di risolvere tutti i problemi riguardanti l'interoperabilità, la qualità del servizio (QoS) e la sicurezza di connessione. Il supporto di comunicazione deve essere fornito per il

maggior numero di dispositivi diversi sia che si servano di protocolli wireless oppure cablati.

- **Sensing**: a questo livello si trovato i dispositivi che operano facendo rilevazioni sull'ambiente circostante.

1.6 IoT, Cloud e GRID Computing

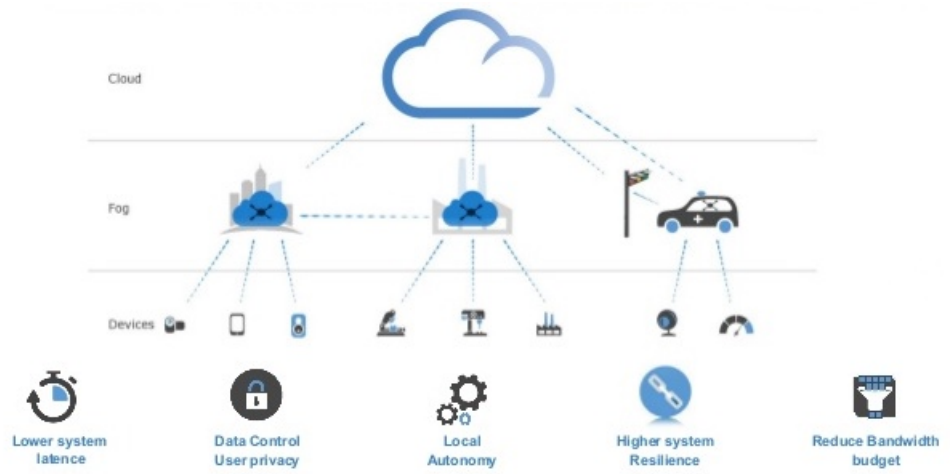

Figura 1.5: Cloud computing e GRID Computing in Internet of Things.

Le due espressioni Cloud Computing e GRID Computing hanno due significati a ni ma differenti fra loro: infatti, per Cloud computing (letteralmente, nuvola informatica) si fa riferimento ad un insieme di tecnologie non solo hardware ma anche software che consentono, mediante un servizio offerto da un provider, di archiviare ed elaborare dati (attraverso CPU o algoritmi vari) inseriti in rete in un modello tipico client-server. In questo modo la potenza computazionale viene resa disponibile 'òn demand", e ciò riduce fortemente costi e sprechi; inoltre, i sistemi Cloud rappresentano per le smart city e per le smart mobility un ambiente nel quale le risorse di calcolo e di immagazzinamento dati sono condivise fra gli utenti e ciò garantisce la scalabilità

delle risorse e lo sviluppo di servizi. Il GRID Computing (in italiano, griglia informatica), invece, si basa fondamentalmente su una serie di computer (abitualmente server), che interconnessi fra loro grazie alla rete Internet, condividono e sfruttano la loro potenza di calcolo per la gestione di una grande quantità di dati. In tutti e due i casi si fa riferimento comunque all'ultimo livello della pila presentata in figura 1.5. La scalabilità delle applicazioni a supporto di Internet of things può essere di due tipi:

- *orizzontale*: raddoppiando il numero di server che ospitano un determinato servizio si ottiene quasi un esatto raddoppio dei potenziali end-user; tale soluzione è utile se il numero di utenti cresce a dismisura e si avvale del load balancing per smistare al meglio il carico di lavoro dei nodi;

- *verticale*: è possibile accrescere le capacità operative agendo direttamente sui componenti hardware del nodo, ottenendo, in questo modo, una maggiore potenza per soddisfare le esigenze.

  I servizi che le piattaforme Cloud forniscono possono essere erogati mediante tre modalità differenti [4]:

  - **SaaS** (*Software as a Service*): indica una modalità di erogazione dei servizi in cui le applicazioni sono eseguite su computer remoti gestiti da terzi e gli utenti possono collegarsi a questi servizi tramite un normale browser. Questo tipo di Cloud computing si riferisce al collezionamento dei dati in modo sicuro ed e ciente a dando la scalabilità del servizio alla società che gestisce il servizio stesso.
  - **PaaS** (*Platform as a Service*): è un tipo di servizio Cloud che mette a disposizione spazio di archiviazione e altre risorse per la raccolta, l'analisi e la memorizzazione di tutti dati provenienti da sensori/dispositivi della rete. I servizi forniti in qusto caso comprendono una serie di funzionalità a supporto di sviluppo/creazione/gestione delle Cloud app.

- **IaaS** (*Infrastructure as a Service*): fornisce invece un'intera infrastruttura (server, spazio di archiviazione, rete). Questa soluzione risulta ideale per chi desidera testare delle nuove applicazioni senza fare lo sforzo di un investimento nell'intera infrastruttura, è il tipico esempio delle start-up.

Il Cloud computing utilizzato come tecnologia indipendente abilita gli utenti ad avere un'astrazione del luogo fisico di archiviazione dei dati assegnando loro delle risorse dinamiche che un provider fornisce come servizio, i dati sono organizzati in maniera invisibile all'utente ma il servizio garantisce che esso abbia sempre la sensazione che tutto ciò che vuole sia a sua completa disposizione; l'unione di questo concetto con il paradigma di Internet of Things porta una quantità notevole di vantaggi e di sviluppo di applicazioni sempre più evolute. Piattaforme di raccolta dei dati vitrualizzate su Cloud permettono lo sviluppo di applicativi, che uniti ai moderni algoritmi di intelligenza artificiale, posso automatizzare i processi decisionali che, sfruttando la potenza di calcolo distribuita, vanno a favore della prevenzione di incidenti o situazioni di pericolo in ambiti come quelli smart city o di automatizzazione dei trasporti. Avere infrastrutture di elaborazione separate dalle unità perifieriche di calcolo, come abbiamo visto, porta all'apertura di un vasto mondo di possibili applicazioni che tuttavia potranno essere realizzate solamente nel momento in cui i protocolli per lo streaming di big data saranno effettivamente implementati e quando scogli come la gestione della privacy e della sicurezza dei dati saranno superati.

Il Cloud computing all'interno dell'architettura individuata nel paragrafo 1.3 si inserisce come ultimo livello all'interno dello stack e rappresenta proprio quel collegamento con i sistemi mediatori di reti di sensori che sarà poi integrato nel web, il mash-up fra un'architettura di questo tipo ed il web porta alla creazione del web of things cioè una piattaforma distribuita di gestione avanzata ed intelligente di qualsiasi smart device con il vantaggio di presentarsi in una forma che garantisce la fruibilità a qualsiasi utilizzatore medio.

1.7 Piattaforme per Internet of Things

Lo scopo di una piattaforma per Internet of Things ha lo scopo di collegare un insieme di sensori a reti di dati. Tali piattaforme collegano la rete di dati ad un insieme di sensori e fornisce approfondimenti attraverso applicazioni di backend per dare un senso alla pletora di dati generati da una rete di sensori.

Alla luce delle possibilità che IoT offre alla società tecnologica essa ha iniziato a capilatizzarla. Al momento sono disponibili molte piattaforme per IoT che forniscono la possibilità di distribuire applicazioni on the go, ovvero che permettono di monitoreare in qualsiasi momento lo stato della propria rete di sensori utilizzando dispositivi mobili (smartphone, tablet e così via).

La differenza tra una piattaforma e un middleware è che una piattaforma consiste in un insieme di strumenti, prodotti e standard utilizzati in combinazione per fornire una soluzione ad un problema; un middleware è un componente di tale soluzione.

1.7.1 Amazon Web Services IoT Platform

AWS IoT è una piattaforma Cloud gestita che consente ai dispositivi connessi, fornendo una comunicazione sicura e bidirezionale, di interagire in modo semplice e sicuro con applicazioni nel Cloud e altri dispositivi. Ciò consente di raccogliere dati di telemetria da più dispositivi e memorizzarli o analizzarli. È inoltre possibile creare applicazioni che consentono agli utenti di controllare questi dispositivi dai loro smartphone o tablet. AWS IoT è in grado di supportare miliardi di dispositivi e migliaia di miliardi di messaggi, ed è in grado di elaborare e instradare tali messaggi agli endpoint di AWS e ad altri dispositivi in modo sicuro e affidabile. Con AWS IoT, le applicazioni rimangono collegate e comunicano con tutti i dispositivi, in qualsiasi momento, anche quando non sono collegati.

AWS IoT è costituito dai seguenti componenti:

- **Device gateway**: consente ai dispositivi di comunicare in modo sicuro ed efficiente con AWS IoT.

- **Message broker**: fornisce un meccanismo protetto che consente alle cose e ad applicazioni AWS IoT di pubblicare e ricevere messaggi. È possibile utilizzare direttamente il protocollo MQTT o MQTT su WebSocket per pubblicare messaggi e sottoscriversi al broker. È possibile utilizzare l'interfaccia HTTP REST per pubblicare messagi.

- **Rules engine**: fornisce l'elaborazione dei messaggi e l'integrazione con altri servizi AWS. È possibile utilizzare un linguaggio basato su SQL [10] per selezionare i dati dal payload dei messaggi o elaborare ed inviare i dati ad altri servizi AWS. È inoltre possibile utilizzare il broker messaggi per ripubblicare i messaggi ad altri sottoscrittori.

- **Security and Identity service**: fornisce la responsabilità condivisa per la sicurezza del Cloud AWS. Le cose devono mantenere sicure le proprie credenziali al fine di poter inviare i dati in modo sicuro al broker. Il broker di messaggi e il motore di regole usano le funzionalità di protezione AWS per inviare in modo sicuro i dati ai dispositivi o altri servizi AWS.

- **Thing registry**: organizza le risorse associate ad ogni cosa. Consente di registrare le proprie cose e associargli fino a tre attributi personalizzati. È inoltre possibile associare certificati e MQTT client ID per ogni cosa al fine di migliorare la capacità di gestire e risolvere i problemi legati al suo funzionamento.

- **Thing shadow**: è un documento JSON utilizzato per memorizzare e recuperare le informazioni sullo stato attuale un dispositivo, un applicazione, e così via.

- **Thing Shadows service**: fornisce rappresentazioni persistenti delle proprie cose nel Cloud AWS. È possibile pubblicare le informazioni di stato aggiornate ad un thing shadow ed è possibile sincronizzarne lo stato di una cosa quando si connette. Le cose possono anche pubblicare

il loro stato attuale ad un thing shadow che può essere utilizzato da applicazioni o dispositivi.

1.7.2 Android Things

Android Things (nome in codice Brillo) è un sistema operativo embedded basato su Android, è stato annunciato per la prima volta al Google I/O 2015. È stato progettato per essere utilizzato da dispositivi IoT a basso consumo di energia e con bassa quantità di memoria RAM (32-64 MB), che sono spesso costruti partendo da differenti piattaforme per microcontrollori.

Avrà il supporto per il Bluetooth Low Energy e il Wi-Fi ed utilizzerà il protocollo Nest Weave[2] per la comunicazione tra dispositivi. Nest Weave è un protocollo di rete introdotto da Nest nel 2015 per fornire comunicazioni sicure, robuste e affidabili tra i dispositivi collegati.

[2] `https://developers.nest.com/weave/`

Capitolo 2

Web of Things

2.1 Cos'è Web of Things

Web of Things è un'espressione usata per indicare un raffinamento di IoT che non consideri solamente un dispositivo intelligente connesso in rete bensì anche immerso nel World Wide Web a livello di applicazione.

L'idea di utilizzare il web come livello applicativo per Internet of Things ha iniziato a diffondersi nel 2007 mentre diversi ricercatori, tra i quali Dominique Guinard e Vlad Trifa, iniziarono a lavorare su questi concetti e pubblicarono un primo manifesto ("Towards the WOT Manifesto") nel quale si sosteneva di utilizzare gli standard del web per creare il livello applicativo del Internet of Things.

L'approccio utilizzato è molto interessante perché di fatto il Web of Things non necessita di nessun nuovo standard o di nessuna nuova tecnologia, semplicemente riusa in maniera intelligente tutto ciò che già oggi esiste per progettare applicazioni web (REST, HTTP, Web Socket, JSON etc.).

L'architettura del modello di Web of Things proposto è organizzata in livelli diversi che saranno esplorati nei paragrafi della sezione, ognuno offre servizi differenti all'interno della struttura delle applicazioni che si vorranno andare a creare e mira a risolvere una specifica problematica che l'inserimento di un dispositivo all'interno del web può comportare:

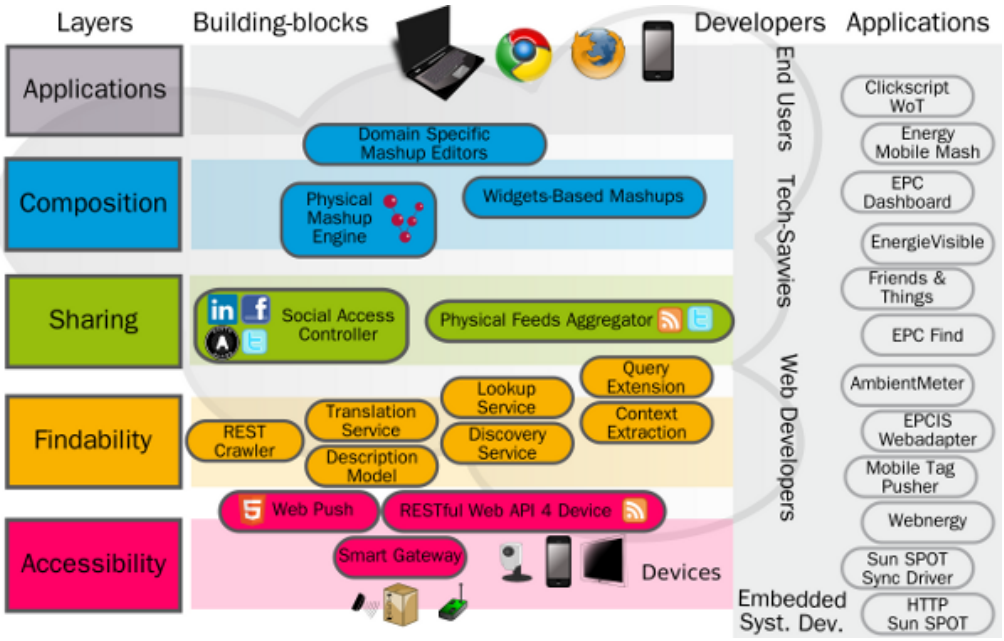

Figura 2.1: Tecnologie ed applicazioni dei diversi livelli che compongono lo stack di Web of Things.

- Livello di *composizione*: questo livello effettua un'attività di combinazione fra le tecnologie web e i dispositivi intelligenti permettendo di creare in modo più semplice applicazioni che utilizzino i servizi dei livelli precedenti. Fornendo web service virtuali per abilitare tutti gli utenti del web ad accedere in modo trasparente alle funzionalità del Web of Things, anche da un semplice browser, combina dati forniti dagli oggetti intelligenti con servizi di presentazione come per esempio Google Maps.

- Livello di *condivisione*: uno dei motivi principali per cui si integrano i dispositivi intelligenti all'interno del web è quello di rendere accessibile al mondo intero una grandissima mole di dati che permetta di migliorare le abitudini e lo stile di vita, a questo proposito lo sharing layer integra i dati con strumenti e applicazioni web di più alto livello che rendono l'analisi dei Big Data effciente, veloce e soprattutto sicura. È a questo livello che vengono implementate le politiche di autenticazione

degli utenti e di protezione dei dati secondo le norme che si riferiscono al codice privacy.

- Livello di *ricercabilità*: la famiglia dei protocolli facenti parte di questo livello si occupa di fornire algoritmi per ricercare particolari servizi o particolari oggetti all'interno del vasto mondo del web. Implementa funzionalità molto simili a quelle che un comune motore di ricerca fornisce nel momento in cui venga posta una richiesta di interrogazione sul contenuto di documenti e pagine web tradizionali.

- Livello di *accessibilità*: è probabilmente, insieme al livello di ricercabilità, il livello più importante all'interno di tutta la struttura perché si occupa di rendere accessibile ogni risorsa del web dotandola di un URL univoco. La piattaforma REST (REpresentational State Transfer) per l'elaborazione distribuita dei dati è a tal proposito la più usata per questo scopo.

2.2 Livello di accessibilità

In questo paragrafo saranno esplorate le tecnologie esistenti il cui utilizzo permette di integrare qualsiasi tipo di risorsa (che sia essa virtuale o fisica) all'interno del web e di poterla raggiungere.

Proprio il termine risorsa non è usato casualmente in questo contesto, la piattaforma sulla quale praticamente qualsiasi tipo di applicazione oggi si appoggia è REST, esso fa della classificazione delle entità in risorse uniche ed indirizzabili il suo concetto fondante, il quale sarà la piattaforma su cui si baserà la discussione del paragrafo riguardante il livello di accessibilità.

2.2.1 Servizi web e architettura REST

Un web service è un sistema a livello software che permette l'interazione fra diverse entità appartenenti alla stessa rete, nell'ottica dell'offerta di servizi

attraverso un'interfaccia che riassuma le funzionalità offerte da uno specifico dispositivo rientra l'architettura REST.

> *"Rest, REpresentational State Transfer, è un'architettura di rete per sistemi distribuiti basati sull'ipertesto, esso delinea come ogni risorsa all'interno del web sia definita e indirizzata."*

(Representational State Transfer, Wikipedia)[1]. Questa tecnologia si differenzia dalle altre presenti per la creazione di servizi web perché non utilizza nessun livello opzionale per il proprio funzionamento: si appoggia ad HTTP per trasmettere i dati sfruttando tutte le opzioni che tale protocollo mette a disposizione; questo non avviene ad esempio nel protocollo SOAP (Simple Object Access Protocol)) che pur rientrando nella categoria degli standard adibiti alla chiamata remota di procedure, sfrutta semplicemente il protocollo HTTP per far trasportare dati che rispettino la formattazione tipica XML-WSDL (Web Services Description Language) e sono inclusi nel payload dei pacchetti.

Con l'avvento di applicazioni RESTful all'interno del web si è iniziato a parlare di Web 2.0, detto anche web dinamico, per sottolineare il fatto che un approccio di questo tipo ha aumentato l'interazione fra utenti e applicazioni online che a differenza del web statico, il quale offriva semplicemente una consultazione di documenti, permette agli utenti l'accesso a servizi quali blog, social network ed uso interattivo di piattaforme di e-commerce.

L'idea centrale di applicazioni REST risiede nel fatto di catalogare qualsiasi entità sotto forma di risorsa; oltre a questo naturalmente esistono altre prerogative che devono essere rispettate:

- Il web deve essere basato sull'identificazione univoca delle risorse attraverso indirizzi URI (Uniform Resource Identifiers).
- Ogni risorsa deve essere disponibile attraverso un'interfaccia che ne riassuma le funzionalità. HTTP fornisce un insieme di comandi (GET,

[1] `http://it.wikipedia.org/wiki/Representational_State_Transfer`

PUT, DELETE, POST, HEAD, etc.) che possono essere utilizzati per dialogare con una risorsa; questo insieme di comandi, se usato con una semantica ben precisa, è suffciente per concludere qualsiasi tipo di comunicazione con ogni risorsa. Per qualsiasi tipo di comunicazione si intendono le tipiche operazioni CRUD (Create, Read, Update, Delete).

- La comunicazione avviene con messaggi auto-esplicativi che permettono di eliminare la fase di negoziazione individuale del formato, in questa ottica sono molto utilizzati JSON e XML che oltre ad essere molto leggeri, offrono una perfetta integrazione con i linguaggi di scripting più comuni quali JavaScript e PHP.

- La comunicazione fra client e server deve essere priva di stato: la richiesta deve contenere al suo interno tutte le informazioni che il server necessita per l'interpretazione e per fornire una risposta adeguata; anche se si fa uso di meccanismi quali cookies per informare il server dell'identità del client, questi permettono sì di informare il server sull'identità dell'utente che si cela dietro la richiesta appena giunta ma non obbliga i due attori della comunicazione a gestire una particolare sessione. La principale ragione di questa scelta è la scalabilità: mantenere lo stato di una sessione ha un costo in termini di risorse sul server e all'aumentare del numero di client tale costo può diventare insostenibile. Inoltre, con una comunicazione senza stato è possibile creare cluster di server che possono rispondere ai client senza vincoli sulla sessione corrente, ottimizzando le prestazioni globali dell'applicazione.

REST, fornendo queste linee guida, non specifica mai come debba avvenire l'effettiva implementazione e lascia libero il programmatore di utilizzare poi la tecnologia che reputa essere la migliore; ad oggi per sviluppare applicazione di questo tipo la tecnologia AJAX (Asynchronous JavaScript and XML) è la più usata.

2.2.2 Rintracciabilità e rappresentazione delle risorse all'interno del web

Uno scoglio tecnologico molto importante che si è posto nella realizzazione del Web of Things è sicuramente quello di avere ogni risorsa raggiungibile in modo univoco attraverso un proprio identificatore.

Applicando il modello RESTful, otteniamo la conclusione che l'assegnazione di un URI ad ogni singola risorsa ci consente di ottenere la rappresentazione voluta, ma anche se il modello applicato ci suggerisce una soluzione, bisogna studiare un'architettura che dia ordine al numero elevato di entità che necessitano di tale identificazione.

In quest'ottica un modello basato su organizzazione gerarchica ed uso di astrazione permette di raggiungere ogni risorsa in modo molto simile a come si può raggiungere ogni singolo file all'interno del file system di un sistema operativo. Un URI è infatti una stringa, che essendo in linguaggio naturale, mantiene una semantica che permette di descrivere ogni risorsa con un nome dotato di significato e permette l'utilizzo del plurale per catalogare risorse aggregate. Un esempio di URI:

```
http://example.com/resources/buttons/1
```

Un approccio alternativo consiste nell'assegnare ad ogni risorsa una stringa che funge da identificatore univoco all'interno della rete di dispositivi in modo tale da avere accesso diretto ad ogni risorsa senza ricorrere a strutture ad albero come illustrato nel caso precedente. Un esempio di URI:

```
http://example.com/resources/AlarmBtn
```

La struttura del collegamento alle risorse è fatta invece dotando ogni entità di una lista ai figli e un puntatore al padre, la risoluzione degli URI nel web è fatta direttamente dal protocollo HTTP, quello che comunemente viene fatto in un browser anteponendo "http://" all'URL di una pagina web. Sfruttando il servizio offerto dal protocollo HTTP e la descrizione tramite

URI delle risorse possiamo in modo univoco nel web risolvere il problema della rintracciabilità di qualsiasi dispositivo intelligente.

Anche se questo meccanismo ci offre la possibilità di contattare ogni dispositivo, non avremmo modo di conoscere quelle che sono le sue funzionalità se non si progettasse un modo che permetta ad esso di comunicare al client la propria interfaccia.

Il design della rappresentazione delle interfacce è fondamentale per non far restare una risorsa un concetto astratto di dispositivo della quale non si conosca nessuna operazione e che non sia in grado di fornire nessun utile servizio; la soluzione a questo problema sta proprio nel ricercare un modo che permetta una comunicazione fra peer senza che sia necessaria una corposa fase di negoziazione. L'uso dei tipi MIME (*Multipurpose Internet Mail Extensions*), supportati da HTTP, combinato con HTML oppure altri formati di rappresentazione delle informazioni come JSON o XML, è sufficiente per fornire l'interfaccia della risorsa contattata: nelle comunicazioni machine to machine si preferisce l'utilizzo di formati di serializzazione come appunto JSON perché permettono una facile conversione delle informazioni ricevute in oggetti (e.g. Java) che poi possono essere utilizzati nelle maniere più svariate dalla logica del software, mentre per la presentazione delle informazioni ricevute ad un utente risulta essere molto più intuitiva una classica pagina HTML che, oltre a fornire una vista più comprensibile dei dati, mette a disposizione tramite link l'accesso all'intera gerarchia di risorse.

Individuato il modo per rintracciare ogni risorsa, bisogna capire come si possa interagire con un dispositivo. Prendiamo l'esempio di un sensore di temperatura: sarà molto probabile che la sua interfaccia offra operazioni per interagire con il valore della temperatura nell'ambiente in cui questo sensore è posto, una volta che abbiamo contattato l'URI che identifica la risorsa "sensore di temperatura" ci aspetteremo una risposta contente tutti i servizi che sono messi a disposizione dal dispositivo intelligente (per esempio in formato JSON) con la tipologia di richieste che esso è in grado di interpretare. Per comunicare le operazioni ammesse, e quindi i comandi HTTP accettati

da un dispositivo, viene molte volte utlizzata la cluasola OPTIONS di tale protocollo che restituisce proprio una risposta con la lista delle operazioni consentite.

Utilizzando questo meccanismo, una volta ottenuta la rappresentazione dell'oggetto, con un'altra semplice richiesta HTTP si può utilizzare uno dei servizi messi a disposizione: sfruttando una richiesta di tipo GET, per esempio, ci sarà dato il valore corrente di temperatura, con una richiesta di tipo PUT possiamo far eseguire un comando al sensore o ordinare lo spegnimento (accensione) del dispositivo stesso.

Con meccanismi analoghi vengono utilizzate tutte le 5 tipologie di richieste HTTP, inoltre, rispettando le prerogative che il modello REST suggerisce, è facilmente apprezzabile il fatto che tale meccanismo (a meno della conoscenza della rappresentazione) permetta ad ogni client di inviare richieste e di accedere ai servizi senza che lo stato della connessione debba essere mantenuto. Sempre sfruttanfo tutti i meccanismi che il protocollo HTTP ci mette a disposizione, un web service fornito da uno oggetti intelligenti può segnalare anomalie e malfunzionamenti utilizzando i codici di stato delle risposte HTTP.

2.3 Livello di ricercabilità

Questo livello ha la peculiarità di offrire tutti quegli strumenti che possano far diventare ogni risorsa non solo accessibile da qualsiasi entità facente parte del web, ma anche ricercabile e catalogabile da altre applicazioni.

Difatti, seppur il processo di integrazione è reso possibile dai meccanismi già discussi, nel web sono presenti miliardi e miliardi di URI che rappresentano altrettante risorse. Quindi è necessario escogitare una maniera per indicizzare tutti i dispositivi intelligenti in previsione di una crescita importante che si prevede porterà alla creazione di un ecosistema in cui debbano convivere sempre un numero maggiore di servizi forniti attraveso il web.

Senza questo livello ogni tipo di risorsa in rete, qualsiasi sia la sua natura, risulterebbe non utilizzabile a meno che non si conosca a priori il suo URI.

In un certo senso la sfida che si propone è molto simile a quella che ha portato i motori di ricerca ad evolversi sempre più per fornire all'utente che utilizza il web nella sua forma tradizionale i documenti che più siano attinenti a quella che è la ricerca che esso desidera, se oggi utiliziamo tutti quanti noi il web avendo a disposizione ciò che esigiamo con un semplice click lo dobbiamo al lavoro fatto dagli ingegneri per sviluppare algoritmi che soddisfino nel modo più che sufficiente ed esaustivo qualsiasi tipo di ricerca eseguita. Ma se nel web tradizionale il lavoro che il motore di ricerca esegue è fatto scandendo ed indicizzando documenti in forma di ipertesto, nel Web of Things tutto questo deve essere fatto su sensori e altri dispositivi fisici caratterizzati da comportamenti altamente dinamici e necessitano la risoluzione di parametri di ricerca che filtrano la totalità dei dispositivi combinando quelle che sono le informazioni fornite dalle interfacce REST. Il problema che si pone, generalizzando, riguarda l'indicizzazione dei servizi in generale fatta con un'analisi che proponga una soluzione valida nell'interazione machine to machine e human to machine, tale soluzione risiede nell'uso di strutture di metadati, cioè dati che descrivono altri dati, che si rendano comprensibili ed utilizzabili dai motori di ricerca offrendo allo stesso tempo parametri che permettano agli utenti di filtrare i servizi nel modo desiderato. Infine i motori di ricerca tradizionali basati sull'indicizzazione dei documenti web utilizzano algoritmi che assumono il fatto che i cambiamenti all'interno delle pagine avvengano in tempi non molto rapidi permettendo una maggiore effcienza, questa supposizione è completamente non valida se parliamo di oggetti che forniscono per la maggior parte servizi real-time.

Prendendo come esempio un sensore che si debba autodescrivere fornendo una struttura di metadati, questo dovrà certamente caratterizzarsi tramite una serie di proprietà che possiamo definire statiche se riguardano parametri che durante tutto il suo ciclo di vita non varieranno, fra questi alcuni descriveranno le sue informazioni di costruzione e i servizi per cui è stato

configurato; altre proprietà, dette *dinamiche*, invece si riferiranno a quello che è lo stato attuale del sensore (o del dispositivo in generale) ed i dati che riguarderanno le sue ultime rilevazioni compresi anche meccanismi di segnalazione di guasti o di malfunzionamenti. Un utente che voglia interfacciarsi con un'architettura di Web of Things ottimale ha l'esigenza di porre le interrogazioni più svariate che possono rivolgersi alla selezione di particolari caratteristiche sia ponendo delle condizioni sulle proprietà statiche (e.g. tutti i sensori di prossimità ad infrarossi) oppure sulle proprietà dinamiche (e.g. tutti i sensori di prossimità posizionati nel raggio di 5 km dalla posizione attuale attivati negli ultimi 5 minuti).

L'integrazione di queste funzionalità appena descritte all'interno dei motori di ricerca e dei browser utilizzati oggi deve prescindere dall'utilizzo di formati per la codifica delle proprietà dei dispositivi da questi ultimi comprensibili; per dare semantica agli elementi delle pagine web uno dei formati più utilizzati è lo standard microformats che è interpretato dalle piattaforme di ricerca più famose quali Google e Yahoo.

2.3.1 Microformat

Microformat [2] è una parte di mark-up che si inserisce all'interno di HTML 5 e che permette di dare semantica alle pagine web. Ha la peculiarità di adattarsi in modo effciente nel WoT in quanto mantiene il giusto compromesso tra la comprensibilità agli utenti e alle macchine, come suggerito anche dalla definizione fornita da Drew McLellan [30] (componente del gruppo creatore del formato):

> *"Microformats are a way of attaching extra meaning to the information published on a web page. This extra semantic richness works alongside the information already presented, and can be used for the benefit of people and computers. This is mostly done*

[2] `http://microformats.org`

through adding special pre-defined names to the class attribute of existing XHTML markup."

Come per quasi ogni parte dell'architettura WoT anche l'utilizzo di questo formato con l'obiettivo di dare una semantica alle pagine web prevede il riuso di tecnologie già esistenti: infatti partendo dall'analisi di un primo esempio di microformat ci si può immediatamente accorgere di come la sintassi sia immersa all'interno di una normale pagina di presentazione HTML: l'utilizzo dell'attributo `class`, e dei tag `div`, `span` etc.

```
<div class="vcard>
    <div class="fn">John Doe</div>
    <div class="org">The example company</div>
    <div class="tel">604-555-1234</div>
    <a class="url">http://www.example.com</a>
</div>
```

Microformat è in realtà uno standard unico che contiene diversi formati standardizzati ognuno dei quali a sua volta è dedicato alla descrizione di una categoria di risorsa diversa: nell'esempio è mostrato un microformat di tipo vcard che serve per descrivere un formato di file per biglietti da visita elettronici. Come è possibile notare il `div` principale, che contiene tutte le informazioni della persona, specifica il tipo di formato utilizzato e i tag, gerarchicamente sottomessi a quello principale, sfruttando l'attributo `class`, elencano ogni singola proprietà; se questo è un semplice esempio che serve per introdurre il meccanismo nel suo formato più elementare, tanti altri microformat permettono la descrizione delle risorse (anche fisiche) più svariate. Qui di seguito un elenco dei microformat più utilizzati e di conseguenza quelli che quasi tutti i motori di ricerca e i browser sono in grado di interpretare:

- *geo*: è un microformat molto utilizzato nell'ambito della descrizione di oggetti intelligenti, permette tramite i suoi attributi di geolocalizzare i dispostivi e darne una descrizione delle proprietà statiche e dinamiche.
- *h-product*: permette l'identificazione di prodotti, dei brand commerciali e la catalogazione tramite specifici tag.

- *h-card*: come già detto permette, seguendo le linee dettate dallo standard v-card, la descrizione degli individui.

- *h-calendar*: permette la rappresentazione di eventi ed è per esempio largamente usato da organizzazioni quali Facebook e Wikipedia.

- *rel-tag*: questo forse è quello più interessante nel contesto analizzato. È un metadato che rientra all'interno della categoria dei microformat, di fatto sono dei termini, delle parole associate alla descrizione di un servizio o di una risorsa che permettono la loro classificazione, ed una volta inseriti nel web i motori di ricerca permetteranno ad un utente di navigare tra le risorse caratterizzate dai soli tag desiderati. Per meglio comprenderne il significato, questo microformat è alla base dei comuni *hashtag*.

L'interpretazione di questo formato inserito all'interno del normale codice HTML contenuto nelle risposte dei server è sempre più diffusa, e a testimonianza del fatto esiste un progetto lanciato dalla Mozilla Foundation che tramite una particolare estensione chiamata Operator, sul suo browser Firefox, permette la visualizzazione personalizzata delle informazioni raccolte durante la navigazione, ed interpretate tramite il parsing dei microformat; inoltre sempre più applicazioni web, interpretando i dati che giungono tramite la rete, riescono ad adattare la vista allo specifico utente in base ad una specifica richiesta. L'esportazione dei dati anche su applicazioni desktop permette di conservare la semantica delle informazioni, se un servizio web volesse indicare la propria posizione, per esempio, essa potrebbe essere interpretata attraverso questo standard anche da applicazioni di mappe che mostrerebbero in tempo reale la localizzazione.

Concentradosi nell'ambito degli oggetti intelligenti, notiamo come attraverso questo semplice meccanismo possiamo descrivere molti parametri di ogni oggetto a partire dalle informazioni di produzione e di proprietà del singolo dispositivo creando attraverso i tag dei percorsi di navigazione fra risorse correlate (per esempio utilizzando un tag descriva il brand di produzione).

La cosa forse più importante ed il motivo per cui si è scelto di parlare del microformat come piattaforma a livello di ricercabilità nell'architettura di WoT descritta è *hRESTs* [20]: esso, seppur ancora non è stato riconosciuto come uno standard dagli enti internazionali non permettendone di fatto una vasta diffusione,

integra con lo stesso meccanismo appena descritto anche tutte informazioni che descrivono i servizi di tipo REST che una risorsa nel web può offrire, portando con sé anche i vantaggi di indicizzazione che una futura e auspicabile integrazione da parte dei motori di ricerca possono comportare.

2.3.2 Infrastruttura di ricerca per i dispositivi

Se il meccanismo dei microformats permette di fornire una rappresentazione semantca alle risorse del web, d'altra parte esso è limitato da una serie di fattori che il contesto oggetto della discussione impone. Per prima cosa si deve ragionare sul fatto che stiamo parlando di un contesto in continua evoluzione che prevede l'ingresso e l'uscita delle varie risorse in modo completamente dinamico e imprevedibile, si rende quindi necessaria, al fine di ottenere una perfetta ricercabilità, la creazione di un'infrastruttura che gestisca tutte le evenienze che si possono presentare.

Il primo problema che si pone è sicuramente quello di avere un meccanismo capace di reagire all'ingresso e all'uscita di risorse dalla rete, cosa che può avvenire molto frequentemente, dal momento che gli oggeti fisici che compongono il Web of Things non necessariamente sono sempre connessi alla rete e non sono esenti da guasti o altre eventualità. Per risolvere il problema dell'avvio sono presenti alcuni protocolli che permettono di auto-registrare e segnalare nel web i propri servizi, alcuni di questi sono Service Location Protocol (SLP), Universal Plug and Play (UPnP), Device Profile for Web Service e Apple's Bonjour; per quanto riguarda una vera e propria infrastruttura che possa al meglio gestire tutte le funzionalità desiderate dal contesto ideale del WoT bisogna ancora aspettare. Al giorno d'oggi protocolli e idee non mancano ma nessuno di questi è diventato uno standard, i dispositivi sono collegati ancora in delle reti che risultano essere sufficienti solo per usi privati e risultano in un certo senso ancora isolati.

2.4 Livello di condivisione

L'architettura che fino ad ora si è proposto di costruire porta con sé una falla di non poco conto, con i meccanismi descritti si può mettere a rischio la privacy dei dati emessi dai web service forniti da ogni dispositivo intelligente; se non esi-

ste una politica di regolamentazione dell'accesso alle risorse chinque può venire a consocenza di dati sensibili che non gli appartengono.

In linea con quella che è stata la politica di riuso fino ad ora utilizzata, con l'obiettivo di creare la nostra struttura di Web of Things, valutiamo la possibilità di utilizzare HTTP con il suo sistema di autenticazione che permette, accoppiato con SSL/TLS per la crittografia, di implementare un meccanismo di identificazione nelle richieste indirizzate al server.

Questa pratica è una buona prassi per collegamenti con server quando l'autenticazione ci permette di accedere a dati personali ma così non è se pensiamo al collegamento a dispositivi intelligenti: il numero dei dispostivi che fungono da server in questo tipo di comunicazione può essere molto elevato (se elevato è il numero di sensori che in prospettiva faranno parte del WoT) e di conseguenza anche il numero di account e di autenticazioni necessarie per proteggere le connessioni e i dati sensibili lo sarebbe, creando disagi per l'utente. Questo scenario è quindi inimmaginabile e si manifesta allora la necessità di avere un meccanismo piùleggero ed effciente che permetta sia di effettuare ogni connessione ad un dispositivo intelligente in maniera sicura sia di non gravare eccessivamente sull'esperienza d'uso dell'utente e sul degrado di prestazione che l'impegno nella gestione del riconoscimento del client da parte degli oggetti intelligenti può comportare.

Una soluzione per la condivisione delle informazioni partendo da un qualcosa che già esiste sono i social network: lo sviluppo in tema di privacy, avuto da tutte le piattaforme social negli ultimi anni, garantisce meccanismi altamente affidabili e non facilmente violabili per la protezione delle informazioni e la condivisione con le sole persone che rientrano all'interno della "rete" personale di ogni utente. Tutti i social network utilizzano delle strutture a grafo per la memorizzazione delle relazioni di ogni utente, essi grazie alla loro diffusione globale costituiscono uno spaccato molto realistico su quelli che sono i rapporti esistenti tra le persone e l'introduzione di un livello di autenticazione social al di sopra dei dispositivi intelligenti porta a definire il concetto di Social Web of Things [7].

Inoltre il mercato odierno necessita dell'integrazione di tutte queste piattaforme social anche all'interno, per esempio, delle applicazioni mobile ed esistono grazie a questo API che sono largamente utilizzate a questo proposito (Open Social una di queste), i vantaggi nell'implementare questa strategia sono svariati: per

prima cosa si avvicinerebbero in modo più trasparente gli utenti alle nuove tecnologie rendendo la loro esperienza facilitata dalla fiducia che in tutte le piattaforme social già ripongono, una piattaforma nuova che non si poggia su nulla di esistente avrebbe bisogno di un tempo di avvio fisiologico che in questa maniera non è necessario perché i Database ed i grafi sono già esistenti. L'integrazione potrebbe poi essere a doppio filo se si pensa al modello di condivisione delle informazioni che tutti quanti utilizziamo semplicemente pubblicando messaggi sui profili social delle persone a noi vicine: in questa maniera se volessimo condividere un nuovo sensore potremmo semplicemente renderlo accessibile pubblicando una notizia sul nostro social network preferito.

Un'architettura di questo tipo, con un livello di condivisione che si vede reso sicuro e protetto appoggiandosi sullo strato dei social, necessita di un controller capace di gestire autorizzazioni e autenticazioni collezionando tutti i gateways intelligenti [15] e i dispositivi intelligenti che un utente registra. La sua funzionalità è quella di essere un proxy interposto fra i middleware che gestiscono le singole reti di sensori ed i sistemi che invece governano i social, il controller può quindi tenere traccia delle relazioni e delle interazioni fra i sensori/attuatori e gli utenti e permetterebbe la realizzazione dei sistemi di registrazione per le notifiche di eventi importanti, l'accesso in lettura e scrittura potrebbe essere regolato dal proprietario di ogni dispositivo semplicemente abilitando alcuni o tutti i metodi proposti da HTTP. Se per esempio ipotizzassimo il metodo PUT utilizzato come meccanismo per fare l'impostazione delle proprietà di un dispositivo intelligente, il proprietario potrebbe riservare tale funzione solo a una rete ristretta di persone a lui collegate tramite le relazioni social e il controller, che è il proxy in tale comunicazione, bloccherebbe accessi non autorizzati con questo tipo di richiesta HTTP.

La creazione di applicazioni web che implentino la funzionalità di controller per l'accesso e la condivisione dei dispositivi intelligenti in ambito Social Web of Things, come già accennato, è resa possibile dall'esistenza di una larga varietà di API; il primo passo per entrare a far parte del processo di condivisione delle risorse è l'autenticazione ad un server controller, e al riguardo fra i vari protocolli esistenti si potrebbe scegliere OAuth 2.0 [16] che è uno standard open-source sviluppato mettendo al centro del progetto la sicurezza degli utenti. Esso per il suo funzionamento fa uso di HTTP integrandosi in modo eccellente all'interno del

web e, attraverso l'uso dei cookies, non deve esistere sessione fra server e client; ogni richiesta seguente alla prima riporterà nel proprio header un cookie che farà riconoscere al server l'identità dell'utente. Il secondo passo nella realizzazione del nostro controller per Social WoT sta nel recupero delle informazioni sulle dipendenze sociali dell'utente appena autenticato: per questa funzionalità si può usare un protocollo come OpenSocial [3]; il suo compito è quello di recuperare una serie di connessioni sicure estrapolando i dati dai grafi dei social network mediante servizi offerti da interfacce di tipo REST ed elaborandoli, così si potrà creare una rete sicura di condivisione delle informazioni fornite dai dispositivi di cui l'utente autenticato è proprietario. Se si volessero però integrare piattaforme come Facebook o Twitter, queste non sono supportate dai protocolli open-source fino ad ora citati e necessitano invece di API che fanno uso di protocolli proprietari, questo è un fattore negativo perché si scontra con quei principi di interoperabilità di cui IoT e WoT si fanno portatori.

Gli scenari applicativi interessanti di questa integrazione tra Web of Things e social network sono molteplici se si aggiunge un ulteriore strato di configurazione e registrazione di eventi generati dai sensori: infatti mediante dei meccanismi di polling da parte del server sui sensori o push da parte dei sensori verso il server [11] il controller già citato per la regolamentazione degli accessi può generare in modo automatico degli eventi che, essendo poi notificati, comunicano alla rete di persone desiderate quello che effettivamente sta accadendo. Per essere più chiari possiamo immaginare lo scenario in cui una persona necessita di far sapere ad altre, per qualsiasi tipo di evenienza, il momento in cui abbandona l'uffcio e quindi lascia il posto di lavoro: se un sensore fosse impostato per essere sensibile alle variazioni di corrente che sono presenti sul PC dell'interessato, un controller configurato per generare eventi al di sopra o al di sotto di certe soglie potrebbe notificare tali eventi in modo del tutto automatico facendo risparmiare del tempo agli utilizzatori.

Tale controller ha sicuramente un'architettura software stratificata e non di banale implementazione, così come ogni signolo oggetto intelligente ed ogni singolo gateway intelligente forniscono un tipo di accesso mediante architettura REST, anche per quanto riguarda questo importante componente dell'infrastruttura del Web of Things l'accesso alle sue funzionalità deve essere regolamentato mediante

[3] `http://opensocial.org`

tale tecnologia. Analizzando le funzioni a grandi linee si possono delineare dei moduli in cui strutturare il software e la sua business logic:

- **Proxy e gateway**: questo modulo si deve occupare della verifica delle credenziali e del redirect di tutte le richieste di connessione nel caso in cui colui che tenta di stabilire una comunicazione con un dispositivo intelligente ne abbia il diritto.
- **Connessione ai social network**: questo modulo deve offrire in modo trasparente tutte le funzionalità complementari per stabilire un collegamento con i social che l'utente possiede, e per ricevere quindi tutte le informazioni riguardanti la rete di conoscenti della specifica persona. È in questa parte che la logica del programma sfrutta le API per interfacciarsi con i social network.
- **Manager di autenticazione**: questo modulo si frappone fra i due precedenti e si occupa di gestire la rete di connessioni presenti fra utenti e social network in modo del tutto trasparente evitando ogni volta la richiesta delle credenziali di accesso, le impostazioni sulla possibilità di utilizzo dei cookie teoricamente possono essere proprio fatte su questo livello.
- **Aggiornamento e registrazione**: questo modulo si occupa di gestire ogni evento ed ogni singolo componente intelligente registrato. Collezionando le preferenze degli utenti gestisce la notifica di eventi generati dai sensori restando quindi sensibile a tutte le variazioni che nell'ambiente avvengono.

2.4.1 User experience

All'interno di quelle che sono le dinamiche del livello di condivisione rientra un progetto di studio molto affascinante promosso dalla Ericsson[4], i suoi ingegneri sono tra i primi che circa nel 2011 cercarono di trovare un nesso che potesse collegare Internet of Things alle persone. È infatti importante per diffondere l'uso di una nuova tecnologia a cui la massa degli utenti non è abituata riuscire dare loro un disegno, un modello mentale che le permetta di associare il meccanismo

[4] `http://www.ericsson.com/uxblog/2012/04/a-social-web-of-things/`

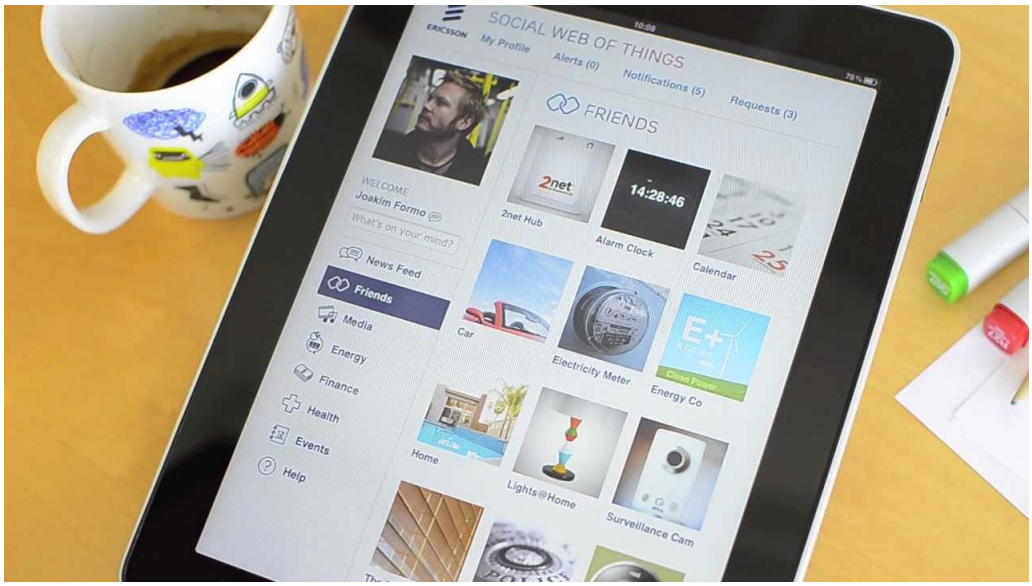

Figura 2.2: Prototipo di un'applicazione per Social Web of Things.

che si cela dietro all'integrazione nel web degli oggetti della vita di tutti giorni; il modello a cui gli ingegneri pensarono fu proprio quello dei social network: essi già introducono i concetti di amicizia e di legame sociale nel web che assolutamente sono necessari per permettere un meccanismo che induca sicurezza e fiducia verso la tecnologia da parte dell'utente. Avvicinare l'utilizzatore medio a questo mondo è una sfida che va vinta per convincere chiunque di quali siano gli effettivi vantaggi che si possono avere nella vita quotidiana ma questo va fatto offrendo una user experience concreta: in ogni livello dello strato che si va a costruire al di sopra del singolo dispositivo fisico l'astrazione deve arrivare ad una raffinatezza sempre maggiore fino a che la percezione dell'oggetto sia quella di un'entità in grado ragionare e dialogare con noi. A questo propostito possiamo pensare di dotare di un proprio profilo social ogni singolo dispositivo, ad esso possiamo mandare messaggi testuali che percorrendo lo stack dei livelli sottostanti il profilo del dispositivo saranno trasformati, con degli opportuni parsing, in comandi ben precisi; già oggi esistono assistenti vocali evoluti che sono in grado di capire la semantica delle nostre frasi e quindi lo scenario ipotizzato non è del tutto improbabile. Certamente per arrivare al livello descritto molte ricerche dovranno essere compiute e molti mezzi tecnologici dovranno essere dispensati, ma pensare ad immagini come quelle

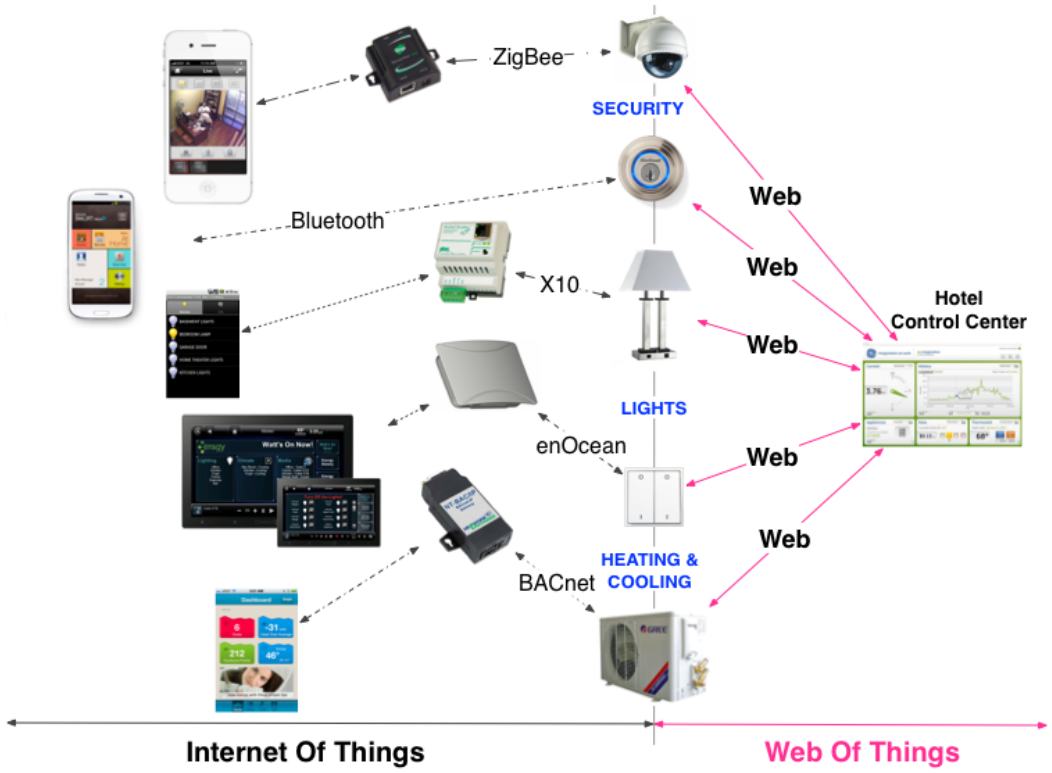

Figura 2.3: Combinazione tra dispositivi fisici e servizi web.

in figura 2.2 non è ad oggi così utopistico.

2.5 Livello di composizione

In questo livello si cerca di avvicinare gli sviluppatori alla creazione di applicazioni Web 2.0 che utilizzino i dati e le funzionalità fornite dalla rete di sensori globale, qui il focus si sposta maggiormente verso la parte di front-end e il livello di presentazione dei dati.

Nel web dinamico assume grossa importanza il concetto di aggregazione intesa come aggregazione di contenuti e funzionalità provenienti da fonti esterne (API, RSS o JavaScript) a livello di presentazione; nel tempo questo meccanismo ha preso sempre più piede e di conseguenza sempre maggiore è il numero di tool (Google App Engine o Yahoo Pipes) messi a disposizione per avvicinare anche i

meno esperti alla sua realizzazione, è possibile idearne uno senza scrivere nemmeno una riga di codice. L'alternativa ai tool è rappresentata sempre dall'utilizzo delle interfacce REST o SOAP fornite da un applicativo web attraverso linguaggi di programmazione come PHP, ASP.NET o Java.

Per comprendere meglio cosa si intende per mashup prendiamo come esempio i maps mashup, molto utilizzati nel web moderno: essi sono mashup dove i dati di una determinata fonte vengono visualizzati all'interno di mappe; per questo scopo vengono molto usati i servizi di Google Maps. Quest'ultimo permette la visualizzazione di mappe di tutto il mondo ed offre l'accesso libero alla piattaforma Maps, quindi a tutti i suoi servizi e a tutte le web API, affiancati da un'ottima documentazione con lo scopo di incentivare gli utenti a sviluppare sempre nuovi servizi. Un esempio di maps mashup è FlickrViewer, l'unione tra Google Maps e Flickr, che permette di posizionare le proprie foto, caricate in Flickr, sulla mappa del mondo. Esistono anche mashup riguardanti il mondo dell'e-commerce, degli esempi sono quelli realizzati per Amazon.com ed eBay: permettono di avere sotto controllo dei prodotti a cui si è interessati per l'acquisto, oppure di effettuare ricerche o confrontare i prezzi fra varie offerte provenienti da altri siti.

All'interno dell'architettura fino ad ora esposta per il Web of Things ci siamo fermati alle linee guida per un buon livello di condivisione basato sull'utilizo dei social network, tale livello essendo parte del web 2.0 sicuramente fornirà un'interfaccia RESTful a cui riferirsi per la creazione di applicazioni user-friendly, sarà lui la nostra fonte di informazioni per creare il mashup desiderato, ogni sviluppatore dovrà solo dare sfogo alla propria fantasia per cercare di creare la migliore applicazione possibile.

2.6 Esempi di piattaforme per Web of Things

2.6.1 EVRYTHNG

EVRYTHNG [5] è una piattaforma dedicata agli sviluppatori che vogliono utilizzare le funzionalità del Web of Things, fondata nel 2011 da Niall Murphy e Andy Hobsbawm, essa si occupa di gestire l'identità digitale dei dispositivi intelligenti

[5] `https://evrythng.com`

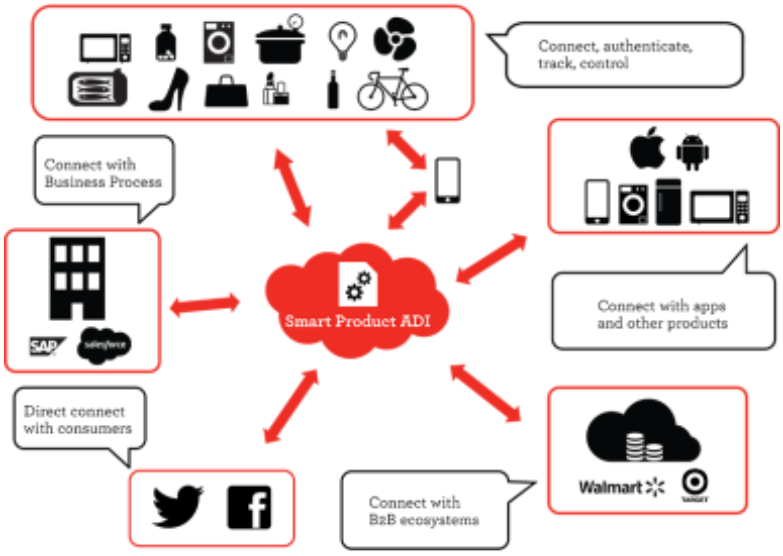

Figura 2.4: Integrazione degli Active Object Identifier in oggetti di uso quotidiano e collegamento con i social network, dispositivi mobili, ecosistemi per il commercio interaziendale e così via.

collegati nel web e fornisce supporto per la creazione di applicazioni real-time che utilizzino tali dispositivi.

Iscrivendosi, vengono forniti la documentazione e gli strumenti necessari per immettere i propri dispositivi intelligenti all'interno del web, la caratteristica di ogni oggetto che venga a far parte di tale piattaforma è l'assegnazione di un ADI (Active Digital Identity) cioè un identificatore univoco all'interno del Cloud dell'organizzazione; una volta conclusa questa operazione un oggetto diverrà effettivamente un Web Object che sarà utilizzabile in una vasta gamma di modalità diverse come indicato in figura 2.6.1 sfruttando tutte le API sviluppatore messe a disposizione.

L'Active Digital Identity per un oggetto fisico ha capacità semantiche, è in grado di fornire descrizioni standard mediante metadati che consentono alle applicazioni di comprendere e utilizzare i servizi dell'oggetto e alla piattaforma stessa di fornire servizi di browsing all'interno dell'insieme dei prodotti registrati. Gestiti come oggetti Web in un processo di sviluppo software, dispositivi fisici possono essere facilmente collegati ad altre risorse web ed entità come social network, re-

cord di CRM (Customer Relationship Management) [21], sistemi ERP (Enterprise Resource Planning) [3] e così via. L'identià online può essere rafforzata generando QRcode e tag NFC specifici che rappresentano l'URI della risorsa nel web in modo univoco; gli ADI permettono inoltre la catalogazione per funzionalità dinamiche o caratteristiche statiche dei dispositivi, e le API offrono la possibilità di generarem in maniera non molto complessa, integrazioni con alcuni tra i più famosi servizi online come Flickr o Google Maps, con il vantaggio della gestione dei servizi attraverso una piattaforma basata su Cloud computing. Per meglio capire cosa si intende per identità digitale e Web Object vediamo il seguente esempio: per la registrazione di un oggetto presso l'account di uno sviluppatore avviene uno scambio di messaggi HTTP fra client e piattaforma, lo sviluppatore diventerà il proprietario dell'oggetto con la possibilità di gestire tutte le possibili configurazioni che rigurdano la ricercabilità nel web attraverso degli appositi tag.

Ogni sviluppatore ha a disposizione una dashboard con una GUI molto intuitiva che gli permette di creare progetti a cui verranno collegate una o più applicazioni, l'organizzazione prevede che il creatore del progetto ne divenga il proprietario e possa decidere quali siano i diritti di utilizzo da parte degli altri utenti dei servizi offerti dai dispositivi che, gerarchicamente, appartengono a tale progetto. Ogni applicazione prodotta da terzi può utilizzare i servizi dei dispositivi attraverso le API sviluppatore solo se ne conosce la chiave di accesso, e in generale le applicazioni si devono registrare, sempre attraverso l'uso della dashboard dedicata agli sviluppatori, al progetto specificando delle chiavi per l'utilizzo delle API (il meccanismo è visibile nella richiesta HTTP già menzionata per la registrazione dei dispositivi). Gli utenti delle applicazioni si devono registrare anch'essi attraverso la piattaforma che in modo automatico manterrà gli accessi sicuri e separati attraverso il proprio *authentication manager*, ognuno degli utenti finali avrà la possibilità di configurare quali siano le proprie preferenze in modo che la piattaforma, raccogliendo i dati dai dispositivi fisici, possa elaborare le informazioni per determinare quali siano le notifiche di eventi che ogni utente debba ricevere in modo personalizzato.

Il progetto collegato a questa piattaforma ha riscontrato notevole successo e non poche sono le aziende leader del settore tecnologico che se ne sono interessate: fra il 2011, anno di lancio, ed oggi gli investimenti giunti sono nell'ordine dei milioni di dollari e la base operativa si è estesa sia all'Europa che all'America contando la

registrazione di svariati prodotti fabbricati in tutti e cinque i continenti.

2.6.2 WeIO

WeIO[6] è una piattaforma hardware e software open source per la prototipazione e la creazione di oggetti interattivi connessi con collegamenti senza fili utilizzando solo linguaggi popolari nell'ambito della programmazione web come HTML5, CSS3 e JavaScript per la programmazione lato client e Python 2.7 per la programmazione lato server.

WeIO utilizza come implementazione del meccanismo Zeroconf [6] *Apple Bonjour*[7]; tale meccanismo consente di individuare automaticamente la presenza di dispositivi WeIO nella rete locale.

È possibile accedere al file system di ogni dispositivo WeIO attraverso SSH [31] e Samba[8]. Per programmare un dispositivo è possibile collegarsi al dispositivo desiderato attraverso un web browser e utilizzare il web editor integrato per scrivere il codice sorgente, oppure scrivere il codice sorgente sulla propria macchina e copiarlo successivamente all'interno della memoria del dispositivo.

[6] `http://we-io.net/hardware/`

[7] `https://developer.apple.com/bonjour/`

[8] `https://www.samba.org`

Capitolo 3

Piattaforma per Web of Things

3.1 Obiettivo

L'obiettivo è consentire ai dispositivi intelligenti di pubblicare aggiornamenti riguardo lo loro stato e di alterarlo attraverso un'apposita interfaccia.

Questo problema potrebbe essere facilmente risolto introducendo un web server al di sopra di una piattaforma per IoT già presente (Android Things[1], IoTivity,[2] Apple HomeKit[3], Oracle IoT platform[4], etc.) come gateway verso il web.

Nonostante questa soluzione sia semplice e pratica da implementare ha lo svantaggio di non includere la maggior parte dei dispositivi presenti sul mercato dal momento che tali piattaforme esistono solo per alcuni. Dal momento che il nostro obiettivo è quello di creare una piattaforma universale nella quale deve essere possibile integrare i dispositivi più disparati questa soluzione non è sufficiente e dobbiamo guardare oltre.

Tra le tipologie di dispositivi non integrati da queste piattaforme vi sono i microcontrollori; i quali sono dispositivi special-purpose che eseguono un singolo programma alla volta, sono privi di sistema operativo, hanno un consumo energetico minore rispetto ad altre tipologie di dispositivi come i System-on-a-Chip e

[1] `https://developer.android.com/things/`
[2] `https://www.iotivity.org`
[3] `https://developer.apple.com/homekit/`
[4] `https://www.oracle.com/us/solutions/internetofthings`

sono particolarmente indicati per applicazioni nelle quali è fondamentale il rispetto delle scadenze temporali.

Il programmatore in questo caso può avere pieno accesso all'hardware, fino a poter rimuovere il bootloader al fine di evitare ritardi durante l'avvio del dispositivo e guadagnare spazio sulla memoria non volatile, oppure sostituirlo con una versione personalizzata specifica per il compito che il microcontrollore andrà a svolgere.

3.2 Organizzazione del sistema

Nel sistema saranno presenti due tipologie di dispositivi:

- **Gateway**: sono dispositivi general-purpose (e.g. System-on-a-Chip) che hanno una moderata capacità computazionale, fungono da punto d'ingresso per i messaggi che interessano i dispositivi presenti nella rete e da punto di uscita per i messaggi rappresentanti lo stato di un dispositivo;
- **Dispositivi intelligenti**: sono dispositivi che acquisiscono informazioni dall'ambiente in cui sono situati (e.g. rilevazione della temperatura, rilevazione della quantità di luce nell'ambiente, etc.) ed interagiscono con l'ambiente (e.g. accensione di una lampadina, attivazione di un allarme sonoro, etc.).

I requisiti fondamentali del sistema saranno:

- Permettere la comunicazione con i dispositivi più disparati;
- Gestire la connessione di nuovi dispositivi alla rete;
- Gestire la disconnessione di dispositivi già in uso.

Tutto ciò dovrà essere eseguito mentre il sistema è in esecuzione e permetterà di renderlo estremamente dinamico ai cambiamenti.

3.2.1 Organizzazione dei gateway

Ogni gateway sarà responabile di eseguire i seguenti compiti:

- Ricevere una richiesta proveniente dall'esterno, nel nostro caso utilizzando un'interfaccia RESTful;
- Propagare la richiesta al dispositivo interessato;
- Ricevere la risposta dal dispositivo, che può contenere o i dati della risposta (nel caso la richiesta vada a buon fine) o un messaggio che descriva l'errore che non ha consentito alla richiesta di andare a buon fine;
- Propagare la risposta a colui che ha effettuato la richiesta;
- Propagare gli aggiornamenti sui cambiamenti rilevati nell'ambiente nel quale il dispositivo è situato.

Per raggiungere tale obiettivo si propone di organizzare ogni gateway nei tre livelli software che verranno dettagliati di seguito.

Interfaccia web

Questo livello software è rapppresentato da un'interfaccia RESTful e WebSocket [24] che verranno utilizzate per ricevere aggiornamenti "real-time" riguardo l'aggiornamento dello stato dei sensori presenti nell'ambiente. WebSocket consente di implementare una comunicazione full-duplex attraverso una singola connessione TCP/IP, ciò consente ad un client di ricevere messaggi di aggiornamento senza dover effettuare una nuova richiesta e dover instaurare una nuova connessione tra il client e il server ogni volta che ha necessità di comunicare.

I messaggi ricevuti saranno trasmessi ad un secondo livello software responabile della gestione dei messaggi provenienti dall'interfaccia web e dai dispositivi sottostanti ed indirizzarli al giusto destinatario.

Gestione dei messaggi

Questo livello software lo scopo di gestire le richieste effettuate e le *risposte* o gli *aggiornamenti* ricevuti dai dispositivi.

Per *risposta* si intende il messaggio ricevuto (che può essere di successo o di errore) dal dispositivo a fronte di una richiesta effettuata. Per *aggiornamento* (o *notifica*) si intedende un messaggio che il dispositivo produce a fronte di una varianzione rilevata nell'ambiente.

Queste due tipologie di messaggio avranno ovviamente una politica di gestione differente in quanto le risposte sono indirizzate ad un unico destinatario (colui che ha effettuato la richiesta) e le notfiche saranno destinate a tutti coloro che sono in ascolto sul canale di comunicazione[5].

Comunicazione con i dispositivi

Questo livello software è responsabile di inoltrare le richieste ai dispositivi e riceverne le risposte o aggiornamenti riguardo ai cambiamenti rilevati.

Poiché le tecnologie abilitanti sono molteplici (Wi-Fi, Bluetooth, Bluetooth Low Energy, ZigBee, etc.) questo livello software dovrà essere il più modulare possibile, dovrà fornire la medesima interfaccia di programmazione a prescindere dalla tecnologia abilitante e consentire di poter scambiare tecnologia agilmente a seconda di quella più indicata per l'utilizzo specifico senza avere ripercussioni sulla logica applicativa o sul codice già presente.

3.2.2 Organizzazione dei dispositivi intelligenti

Ogni dispositivo intelligente sarà equipaggiato con una libreria adibita alla comunicazione con il gateway assegnatogli. Tale libreria dovrà fornire le primitive di comunicazione che il programmatore utilizzerà per notificare aggiornamenti sullo stato del dispositivo e rispondere alle richieste inoltrategli.

3.3 Infrastruttura per la comunicazione

Come infrastruttura per la comunicazione si è scelto di utilizzare un **Message-Oriented Middleware** (in breve MOM); è un'infrastruttura per sistemi distribuiti dove è richiesta un'elevata comunicazione interna, quindi i componenti hanno necessità di condividere informazioni tra loro affinché altri possano processarle.

Tale infrastruttura permette di ignorare i dettagli dei sistemi operativi, dei protocolli di rete e l'utilizzo di risorse condivise, a tal fine i processi sfruttano una

[5] `https://it.wikipedia.org/wiki/Canale_(telecomunicazioni)`

comunicazione a scambio di messaggi[6], invece di chiamate a procedure remote o memoria condivisa, utilizzando come primitive di comunicazione `send` e `receive`.

Il vantaggio principale nell'utilizzare un MOM è la possibiiltà di disaccoppiare i componenti.

Questa infrastruttura consente di costruire un'**Event-driven architecture** (in breve EDA).

Un'architettura guidata dagli eventi è composta in quattro entità:

- **Creatore di eventi**: è la sorgente dell'evento, sa solo che un evento si è verificato;
- **Consumatore di eventi**: ha necessità di sapere che un evento è avvennuto affiché possa reagire di conseguenza, potrebbe essere conivolto nella sua elaborazione o potrebbe essere semplicamente influenzato da esso;
- **Notificatore di eventi**: funge da uomo di mezzo e si occupa di diffondere gli eventi scatenati dai creatori agli opportuni consumatori;
- **Evento**: variazione di stato rilevata dal software o dall'hardware, nel nostro caso sono rappresentati dalla ricezione dei messaggi scambiati tra i vari componenti nel sistema.

Il vantaggio di utilizzare un'architettura guidata dagli eventi sta nel fatto che permette a un gran numero di produttori e consumatori di scambiare informazioni quasi in tempo reale.

L'utilizzo di un MOM per la costruzione di un'EDA consente di costruire un sistema software eterogeneo estremamente reattivo, ad alte prestazioni che garantisce continuità nei servizi erogati.

3.3.1 Rappresentazione dei messaggi

Per la rappresentazione dei messaggi si è scelto di utilizzare il formato JSON[7] (*JavaScript Object Notation*), nonostante sia basato sul linguaggio JavaScript ne è indipendente ed è adatto per l'interscambio di messaggi tra applicazioni.

[6] `https://it.wikipedia.org/wiki/Comunicazione_a_scambio_di_messaggi`
[7] `http://www.json.org/json-it.html`

È preferito rispetto ad altri formati (e.g. XML[8]) per via della sua essenzialità, facilità di lettura e scrittura e il forte supporto offerto dai linguaggi di programmazione.
Le strutture dati su cui è basato sono:

- Un insieme di coppie *chiave/valore* (oggetti), per separare la chiave dal valore si utilizzano i doppi punti (`:`), un insieme di coppie è racchiuso tra parentesi graffe (`{...}`) ed ogni coppia è separata dalla successiva da una virgola (`,`). Una chiave è una sequenza di caratteri racchiusa tra doppi apici (`"..."`), mentre un valore è un qualsiasi tipo di dato supportato dal formato JSON.
- Un elenco di *elementi* (array), ogni elenco è racchiuso tra parentesi quadre (`[...]`) ed ogni elemento è separato dal successivo da una virgola (`,`). Un elemento può essere un qualsiasi tipo di dato supportato dal formato JSON.

I tipi di dato supportati sono:

- Booleani (`true` e `false`);
- Numeri (interi, reali, virgola mobile);
- Sequenze di caratteri, racchiuse tra doppi apici (`"..."`);
- Oggetti;
- Array;
- Nessun valore (`null`).

3.3.2 Semantica dei messaggi

Per descrivere la semantica dei messaggi si è scelto di utilizzare JSON Schema[9].

JSON Schema è un vocabolario che consente di annotare e verificare documenti JSON, ha una documantazione pulita e facile da interpretare da parte di umani ed elaboratori. Consente una verifica strutturale completa, utile per la verifica dei dati ricevuti e per l'automazione del collaudo del software.

[8] `https://it.wikipedia.org/wiki/XML`
[9] `http://json-schema.org`

Semantica delle richieste

```
{
  "$schema": "http://json-schema.org/schema#",
  "type": "object",
  "description": "Representation of request messages.",
  "oneOf": [
    ...
  ]
}
```

Questa porzione del documento descrive la semantica di una richiesta: utilizza la versione corrente dello schema per descrivere il formato del messaggio, il messaggio è rappresentato da un oggetto e deve soddisfare esattamente una delle descrizioni che verrano dettagliate di seguito.

```
...
{
  "properties": {
    "action": {
      "type": "string",
      "enum": ["INQUIRY"]
    },
    "query": {
      "type": "object",
      "properties": {
        "duration": {
          "type": "integer",
          "minimum": 1
        }
      },
      "additionalProperties": false,
      "required": ["duration"]
    }
  },
```

```
  "additionalProperties": false,
  "required": ["action"]
}
...
```

La richiesta per ricercare i dispositivi disponibili dovrà possedere una proprietà `action` con valore `INQUIRY`; opzionalmente potrà avere una proprietà `query`, il quale valore deve essere un oggetto che dovrà avere una proprietà `duration` che rappresenta la durata in secondi della ricerca, il suo valore dovrà essere un intero strettamente maggiore di zero.

```
...
{
  "properties": {
    "action": {
      "type": "string",
      "enum": ["READ"]
    },
    "resourceId": {
      "type": "string",
      "pattern": "^[a-zA-Z_\\$][\\w\\$]*$"
    },
    "query": {
      "type": "object",
      "minProperties": 1
    }
  },
  "additionalProperties": false,
  "required": ["action", "resourceId"]
}
...
```

La richiesta per leggere lo stato di una risorsa dovrà possedere una proprietà `action` con valore `READ`, una proprietà `resourceId` che rappresenta l'identificativo

della risorsa sulla quale si intende effettuare l'operazione di lettura e opzionalmente un oggetto `query` che, se presente, non potrà essere vuoto.

```
...
{
  "properties": {
    "action": {
      "type": "string",
      "enum": ["WRITE"]
    },
    "resourceId": {
      "type": "string",
      "pattern": "^[a-zA-Z_\\$][\\w\\$]*$"
    },
    "payload": {
      "type": "object",
      "minProperties": 1
    },
    "query": {
      "type": "object",
      "minProperties": 1
    }
  }
}
...
```

La richiesta per modificare lo stato di una risorsa dovrà possedere una proprietà `action` con valore `WRITE`, una proprietà `resourceId` che rappresenta l'identificativo della risorsa sulla quale si intende effettuare l'alterazione di stato, un oggetto `payload` non vuoto che rappresenta il corpo della richiesta e opzionalmente un oggetto `query` che, se presente, non potrà essere vuoto.

Semantica delle risposte

```
{
```

```
  "$schema": "http://json-schema.org/schema#",
  "type": "object",
  "description": "Representation of response messages.",
  "oneOf": [
    ...
  ]
}
```

Questa porzione del documento descrive la semantica di una risposta: utilizza la versione corrente dello schema per descrivere il formato dei messaggi, un messaggio è rappresentaoi da un oggetto e dovrà soddisfare esattamente una delle descrizioni che verrano dettagliate di seguito.

```
...
{
  "type": "object",
  "properties": {
    "response": {
      "type": "object",
      "properties": {
        "message": {
          "type": ["object", "array"],
          "minItems": 1,
          "minProperties": 1
        },
        "error": {
          "type": "string",
          "minLength": 1
        }
      },
      "oneOf": [
        {
          "required": ["message"]
        },
```

```
        {
          "required": ["error"]
        }
      ],
      "additionalProperties": false
    }
  },
  "required": ["response"]
}
...
```

I messaggi provenienti dal gestore della comunicazione dovranno specificare una proprietà `response`. Tale proprietà dovrà specificare o la proprietà `message`, che può essere o un oggetto non vuoto o un array non vuoto, o la proprietà `error` che descrive (sotto forma di stringa) l'errore che si è verificato.

```
...
{
  "type": "object",
  "properties": {
    "response": {
      "type": "object",
      "properties": {
        "resourceId": {
          "type": "string",
          "pattern": "^[a-zA-Z_\\$][\\w\\$]*$"
        },
        "message": {
          "type": ["object", "array"],
          "minItems": 1,
          "minProperties": 1
        },
        "error": {
          "type": "string",
```

```
        "minLength": 1
      }
    },
    "oneOf": [
      {
        "required": ["resourceId", "message"]
      },
      {
        "required": ["resourceId", "error"]
      }
    ],
    "additionalProperties": false
  },
  "event": {
    "type": "object",
    "properties": {
      "resourceId": {
        "type": "string",
        "pattern": "^[a-zA-Z_\\$][\\w\\$]*$"
      },
      "message": {
        "type": ["object", "array"],
        "minItems": 1,
        "minProperties": 1
      },
      "error": {
        "type": "string",
        "minLength": 1
      }
    },
    "oneOf": [
      {
        "required": ["resourceId", "message"]
```

```
        },
        {
          "required": ["resourceId", "error"]
        }
      ],
      "additionalProperties": false
    }
  },
  "oneOf": [
    {
      "required": ["event"]
    },
    {
      "required": ["response"]
    }
  ]
}
...
```

I messaggi provenienti da una risorsa dovranno specificare o una proprietà `response`, quando il messaggio è stato prodotto a fronte di una richiesta effettuata da un agente esterno, o una proprietà `event`, quando il messaggio è stato prodotto a fronte di una variazione rilevata da uno dei sensori con cui è equipaggiato. In entrambi i casi dovrà specificare una proprietà `resourceId` che contiene l'identificativo del dispositivo ed una proprietà `message`, che può essere o un oggetto non vuoto o un array non vuoto, in alternativa una proprietà `error` che descrive (sotto forma di stringa) l'errore che si è verificato.

3.4 Descrizione delle operazioni

In questa sezione verrano descritte, corredate da esempi, le operazioni che la piattaforma per Web of Things renderà disponibili ai suoi utilizzatori.

3.4.1 Richiedere la lista delle risorse vicine

In risposta ad una richiesta HTTP `GET` al percorso `/inquiry` la piattaforma dovrà resistuire una lista degli identificativi delle dispositivi vicini.

Parametri di interrogazione

Nome	Tipo di dato	Opzionale	Descrizione
`duration`	`integer`	Sì	Durata della ricerca (in secondi).

Esempio

```
--> REQUEST
GET http://example.com/api/inquiry?duration=4


<-- RESPONSE
200 OK
[
    "button",
    "led"
]
```

3.4.2 Richiedere la lista delle risorse connesse

In risposta ad una richiesta HTTP `resources` al percorso `/inquiry` la piattaforma dovrà resistuire una lista degli identificativi delle risorse connesse.

Esempio

```
--> REQUEST
GET http://example.com/api/resources


<-- RESPONSE
200 OK
[
```

```
    "button",
    "led"
]
```

3.4.3 Connettere una risorsa

In risposta ad una richiesta HTTP `POST` al percorso `/resources` la piattaforma dovrà connettere la risorsa indicata oppure rispondere con un messaggio di errore nel non sia possibile connetere la risorsa indicata o sia già connessa.

Corpo della richiesta

Nome	Tipo di dato	Descrizione
`resourceId`	`string`	Identificativo della risorsa.

Esempio

```
POST http://example.com/api/resources

{
    "resourceId": "button"
}



<-- RESPONSE
204 NO CONTENT
```

3.4.4 Richiedere lo stato di una risorsa

In risposta ad una richiesta HTTP `GET` al percorso `/resources/:resourceId` la piattaforma dovrà resistuire lo stato corrente della risorsa.

Parametri di percorso

Nome	Tipo di dato	Descrizione
`resourceId`	`string`	Identificativo della risorsa.

Esempio

```
--> REQUEST
GET http://example.com/api/resources/button


<-- RESPONSE
200 OK
{
    "pressed": false
}
```

3.4.5 Aggiornare dello stato di una risorsa

In risposta ad una richiesta HTTP `PUT` o `PATCH` al percorso `/resources/:resourceId` nel cui corpo vi sono specificate le proprietà della risorsa che si andranno ad aggiornare la piattaforma dovrà aggiornare lo stato corrente della risora e restituire il suo stato aggiornato.

Parametri di percorso

Nome	Tipo di dato	Descrizione
`resourceId`	`string`	Identificativo della risorsa.

Esempio

```
--> REQUEST
PUT http://example.com/api/resources/led

{
    "on": true
}


<-- RESPONSE
200 OK
```

```
{
    "on": true
}
```

3.4.6 Disconnettere una risorsa

In risposta ad una richiesta HTTP `DELETE` al percorso `/resources/:resourceId` la piattaforma dovrà disconnettere la risorsa indicata oppure rispondere con un messaggio di errore nel non sia possibile disconnettere la risorsa indicata o non sia connessa.

Parametri di percorso

Nome	Tipo di dato	Descrizione
`resourceId`	`string`	Identificativo della risorsa.

Esempio

```
--> REQUEST
DELETE http://example.com/api/resources/led


<-- RESPONSE
204 NO CONTENT
```

Capitolo 4

Implementazione del prototipo

In questo capitolo verranno introdotte le tecnologie e le librerie utilizzate durante lo sviluppo del prototipo, in particolare gli ambienti di sviluppo Vert.x[1] (Java) e PyBluez[2] (Python) per la programmazione dei gateway e Wiring[3] (C/C++) per la programmazione dei dispositivi intelligenti, successivamente verranno illustrati i dettagli delle analisi delle prestazioni dei vari componenti.

4.1 Vert.x

In questa sezione si andrà ad esplorare che cosa è Vert.x, come funziona, quali vantaggi porta rispetto a soluzioni concorrenti (come Node.js[4]) ed infine si andrà a descrivere brevemente che cos'è il sistema di moduli e quali di essi sono stati utilizzati per lo sviluppo del prototipo.

4.1.1 Cos'è Vert.x

Citando il sito ufficiale: *"Eclipse **Vert.x** is a tool-kit for building **reactive** applications on the **JVM**."*

In queste poche parole possiamo già individuare alcune parole chiave:

[1] `http://vertx.io`
[2] `https://github.com/karulis/pybluez`
[3] `http://wiring.org.co`
[4] `https://nodejs.org`

- *a tool-kit*: Vert.x è una libreria, può essere introdotto come dipendenza di un progetto esistente ed essere utilizzato assieme alle librerie e ai framework che già che si stanno utilizzando.

- *reactive*: consente, grazie alla sua natura *non bloccante* e *guidata dagli eventi*, di sviluppare sistemi reattivi ad alte prestazioni utilizzando un piccolo numero di thread di sistema. In altre parole Vert.x consente di far scalare un'applicazione utilizzando una quantità minima di hardware.

- *on the JVM*: nonostante i suoi moduli siano scritti in Java, Vert.x è supportato anche altri linguaggi: JavaScript, Groovy, Ruby, Ceylon, Scala e Kotlin. Ciò consente di scrivere applicazioni in linguaggi diversi e addirittura di utilizzare linguaggi diversi all'interno della stessa applicazione.

Se si intende utilizzare Vert.x all'interno della propria applicazione è necessario ottenere un'istanza della classe `Vertx`, attraverso la quale è possibile accedere alle funzionalità reattive e non bloccanti che la libreria mette a disposizione.

Ogni istanza della classe `Vertx` è configurabile attraverso un'istanza della classe `VertxOptions`, questa classe consente di personalizzare la configurazione di default con la quale verrà instanziata. Tra le opzioni configurabili sono presenti: il massimo numero di worker thread che potranno essere utilizzati dall'istanza, il numero massimo di event loop thread che potranno essere utilizzati, se l'istanza dovrà essere raggruppata con altre istanze o meno, etc.

Le API di Vert.x sono asincrone e guidate dagli eventi, nel caso il programmatore sia interessato ad un evento può registrare una procedura che verrà chiamata successivamente quando l'evento avverrà, in altre parole non è necessario attendere l'arrivo di un'informazione se questa non può essere reperita immediatamente.

Nell'ecosistema di Vert.x quasi tutte le operazioni non bloccano il thread sul quale vengono eseguite, ciò consente di gestire un alto livello di concorrenza utilizzando un piccolo numero di thread di sistema.

Ogni istanza della classe `Vertx` mantiene un certo numero di event loop thread, in assenza di diversa configurazione da parte dell'utente tale numero è uguale al doppio dei nuclei elaborativi messi a disposizione dalla CPU. Ciò consente ad un'applicazione di scalare *verticalmente* all'interno dello stesso processo.

La regola d'oro in Vert.x è non bloccare l'event loop; gni procedura, quando invocata dalla libreria, non dovrà restare in attesa di informazioni o del soddisfacimento di una qualche condizione. Per eseguire codice bloccante è possibile o creare una worker pool personalizzata oppure utilizzare quella fornita dall'istanza della classe `Vertx`.

Le applicazioni che utilizzano Vert.x possono essere sviluppate utilizzando un modello simile quello degli attori [2] nella programmazione concorrente.

Nonostante questo modello sia fornito dalla libreria è del tutto opzionale, infatti un'applicazione potrebbe essere scritta anche senza utilizzare tale modello se lo si desidera.

Un'applicazione può essere costituita da diversi componenti, chiamati *verticles*, che sono eseguiti indipendentemente su un singolo thread. Grazie al fatto che i verticles non condividono alcuno stato interno, o almeno non dovrebbero, tra loro possono essere eseguiti in parallelo consentendo di sbrigare più eventi contemporaneamente.

Esistono tre tipi di verticles:

1. *Standard Verticle*: ogni verticle standard viene assegnato ad un event loop e viene garantito dalla libreria che verrà sempre eseguito su tale. Ciò consente di scrivere codice sincrono senza doversi preoccupare dei problemi tipici dei sistemi concorrenti e lasciare a Vert.x la gestione di tale concorrenza.

2. *Worker verticle*: ogni worker verticle, a differenza di quelli standard, viene eseguito dalla worker thread pool invece che da un event loop. Il loro scopo è quello di eseguire codice bloccante. Possono essere eseguiti da thread differenti in diversi periodi di tempo, ma mai concorrentemente.

3. *Multi-threaded worker verticle*: come i worker verticle possono essere eseguiti da thread differenti ma a loro differenza possono anche essere eseguiti concorrentemente.

La comunicazione tra verticles è fornita attraverso un canale di comunicazione condiviso chiamato *event bus*.

Ogni istanza della classe `Vertx` fornisce accesso a tale canale. Permette la comunicazione tra verticles a prescindere dal linguaggio nel quale sono scritti e dal fatto che facciano parte della stessa istanza della classe `Vertx` o meno.

L'event eus supporta diversi paradigmi per lo scambio di messaggi: peer-to-peer, publish/subscribe e request/response.

Per ricevere messaggi è necessario registrare una procedura ad un indirizzo (una semplice stringa) che verrà poi chiamata quando un messaggio sarà disponibile.

Un messaggio può essere pubblicato ad un indirizzo, in questo caso tutte le procedure registrate ad un indirizzo verranno invocate; oppure inviato, in quest'ultimo caso soltanto una delle procedure registrate verrà invocata seguendo una politca a round-robin. Ad ogni messaggio è inoltre possibile rispondere con un altro messaggio, questa operazione può essere ripetuta per un numero indefinito di volte dando vita ad una comunicazione privata tra due componenti.

Raggruppando assieme più istanze della classe `Vertx` è possibile unificare gli event bus delle singole istanze, creando un singolo event bus distribuito tra le varie istanze che consente a componenti della stessa applicazione (o di applicazioni diverse) di comunicare tra loro.

L'event bus inoltre può essere esteso al web e a client TCP, ciò consente di creare un bus di eventi distribuito che comprende: applicazioni che utilizzando Vert.x, web browser ed altre applicazioni che non utilizzano Vert.x ma che sfruttando questi bridge possono essere integrate nell'ecosistema di Vert.x avendo la possibilità di ricevere ed inviare messaggi.

4.1.2 Perché Vert.x

Le applicazioni web moderne e l'aumento delle vendite nel settore degli smartphone e tablet hanno ridefinito i compiti previsti per un server web. Node.js è stata la prima tecnologia che ha riconosciuto lo spostamento del paradigma e ha offerto una soluzione.

Node.js, utilizzando il motore JavaScript V8 di Google, porta con sé alcuni svantaggi, tra cui il fatto che ogni applicazione può sfruttare al più un singolo nucleo elaborativo per l'esecuzione. Ciò significa che se si ha la necessità di eseguire operazioni bloccanti l'intera applicazione dovà attendere il termine di tale operazione e non potrà servire le altre richieste che nel frattempo verranno effettuate oppure duplicare una o più volte il processo e gestire i diversi processi in esecuzione.

Vert.x prende alcune delle innovazioni di Node.js e le rende disponibili sulla Java Virtual Machine, unendo idee fresche con uno degli ambienti runtime più sofisticati e più rapidi disponibili sul mercato. Vert.x è dotato di una serie di interessanti funzionalità che lo rendono interessante per chiunque sviluppi applicazioni web.

I principali vantaggi che Vert.x porta nell'ambito dello sviluppo di applicazioni web sono:

- La possibilità di utilizzare librerie affermate provenienti dall'intero ecosistema Java.
- Offre il supporto per più linguaggi di programmazione, in un team di sviluppo dove ogni sviluppatore ha competenze in linguaggi diversi consente ad ognuno di essi di sviluppare nel linguaggio nel quale preferiscono al fine di aumentare la produttività
- Senza l'ausilio di librerie esterne consente di:
  - Creare applicazioni web;
  - Servizi RESTful;
  - Effettuare lo streaming di eventi;
  - Possibilità di accesso asincrono a basi di dati (come MySQL e MongoDB);
  - Offre il supporto per web socket;
  - Possibilità di accesso asincrono al file system;
  - E molto altro...
- È in grado di utilizzare tutti i nuclei elaborativi che il sistema mette a disposizione al fine di accelerare la velocità di elaborazione dei dati.
- Fornendo il supporto alle tecnologie per effetturare il clustering, permette di scalare orizzonalmente senza effettuare modifiche nel codice ma semplicemente aggiungendo nuove macchine.

- Essendo costruito al di sopra del modello reattivo [5] gli sviluppatori non devono necessariamente essere a conoscenza di tutti i dettagli della programmazione concorrente e possono concentrarsi sullo sviluppo della logica applicativa.

- Grazie al bus di eventi integrato permette lo scambio di messaggi all'interno della stessa applicazione e tra applicazioni diverse.

- Offre il supporto nativo per un'architettura a microservizi [22].

- Permette di lanciare verticles programmaticamente per eseguire compiti che impiegeranno un lungo periodo di tempo per essere completati in parallelo senza che si ostruiscano a vicenda.

- Non è framework restritivo o un application server (come Apache Tomcat, Wildfly, Glassfish, e così via) e non esiste una maniera corretta per scrivere un'applicazione, Vert.x forisce un insieme di strumenti e funzionalità utili che permettono agli sviluppatori di creare applicazioni a modo loro che possono essere eseguite in maniera indipendente.

4.1.3 Il sistema di moduli

Vert.x offre un sistema di moduli che sfrutta le funzionalità di basso livello che il nucleo della libreria mette a disposizione per facilitare lo sviluppo di applicazioni reattive. Tra i vari moduli possiamo trovare:

- Moduli per la creare moderne applicazioni web e architetture a microservizi.

- Moduli per l'accesso asincrono a basi di dati.

- Moduli per integrare la propria applicazione con diversi servizi già esistenti. Offre client per l'invio di mail, per l'interazione con RabbitMQ[5], e molti altri.

- Moduli per estendere il bus di eventi oltre la Java VM. Offre un bridge per interagire con Vert.x da qualsiasi applicazione dal momento che tale modulo

[5] `https://www.rabbitmq.com/`

è scritto al di sopra del protocollo TCP, un bridge per interagire con Apache Camel[6] e un bridge che, utilizzando SockJS[7], offre la possibilità di estendere il bus di eventi ai web browser.

- Moduli per implementare l'autenticazione e l'autorizzazione all'interno di un applicazione, tra i modli presenti abbiamo anche JWT [18] e OAuth 2 [16].

- E molti altri...

Per lo sviluppo del prototipo sono stati utilizzati due moduli: il modulo per la creazione di applicazioni web (che comprende il modulo per estendere il bus di eventi ai browser) per la creazione dell'interfaccia RESTful e permettere alle applicazioni interessate di restare in ascolto per l'arrivo di eventi dai dispositivi intelligenti; infine il modulo per estendere il bus di eventi su socket TCP, affinché qualsiasi applicazione possa restare in ascolto per l'arrivo di richieste da parte del'interfaccia web e reagire di conseguenza.

4.2 PyBluez

PyBluez è un modulo d'estensione che consente l'accesso alle risorse Bluetooth di sistema.

Tra le funzionalità messe a disposizione abbiamo:

- La possibilità di effettuare la ricerca dei dispositivi Bluetooth vicini.

- La possiibilità di creare socket Bluetooth per creare un server e restare in attesa della connessione da parte di cilent oppure connettersi a un server Bluetooth già presente. Offre le primitive di comunicazione `send` e `recv` per l'invio e la ricezione di messaggi.

[6] `http://camel.apache.org`
[7] `https://github.com/sockjs/sockjs-client`

4.3 TCP Event Bus Bridge Client

Python TCP Event Bus Client è un modulo d'estensione per Python che consente di comunicare con il bus di eventi di Vert.x sfruttanto il TCP Event Bus Bridge descritto in precedenza.

Al momento dell'istanziazione un oggetto della classe `Eventbus` tra i parametri disponibili è possibile fornire il nome dell'host al quale si desidera connettersi e la porta sulla quale deve avvenire la connessione.

Al fine di poter ricevere messaggi provenienti dal bus di eventi di Vert.x è necesssario registrare una procedura che verrà chiamata ad un indirizzo specifico quando un messaggio sarà disponibile, è inoltre possibile annullare la registrazione delle procedure registrate ad un indirizzo specifico.

Un messaggio è inviato verso un indirizzo specifico e può essere inviato in modalità: punto-a-punto oppure in broadcast.

Un messaggio può essere:

- Un tipo di dato primitivo;
- Una sequenza di caratteri;
- Un oggetto JSON;
- Un elenco che contieneuno o più dei tipi di dato elencati in precedenza.

All'invio di un messaggio può essere passato come argomento al metodo chiamato (`send` o `publish`) un oggetto per configurare le opzioni di consegna, a tale oggetto possono essere aggiunte o rimosse intestazioni, possono essere aggiunti o rimossi uno o più indirizzi di risposta e infine può essere impostato l'intervallo di tempo per i messaggi di risposta.

Nel caso il messaggio sia inviato in modalità punto-a-punto (utilizzando il metodo `send`) è possibile specificare anche una procedura da invocare nel caso un messaggio di risposta sarà disponibile.

4.4 Wiring

Wiring è una piattaforma di sviluppo pensata per facilitare il compito a progettisti ed artisti nella realizzazione di progetti interattivi come ad esempio l'accensione di luci o dispositivi all'approssimarsi di una persona al dispositivo sviluppato.

Wiring fornisce un framework open-source cross-platform per scrivere e installare programmi su microcontrollori. Tra le piattaforme supportate abbiamo: AVR Xmega, AVR Tiny, TI MSP430, Microchip PIC24/32, e altre piattaforme.

Utilizza il linguaggio C/C++ per la programmazione dei microcontrollori ed utilizza il compilatore AVR-GCC per tradurre il codice sorgente in linguaggio macchina. Fornisce una libreria personalizzata, chiamata anch'essa Wiring, per accedere all'hardware del microcontrollore. I programmi Wiring sono chiamati *sketch* e necessitano soltanto di due procedure per essere eseguiti:

- `setup()`: viene eseguita una sola volta, all'avvio del programma, può essere utilizzata per definire le impostazioni del programma che non verranno più cambiate nel corso della sua esecuzione (come l'inizializzazione dei componenti necessari);
- `loop()`: contiene le istruzioni che verranno richiamate ciclicamente fino allo spegnimento del dispositivo.

4.5 Analisi delle prestazioni

Al fine di fornire una stima sulle prestazioni del prototipo sviluppato è stato eseguito un test effettuando 10.000 richieste e calcolando, in determinati punti chiave, il tempo richiesto per soddisfare una richiesta.

Il test ha lo scopo di stimare:

- Il tempo richiesto ad un agente esterno per ricevere una risposta ad una richiesta;
- Il tempo richiesto all'interfaccia web per ricevere la risposta da inviare all'agente esterno che ha effettuato la richiesta;
- Il tempo richiesto al gateway per inviare la richiesta al dispositivo intelligente interpellato e ricevere una risposta;

- Il tempo richiesto al dispositivo intelligente interpellato per soddisfare una richiesta.

Componente	Minimo	Massimo	Media
Agente esterno	298	2164	427.754
Interfaccia web	263	636	391.448
Gateway	209	478	337.239
Dispositivo	77	83	80.703

Nota: Le latenze sono state rilevate in millisecondi.

Tabella 4.1: Tempi di risposta minimi, massimi e medi tra i componenti. I tempi di risposta rilevati (in millisecondi) sono: tempo richiesto all'agente esterno per ricevere una risposta ad una richista effettuata; tempo richiesto dall'interfaccia web per ricevere la risposta ad una richiesta inoltrata ai livelli inferiori; tempo richiesto per inoltrare una richiesta al dispositivo interessato e ricevere una risposta; tempo richiesto per il dispositivo per soddifare una richiesta.

Dai test effettuati è emerso che la maggior parte del tempo necessario per soddisfare una richiesta è impiegato nella comunicazione tra il gateway ed il dispositivo intelligente interpellato. A riprova di ciò è sufficiente calcolare la differenza fra le medie delle latenze di comunicazione tra i vari componenti per scoprire che in media:

- 36.3065 ms (circa l'8% del tempo totale) è la latenza complessiva di rete che intercorre tra l'invio della richiesta da parte di un agente esterno e la ricezione della risposta da parte dell'interfaccia web;

- 54.2085 ms (circa il 13% del tempo totale) sono impiegati dal gateway per elaborare la richiesta effettuata dall'agente esterno e la risposta ricevuta dal dispositivo interpellato;

- 256.536 ms (circa il 60% del tempo totale) sono impiegati nella comunicazione tra il gateway ed il dispositivo intelligente;

- 80.7033 ms (circa il 19% del tempo totale) sono impiegati dal dispositivo per rispondere ad una richiesta che gli è stata inoltrata.

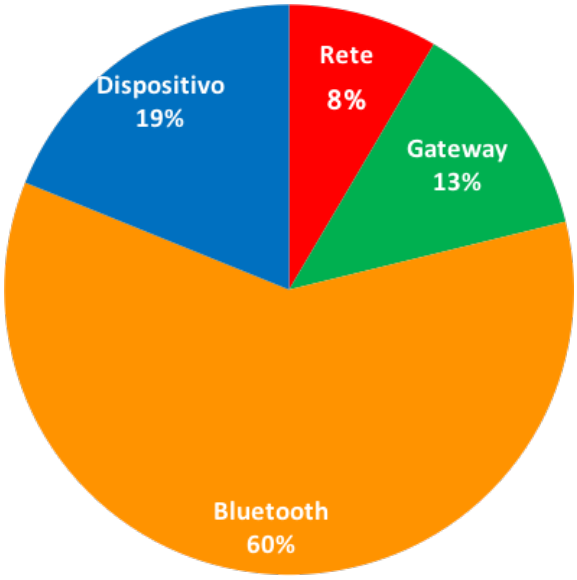

Figura 4.1: Grafico a torta che illustra (in percentuale) la quantità di tempo che ogni componente utilizza, in media, per soddisfare una richiesta.

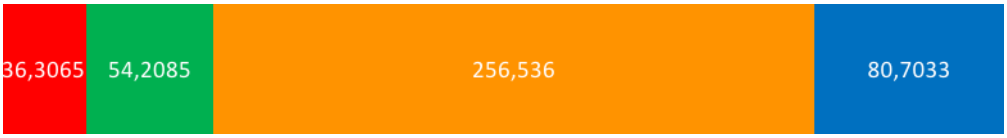

Figura 4.2: Istogramma che illustra (in millisecondi) la quantità di tempo che ogni componente utilizza, in media, per soddisfare una richiesta.

Conclusioni e sviluppi futuri

In questa Tesi si è dapprima discusso del concetto dell'Internet of Things: quali sono le tecnologie abilitanti, i protocolli di rete, i domini applicativi (domotica, robotica, industria biomedicale, monitoraggio in ambito industriale, etc.) e infine i problemi di privacy, sicurezza, autonomia e controllo che ad oggi sono ancora irrisolti. È stato introdotto Web of Things come un livello applicativo di IoT che consente di semplificare l'interazione con oggetti del mondo fisico utilizzando gli standard del Web già esistenti [14, 25, 13, 9] e standard per gestire la sicurezza e l'autorizzazione nelle applicazioni moderne [16, 18]. È stato presentato un prototipo di piattaforma per Web of Things ed anche come essa può essere utilizzata come applicativo per facilitare la realizzazione di applicazioni IoT.

In futuro sarebbe utile fornire il supporto ad altre tecnologie abilitanti per IoT come ZigBee e BLE, dal momento che questi ultimi sono standard di comunicazione specifici tra dispostivi facenti parte dell'IoT. Inoltre sarebbe opportuno rendere l'interfaccia REST conforme al Web Thing Model [28] al fine di consentire un alto livello di operabilità con altre piattaforme per Web of Things.

Infine sarebbe stimolante, integrando la piattaforma per Web of Things sviluppata in questa Tesi con una piattaforma per Web of Augmented Things [8], condurre esperimenti di realtà mista [23] dove oggetti presenti nel mondo fisico coesistono ed interagiscono con oggetti che risiedono nel mondo aumentato, ovvero a fronte di variazioni rilevate nel mondo fisico è possibile osservare una reazione nel mondo aumentato e viceversa.

Ringraziamenti

A conclusione di questo lavoro non mi resta che ringraziare tutti coloro che hanno contribuito alla sua realizzazione.

In primis ci tengo a ringraziare il Prof. Alessandro Ricci e il Dott. Ing. Angelo Croatti per avermi dato l'opportunità di lavorare ad un progetto così interessante e per avermi coinvolto in due presentazioni dove si è mostrata una prima forma di integrazione tra il mondo fisico e la realtà aumentata.

Non possono non ringraziare la mia famiglia, i miei nonni e i miei parenti che, fin da quando ne ho memoria, mi hanno sempre sostenuto e non mi hanno mai fatto mancare nulla.

Per ultimi, ma non meno importanti, voglio rigraziare tutti i miei amici; in particolare Federico, Nicola e Martina con i quali ho condiviso, e continuo a condividere, ricordi indelebili.

Bibliografia

[1] affariitaliani.it. Il marketing virale non esiste, 2010.

[2] Gul A Agha. Actors: A model of concurrent computation in distributed systems. Technical report, MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL INTELLIGENCE LAB, 1985.

[3] Toni M Somers Arik Ragowsky. Enterprise resource planning. *Journal of Management Information Systems*, 19(1):11–15, 2002.

[4] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[5] Jonas Bonér, Dave Farley, Roland Kuhn, and Martin Thompson. The reactive manifesto, 2014.

[6] Dr. Stuart D. Cheshire, Dr. Bernard D. Aboba Ph.D., and Erik Guttman. Dynamic Configuration of IPv4 Link-Local Addresses. RFC 3927, May 2005.

[7] Tein-Yaw Chung, Ibrahim Mashal, Osama Alsaryrah, Van Huy, Wen-Hsing Kuo, and Dharma P Agrawal. Social web of things: A survey. In *Parallel and Distributed Systems (ICPADS), 2013 International Conference on*, pages 570–575. IEEE, 2013.

[8] Angelo Croatti and Alessandro Ricci. Towards the web of augmented things. In *Software Architecture Workshops (ICSAW), 2017 IEEE International Conference on*, pages 80–87. IEEE, 2017.

[9] Douglas Crockford. The application/json media type for javascript object notation (json). 2006.

[10] Chris J Date and Hugh Darwen. *A Guide to the SQL Standard*, volume 3. Addison-Wesley New York, 1987.

[11] Pavan Deolasee, Amol Katkar, Ankur Panchbudhe, Krithi Ramamritham, and Prashant Shenoy. Adaptive push-pull: disseminating dynamic web data. In *Proceedings of the 10th international conference on World Wide Web*, pages 265–274. ACM, 2001.

[12] Thomas Erl. *Service-oriented architecture: concepts, technology, and design*. Pearson Education India, 2005.

[13] Ian Fette. The websocket protocol. 2011.

[14] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Hypertext transfer protocol–http/1.1. Technical report, 1999.

[15] Dominique Guinard, Vlad Trifa, and Erik Wilde. A resource oriented architecture for the web of things. In *Internet of Things (IOT), 2010*, pages 1–8. IEEE, 2010.

[16] Dick Hardt. The oauth 2.0 authorization framework. 2012.

[17] CISCO ISBG. The internet of things: How the next evolution of the internet is changing everything, 2011.

[18] Michael Jones, John Bradley, and Nat Sakimura. Json web token (jwt). Technical report, 2015.

[19] Ari Juels. Rfid security and privacy: A research survey. *IEEE journal on selected areas in communications*, 24(2):381–394, 2006.

[20] Jacek Kopeckỳ, Karthik Gomadam, and Tomas Vitvar. hrests: an html microformat for describing restful web services. In *Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT'08. IEEE/WIC/ACM International Conference on*, volume 1, pages 619–625. IEEE, 2008.

[21] Vineet Kumar. *Customer relationship management.* Wiley Online Library, 2010.

[22] Dmitry Namiot and Manfred Sneps-Sneppe. On micro-services architecture. *International Journal of Open Information Technologies*, 2(9):24–27, 2014.

[23] Yuichi Ohta and Hideyuki Tamura. *Mixed reality: merging real and virtual worlds.* Springer Publishing Company, Incorporated, 2014.

[24] Victoria Pimentel and Bradford G Nickerson. Communicating and displaying real-time data with websocket. *IEEE Internet Computing*, 16(4):45–53, 2012.

[25] Leonard Richardson and Sam Ruby. *RESTful web services.* " O'Reilly Media, Inc.", 2008.

[26] TechRadar. How bluetooth smart is shaping the internet of things, 2014.

[27] Vlad Trifa. Towards the wot manifesto, 2009.

[28] Vlad Trifa, Dominique Guinard, and David Carrera. Web thing model. 2015.

[29] WhatIs.com. What is embedded system?, 2014.

[30] Microformats Wiki. What are microformats?, 2016.

[31] Tatu Ylonen and Chris Lonvick. The secure shell (ssh) protocol architecture. 2006.