Alma Mater Studiorum · Università di Bologna

**SCUOLA DI SCIENZE**
**Corso di Laurea in Scienze di Internet**

# Vending Machines avanzate basate su principi di Physical Web e Internet of Things

**Relatore:**
**Chiar.mo Prof.**
**Luciano Bononi**

**Presentata da:**
**Luca Comellini**

**Sessione II**
**Anno Accademico 2014/2015**

*a Giulia*

# Indice

# Elenco delle figure

# Listings

# Capitolo 1

# Introduzione

Negli ultimi anni l'interesse verso l'*Internet of Things* è cresciuto costantemente ed è previsto che questo fenomeno avrà una boom esponenziale nel prossimi anni, con addirittura 17 mila miliardi di dispositivi connessi previsti per il 2020. Per Internet of Things si intende un'evoluzione dell'uso di Internet, in cui gli oggetti si rendono riconoscibili e acquisiscono "intelligenza" grazie alla possibilità di comunicare informazioni su loro stessi o di accedere a quelle messe a disposizione da altri. La disponibilità di connessioni a Internet è sempre maggiore e a basso costo e con i prossimi sviluppi legati principalmente all'*IPv6* sarà possibile connettere alla rete praticamente qualsiasi oggetto. Questo sviluppo porterà innumerevoli benefici, di cui si tratterà ampiamente all'interno di questo elaborato. Come ciascuna innovazione tecnologica, tuttavia, i benefici si accompagneranno a nuove problematiche:

- **Scoperta**
  Una delle sfide principali è quella di venire a conoscenza degli oggetti *smart*, connessi, attorno a noi, che in qualche modo devono pubblicizzare la loro presenza per essere scoperti e utilizzati.

- **Interazione**
  La molteplicità di oggetti connessi a Internet renderà sempre più onerosa l'installazione e l'utilizzo di un'applicazione specifica per ognuno di essi. Per esempio un'applicazione diversa per ogni produttore di lampadine, serrature, robot per la pulizia dei pavimenti o termostati *smart*. Inoltre, il

download da uno *store* e l'installazione di un'applicazione, per un singolo oggetto che si utilizzerà probabilmente una sola volta, risulta essere uno scoglio gravoso per molti utenti, viste anche le dimensioni che le applicazioni raggiungono al giorno d'oggi, sia per la quantità di dati richiesti per il download, sia in termini di memoria occupata una volta installate.

In una prima fase è quindi stato analizzato e implementato un sistema che permetta a chiunque, in qualsiasi momento, di scoprire e interagire con un oggetto senza dover intraprendere lunghi processi, sfruttando gli strumenti offerti dal progetto *Physcal Web*, che si pone come obbiettivo quello di risolvere le problematiche appena descritte. Questo progetto, guidato da Google, è open source e non brandizzato, e viene proposto come standard, nella speranza che venga adottato dai diversi produttori di software e hardware. Da una parte è presente un trasmettitore Bluetooth, che invia continuamente un *beacon*, seguendo le specifiche del protocollo Eddystone-URL, per segnalare la propria presenza al mondo circostante, continuamente e indipendentemente dalla presenza di un soggetto ricevente. Dall'altra parte ci sarà, a seconda della piattaforma ricevente, un browser o uno strumento di ricezione all'interno del sistema operativo. L'intento finale del progetto Physical Web è quello di creare uno standard, in modo che la *scoperta* possa essere integrata come funzionalità all'interno dei diversi sistemi. Essendo il contenuto trasmesso da questi beacon degli URL, indipendentemente da come è stato implementato, una volta passata la fase di ricezione, verrà utilizzato un qualsiasi browser per aprire la pagina dell'oggetto e interagire con esso.

In una seconda fase, sono state individuate tecnologie alternative, più mature, in modo da essere utilizzate fin da subito senza imprevisti. Infatti, essendo le tecnologie legate al Physical Web ancora agli albori, le funzionalità al momento disponibili sono dipendenti dalle implementazioni effettuate dai vari browser e sistemi operativi, e comunque non ancora complete o alla pari di quelle presenti in applicazioni native. È stato quindi fatto un confronto con una applicazione nativa per Android realizzata ad hoc, che sfruttando gli strumenti resi disponibili dal sistema, individua dei beacon Eddystone-UID, leggendone e interpretandone i dati ad esso associati. A questi è infatti possibile legare dei dati a piacere, che verranno resi disponibili all'applicazione. Essendo il dato principale un URL,

lo stesso che viene utilizzato per l'Eddystone-URL, questo non va a stravolgere la struttura del progetto; l'applicazione è inoltre in grado di intercettare anche le richieste di visualizzazione della pagina effettuate dall'eventuale servizio di individuazione degli URL, nel caso fosse già implementato all'interno del sistema, e di mostrare correttamente la lista dei prodotti ad esso associati.

In questo elaborato si propone un'applicazione pratica delle tecnologie appena descritte alle vending machines, ovvero i distributori automatici di prodotti (cibi, bevande, biglietti ferroviari, prodotti di qualsiasi genere) posti all'interno di luoghi pubblici. In particolare, è stata sviluppata una piattaforma che consente agli utenti di individuare la presenza di distributori automatici attraverso la ricezione di un segnale trasmesso da un beacon Bluetooth. Una volta ricevuto quest'ultimo, all'interno dell'applicazione web o Android, l'utente può interagire con il distributore tramite il proprio dispositivo mobile personale, con il quale si rapporta quotidianamente, trovando quindi un'interfaccia semplice e familiare. L'interazione potrà essere costituita dalla consultazione dei prodotti disponibili e delle informazioni associate, l'acquisto degli stessi e la fruizione di offerte speciali.

# Capitolo 2

# Internet of Things

## 2.1 Stato dell'arte

Sempre più spesso il mondo connesso include oggetti fisici. Macchinari, spedizioni, infrastrutture e dispositivi sono dotati di reti di sensori e attuatori che consentono loro di monitorare l'ambiente circostante, comunicare il proprio stato, ricevere istruzioni, e anche compiere azioni sulla base delle informazioni ricevute. Per esempio, persino le persone possono essere dotate di dispositivi equipaggiati con sensori per monitorare il loro stato di salute. Questo è ciò che si intende con il termine "Internet of Things", abbreviato convenzionalmente in "IoT". Miliardi di dispositivi in tutto il mondo sono attualmente connessi a Internet, compresi computer e smartphone. È stimato che entro la fine del 2015 questo numero raggiungerà i 15 miliardi, con un'ulteriore notevolmente aumento previsto per il prossimo decennio, con stime che vanno da 50 miliardi di dispositivi fino a mille miliardi.[12]

Portando macchinari e beni quali container o letti ospedalieri nel mondo connesso, l'Internet of Things consente nuove modalità di monitoraggio e la gestione di tutte le "parti in movimento" che compongono un business. In qualsiasi momento, il reparto di gestione può vedere lo stato e il flusso di beni o materiali all'interno di una fabbrica, i centri di distribuzione o addirittura sugli scaffali dei negozi. Monitorando i macchinari in tempo reale, le aziende sono in grado di controllare meglio il flusso di merci all'interno delle fabbriche ed evitare

interruzioni, prendendo provvedimenti immediati o attuando una manutenzione preventiva quando sorgono problemi. Oggetti con attuatori integrati in aggiunta ai sensori possono essere programmati per agire per conto proprio. L'adozione diffusa dell'Internet of Things richiederà tempo, ma questo lasso sarà sempre minore, grazie al continuo miglioramento delle tecnologie alla base, come i sensori miniaturizzati e le reti wireless.

Alcuni degli impieghi più promettenti sono nei settori della sanità, delle infrastrutture e della pubblica amministrazione, per aiutare la società ad affrontare alcune delle sue sfide più grandi. Il monitoraggio remoto, per esempio, ha il potenziale per fare una grande differenza nella vita delle persone affette da malattie croniche e contemporaneamente di ridurre una fonte significativa di costi sanitari. La capacità di monitorare e controllare reti elettriche e idriche potrebbe avere importanti ripercussioni per il risparmio di energia, le emissioni di gas serra, e le perdite d'acqua. Utilizzando sensori per raccogliere informazioni e ottimizzare le operazioni, le funzioni pubbliche come la raccolta dei rifiuti potranno diventare molto più efficienti e rapide.

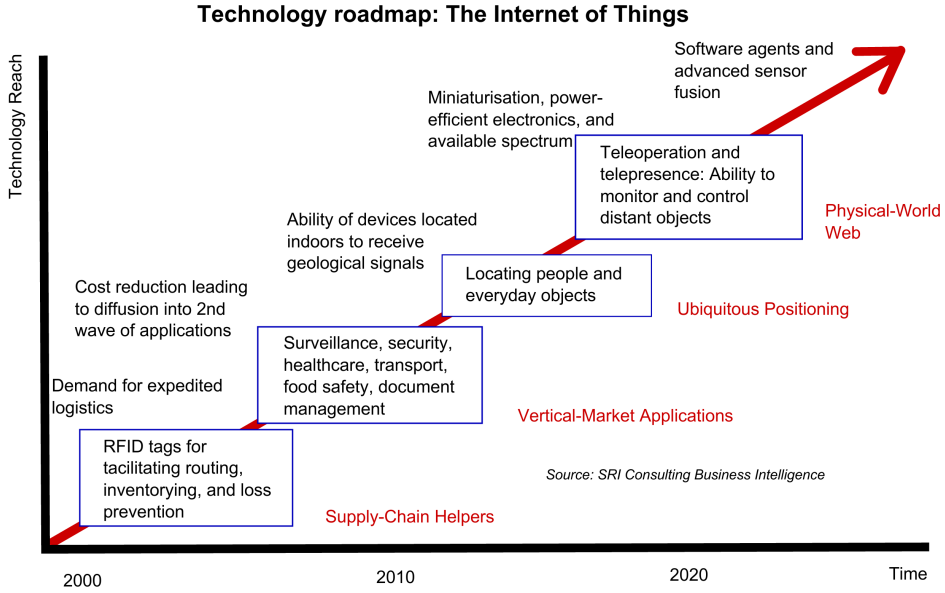



Figura 2.1: Evoluzione dell'Internet of Things

Realizzare il pieno potenziale dell'Internet of Things non è però facile. Per sfruttare l'elevato valore potenziale di queste applicazioni, le aziende dovranno

disporre di sistemi in grado di dare un senso all'enorme quantità di dati che questi sensori remoti possono fornire. Ad esempio, con un uso più diffuso degli *RFID*[1], alcune aziende potrebbero monitorare centinaia di migliaia, o forse addirittura milioni, di oggetti in tempo reale, richiedendo notevoli capacità analitiche.

Unire il mondo fisico e digitale, comporta anche importanti implicazioni per la privacy, la sicurezza e persino sull'organizzazione delle aziende stesse. Come qualsiasi altra connessione dati, le connessioni che permettono a macchine remote di agire senza un operatore umano sono soggette all'attacco da parte di hacker e criminali. I dati raccolti tramite il monitoraggio della salute potrebbero essere oggetto di abusi. Gli oggetti che gestiscono la *smart grid* di una casa (che per esempio possono accendere e spegnere selettivamente l'impianto di climatizzazione o gli elettrodomestici, per risparmiare energia o approfittare di costi dell'elettricità minori) fanno aumentare le preoccupazioni sulla privacy. Questi problemi dovranno essere affrontati preventivamente prima che la società e le imprese siano in grado di godere di tutti i vantaggi dell'IoT.[12]

La definizione "Internet of Things" non ha origini recenti; alcuni professori del Massachusetts Institute of Technology, infatti, hanno iniziato a parlare quasi una ventina di anni fa di un mondo dove le cose (oggetti e sensori) erano connesse e in grado di condividere dati. I dati provenienti da questi dispositivi forniscono informazioni sul business che prima era impossibile ottenere. Il valore inestimabile dei dati ricevuti da questi dispositivi connessi è quello che rende l'Internet of Things così importante. Avendo accesso a questi dati le aziende possono ottimizzare i loro processi di business, individuare nuove opportunità, avere elementi necessari per prendere una decisione informata, capire e prevedere il comportamento dei clienti e partner, in modi che prima non erano immaginabili. Attualmente l'Internet of Things è a un punto di svolta. Sebbene gli esperti di tecnologia abbiano anticipato per anni un mondo con miliardi di dispositivi intelligenti e connessi, diversi fattori sono confluiti recentemente facendo accelerare l'adozione dell'IoT in diversi scenari di business.

Fra i fattori che stanno spingendo l'adozione dell'IoT possiamo trovare:

[1] Radio-Frequency IDentification ovvero identificazione a radio frequenza

- **Calo del costo dell'hardware**:
  Il costo dei componenti dell'Internet of Things come microchip, sensori GPS, di temperatura e accelerometri è crollato grazie al volume delle richieste. Inoltre l'avanzamento tecnologico ha permesso di avere chip sempre più piccoli ma al contempo più potenti e quindi in grado di eseguire software più complessi e avanzati.

- **Aumento delle macchine che comunicano tra di loro**:
  Le comunicazioni Machine-to-Machine (M2M), sono in forte aumento ed è previsto che entro il 2020 il 50% delle aziende ne avrà adottato una qualche forma.[7] M2M significa che un oggetto, ad esempio un macchinario, è in grado di comunicare autonomamente un'informazione, come l'accadimento di un evento, a un'altra macchina che è in grado di analizzare questi dati e dargli un senso.

- **Aumento delle connessioni**:
  In passato le soluzioni IoT erano limitate solo a connessioni cablate o wireless all'interno di una rete locale, ma alimentati dalla capacità che le reti cellulari avanzate possono fornire, anche gli operatori mobili stanno iniziando ad adottare l'IoT.

- **Le soluzioni cloud offrono costi minori, scalabilità e flessibilità**:
  Con la crescita di servizi come Microsoft Azure, Amazon Web Service e Google Compute/App Engine, il cloud storage e la potenza di elaborazione sono sempre più accessibili e disponibili, ampliando così la capacità di analizzare grandi quantità di dati. Scenari di Internet of Things che incorporano archiviazione nel cloud, analisi e altri strumenti, permettono di avere ulteriori vantaggi di scalabilità e flessibilità di cui le imprese hanno bisogno quando viene iniziata o espansa una soluzione IoT.

- **I benefici economici potenziali sono in costante aumento**:
  Secondo il McKinsey Global Institute, l'Internet of Things ha il potenziale per creare un impatto economico compreso tra i 2.7 e i 6.2 mila milardi di dollari.[12]

L'Internet of Things si riferisce all'uso di sensori, attuatori e tecnologie di comunicazione dati costruiti all'interno di oggetti fisici, dalle strade alle case fino ai pacemaker, che danno l'opportunità di tracciare, coordinare o controllare questi oggetti attraverso una rete dati o Internet. Sono presenti tre step nell'applicazione dell'Internet of Things: cattura dei dati dall'oggetto (per esempio informazioni sulla posizione o anche più complesse), aggregazione delle informazioni attraverso una connessione e interpretazione di quelle informazioni, sia valutando la necessità di un'azione immediata, sia raccogliendo i dati nel tempo per progettare un miglioramento del processo. L'Internet of Things può essere sfruttato per creare valore aggiuntivo in molteplici modi. In aggiunta all'incremento della produttività nelle operazioni correnti, può creare nuovi tipi di prodotti e servizi e nuove strategie: i sensori remoti per esempio rendono possibili servizi *pay-as-you-go* come quelli delle auto a noleggio come Enjoy o Car2Go, che hanno fatto il loro ingresso nel mercato italiano negli ultimi anni. Le tecnologie dell'Internet of Things spaziano dal semplice tag identificativo a complessi sensori e attuatori. I tag RFID possono essere associati potenzialmente a qualsiasi oggetto. Sofisticati dispositivi multisensore che comunicano informazioni riguardo la posizione, le performance, l'ambiente e le condizioni stanno diventando sempre più comuni. Con nuove tecnologie come i Micro Electro-Mechanical Systems[1] (MEMS) sta diventando possibile inserire sensori altamente sofisticati in virtualmente qualsiasi oggetto. In virtù del fatto che questi sono prodotti utilizzando processi simili a quelli dei semiconduttori, il loro prezzo sta calando rapidamente. Con l'aumento di disponibilità di tecnologie sempre più sofisticate, le aziende non solo possono tenere traccia del flusso dei prodotti o dei beni fisici, ma possono anche gestire le performance delle singole macchine e sistemi. Questi sensori possono anche essere integrati nelle infrastrutture, per esempio sensori magnetici nelle strade che permettono di contare il numero dei veicoli che stanno passando, consentendo aggiustamenti della viabilità in tempo reale modificando il tempo dei semafori. Altrettanto importante quanto i sensori e gli attuatori, sono i collegamenti dati che trasmettono queste informazioni e i programmi che li analizzano e danno loro

[1]Sistemi "intelligenti" che abbinano funzioni elettroniche, di gestione dei fluidi, ottiche, biologiche, chimiche e meccaniche in uno spazio ridottissimo, integrando la tecnologia dei sensori, degli attuatori e le più diverse funzioni di gestione dei processi

un senso. Sempre più spesso le applicazioni dell'Internet of Things includono impostazioni nelle quali possono essere scatenate azioni automaticamente sulla base dei dati ricevuti dai sensori. Per esempio una fabbrica può reagire automaticamente in base al flusso di materiale ricevuto, oppure è possibile far scattare il verde in un semaforo solo quando il sensore nell'asfalto sente che ci sono delle vetture in attesa o ancora avvisare un medico quando il battito cardiaco di un paziente con un dispositivo di monitoraggio, assume valori fuori soglia.[12]

Utilizzi base dell'Internet of Things sono già ad un buon livello di maturità. Una delle principali applicazioni già in uso prevede l'impiego di tag RFID per tracciare il flusso di materia prima, componenti e beni attraverso i processi di produzione e distribuzione. Questi tag emettono un segnale radio che può essere utilizzato per individuare precisamente la loro posizione, quindi per esempio, mentre un prodotto si muove all'interno di una fabbrica, dei computer possono tracciare esattamente dove si trova in ogni istante. Utilizzando questa informazione le aziende possono individuare colli di bottiglia e agire di conseguenza, gestendo i tempi di introduzione di ulteriori oggetti nel sistema o programmando l'arrivo dei camion per il ritiro delle merci già pronte. I tag RFID nei container o nelle scatole sono utilizzati per tracciare i prodotti durante il loro tragitto all'interno dei magazzini, degli hub di distribuzione e fino agli scaffali e in alcuni casi fino all'arrivo al cliente. Tracciare questi flussi permette alle aziende di ottimizzare i processi di approvvigionamento ed evitare che le disponibilità siano scarse o che i magazzini si carichino eccessivamente. I tag RFID sono utilizzati anche in sistemi di pagamento automatico dei pedaggi come il Telepass, che consentono di velocizzare il flusso nelle strade a pedaggio. In un altro esempio, FedEx offre un programma che permette ai consumatori di tracciare il progresso della spedizione quasi costantemente, inserendo un piccolo dispositivo (grande come un cellulare) all'interno del pacco. Questo dispositivo contiene un GPS e sensori per monitorare la temperatura, umidità, pressione e esposizione alla luce che possono essere critici per spedizioni come quelle di campioni biologici o di dispositivi elettronici sensibili. Questi dispositivi sono programmati per inviare la posizione e le condizioni atmosferiche periodicamente di modo che il cliente possa conoscere esattamente dove si trova e le sue condizioni e quindi sapere immediatamente se è andato fuori rotta o è stato esposto a condizioni rischiose. Questo tipo di

disponibilità dei dati continua ha forti ed importanti implicazioni soprattutto per le aziende che operano con lunghe e complesse catene di distribuzione.[12]

Nei paragrafi successivi si analizzano diverse soluzioni attualmente presenti sul mercato.

## 2.2 Microsoft

Microsoft è interessata al mercato offerto dall'Internet of Things e ha creato recentemente delle soluzioni specifiche, alcune già disponibili come Windows IoT Core e altre in fase di test come Azure IoT.

### 2.2.1 Microsoft Azure IoT

Microsoft Azure è la famosa piattaforma di cloud computing creata da Microsoft che offre soluzioni *PaaS*[1] e *IaaS*[2] con diversi servizi a cui è stato aggiunto quello per l'Internet of Things. In questo modo è possibile monitorare beni per migliorarne l'efficienza, analizzare ed elaborare la mole di dati ricevuta. All'interno di Microsoft Azure è stata creata quindi una sezione specializzata per l'IoT. È possibile trovare:

- **Event Hub**: un centro di raccolta per tutti gli eventi scatenati da sensori o oggetti, che promette di poter gestire milioni di device quasi in tempo reale.
- **Stream Analytics**: permette di creare dashboard e avvisi per analizzare i dati ricevuti.
- **Machine Learning**: uno strumento per poter fare analisi e manutenzione predittiva.
- **Notification Hubs**: un centro per le notifiche con uno strumento per gestirle in modo scalabile e veloce.

[1] Platform as a Service
[2] Infrastructure as a Service

È stato inoltre creato un progetto open source chiamato "ConnectTheDots.io" per facilitare la connessione dei piccoli dispositivi al cloud di Microsoft Azure e appunto sfruttare questi strumenti. In questo progetto sono presenti esempi di codice, script di configurazione e guide per aiutare gli sviluppatori a impostare i device e i sensori e a connetterli con il cloud per poter analizzare le informazioni ricevute.

### 2.2.2 Windows 10 IoT

Microsoft ha annunciato Windows 10 IoT, che potrà essere installato su diverse tipologie di dispositivi, spaziando dai piccoli device come i gateway, passando per dispositivi mobili fino a grossi macchinari aziendali o attrezzature mediche. Windows 10 IoT offre la possibilità di scrivere applicazioni che possono essere eseguite su qualsiasi piattaforma e meccanismi per garantire connessioni in sicurezza con i suoi strumenti cloud di Azure IoT. Inoltre supporta nativamente meccanismi di comunicazione M2M. Nella Figura 2.2 è possibile vedere le differenze e i requisiti minimi delle varie versioni di Windows 10 IoT e i servizi offerti. Essendo Win-

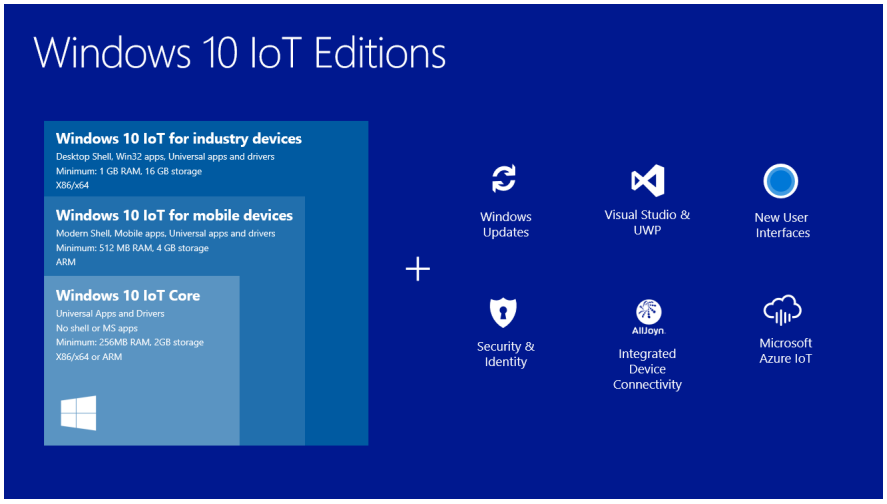



Figura 2.2: Windows 10 IoT Family[1]

dows 10 stato annunciato recentemente, delle tre diverse versioni, attualmente è

[1]Fonte http://www.smashingrobotics.com/microsoft-announces-windows-10-iot-core-public-release/

disponibile soltanto Windows 10 IoT Core che era stata rilasciata anticipatamente come *preview* per permettere agli sviluppatori di testare la piattaforma.

Windows 10 IoT Core è una nuova versione di Windows mirata a piccoli device, soprattutto *embedded*, che non necessariamente possiedono uno schermo. Per i dispositivi embedded con uno schermo non è comunque possibile utilizzare *Windows Shell*, l'interfaccia grafica di Windows, ma è possibile sviluppare una *Universal Windows app*, decidendone l'interfaccia e le funzionalità. Una Universal Windows app è costruita sopra la *Universal Windows Platform*, una piattaforma comune per qualsiasi dispositivo con Windows 10, che permette di avere il *layer* di API principale a disposizione qualunque sia la versione di Windows 10 utilizzata sul dispositivo. Sono disponibili anche API specializzate per quelle che Windows definisce "famiglie" di dispositivi e una di queste è la *IoT family*. Ad esempio un'applicazione che ha come target questa famiglia dispone di un set di API aggiuntive specifiche e può assumere che queste siano sempre presenti. Con Windows 10 non è più possibile specificare una particolare versione del sistema operativo, ma è appunto possibile decidere la famiglia di dispositivi che si vuole andare a raggiungere.[13] Nella Figura 2.3 è possibile vedere come sono organizzate le API nella Universal Windows Platform.

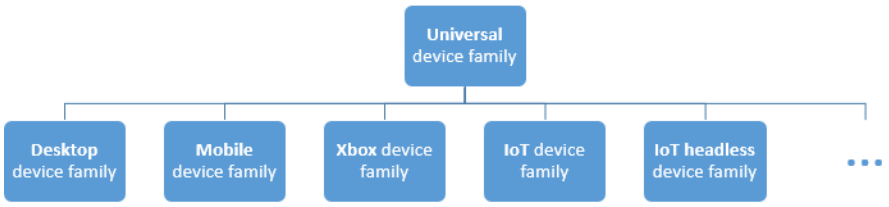



Figura 2.3: Universal Windows Platform[13]

IoT Core è disegnato per avere delle basse barriere d'entrata e per la facilità di costruire applicazioni. Su questa piattaforma è possibile utilizzare diversi linguaggi tra cui: C++, C#, JavaScript, Node.js e Python.
Windows 10 IoT Core è al momento disponibile solo per due piattaforme nativamente: Raspberry Pi 2 e MinnowBoard MAX. Sono presenti però altri strumenti come le librerie Windows Remote Arduino e Windows Virtual Shields for Arduino. La prima libreria rende possibile la comunicazione con un Arduino attraverso

una connessione USB o Bluetooth. Con la seconda libreria è possibile invece scrivere uno Sketch Arduino per Arduino UNO e comunicare con una Universal Window Application eseguibile su qualsiasi dispositivo Windows inclusi i telefoni Windows Lumia. Per esempio è possibile sfruttare praticamente tutti i sensori all'interno di un Windows Phone, in entrambe le direzioni. Questo vuol dire che è possibile sia rilevare dati dall'Arduino per far compiere azioni al telefono, sia utilizzare i sensori di questo per esempio per scatenare azioni nell'Arduino. Per poter sviluppare applicazioni per Windows 10 IoT Core è necessario avere un computer con Windows 10 e Visual Studio 2015.

## 2.3 Ubuntu Core per Internet of Things

Ubuntu è una distribuzione Linux basata su Debian, sviluppata da Canonical nel 2004 che ha visto una grande adozione sia in ambito server che desktop, con numeri in costante aumento.[17]

Nel mese di dicembre 2014, Canonical ha annunciato "Snappy Ubuntu Core", una versione minimale di Ubuntu Server con supporto ai *container* e aggiornamenti transazionali. Snappy è un'evoluzione di Ubuntu Server *Just Enough Operating System* (JeOS), progettato per funzionare in ambienti virtualizzati. Con un *footprint* di soli 380 MB, un kernel specializzato per ambienti server e senza interfaccia grafica, JeOS era perfetto per l'esecuzione di Linux su macchine virtuali; le dinamiche del mercato server e il loro indirizzamento verso le architetture a container hanno però spinto Canonical ad annunciare Snappy come il nuovo sistema operativo Linux minimale basato su Ubuntu, per gli oggetti dell'Internet of Things e per sistemi cloud che li governano.

### 2.3.1 Container

Le applicazioni con una struttura a container incapsulano il runtime, il framework, le librerie e il codice, eliminando la dipendenza dal sistema operativo sottostante. Un host che esegue questi container, richiede un sistema operativo minimale in quanto non interagirà mai con gli strumenti esposti da esso, utilizzando solo l'ambiente di runtime. Questo approccio contrasta con l'attuale modello di di-

stribuzione di Linux, dove il sistema operativo viene fornito con molti strumenti che le applicazioni a container non potrebbero mai utilizzare. Molte distribuzioni si stanno adattando a questo approccio minimalistico, trainate da CoreOS[1] una delle prime distribuzioni ad adottare queste soluzioni e anche Red Hat che ha dato vita a un'iniziativa chiamata "Project Atomic" che ottimizza Red Hat Enterprise, Fedora e CentOS per l'uso dei container.[2]

Anche l'architettura dell'Internet of Things richiede un sistema operativo minimale per gli oggetti che agiscono come *gateway* per l'IoT e che solitamente hanno risorse limitate. Queste distribuzioni di Linux "snellite", giocano quindi un ruolo fondamentale nel connettere i vari sensori al cloud e Ubuntu Core ne è un valido esempio. All'interno di Ubuntu Core, le applicazioni e i dati associati, sono completamente isolate le une dagli altri per evitare problemi di compatibilità e sicurezza. Sia le applicazioni che il sistema operativo vengono aggiornati in modo atomico, cioè viene scaricata una nuova "immagine" e tentata l'installazione, ma se questa operazione non va a buon fine si ritorna automaticamente allo stato precedente del sistema (*rollback*). Per gli aggiornamenti del sistema operativo una volta installati, il sistema tenta un riavvio: se questo va a buon fine la nuova immagine, viene etichettata come buona e diventa quella di default; se il processo fallisce, torna automaticamente all'immagine precedente. Questo sistema è chiamato "transazionale" o "basato su immagini". Questo permette di avere un sistema con interruzioni o downtime ridotti al minimo, ideale per ambienti dove la prevedibilità e l'affidabilità sono di importanza primaria.

### 2.3.2 Architettura di Snappy

Snappy è diversa da una tradizionale distribuzione Linux basata sui pacchetti, come Ubuntu per desktop o server. Ogni parte del sistema è isolata in file di sola lettura e lo stesso avviene per ogni singola applicazione. Gli sviluppatori possono quindi aggiornare le loro applicazioni senza preoccuparsi di creare problemi alle altre installate. Anche se Ubuntu Core è alla base di Snappy, questa non include il

[1] https://coreos.com/
[2] http://www.projectatomic.io/

tradizionale gestore dei pacchetti *apt-get* presente nelle varie distribuzioni basate su Debian, ma un tool chiamato *snappy*.

Snappy è ideato e ottimizzato per eseguire container *Docker* con il suo motore e strumenti a linea di comando pre-installati. Qualsiasi applicazione che può essere eseguita in Docker può essere facilmente essere portata su Snappy. Quest'ultimo è per sua natura modulare ed espandibile, architettato per gestire efficientemente le operazioni delle applicazioni e del sistema operativo sottostante.[19]
Come è possibile vedere in Figura 2.4, sono presenti quattro strati che compongono l'infrastruttura:

- **Hardware**: la parte fisica del dispositivo, chiamata *Enablement*, fornita da un produttore di hardware o da Canonical.
- **Sistema**: il sistema operativo, Ubuntu Core.
- **Framework**: servizi di terze parti creati dai venditori in collaborazione con Canonical per fornire funzionalità specifiche, librerie e fare da mediazione per le risorse condivise, come sensori e documenti. Al contrario delle applicazioni, questi hanno dei permessi speciali che permettono loro di interagire con il sistema. Gli sviluppatori scrivono le applicazioni avendo come target un determinato framework, ad esempio Docker. I framework sono indipendenti, non possono dipendere su altri framework e sono mantenuti autonomamente senza impattare il sistema.
- **Applicazioni**: le applicazioni che vengono eseguite in Snappy sono chiamate *Snapps* e possono essere eseguite solo all'interno di un framework. Sono distribuite sotto forma di pacchetti e ognuna ha la sua directory isolata sul filesystem.[19]

### Docker

Docker permette di "impacchettare" un'applicazione con tutte le sue dipendenze in un'unità standard per lo sviluppo di software. Un container Docker incapsula il software in un filesystem completo che ha tutto il necessario per essere eseguito:

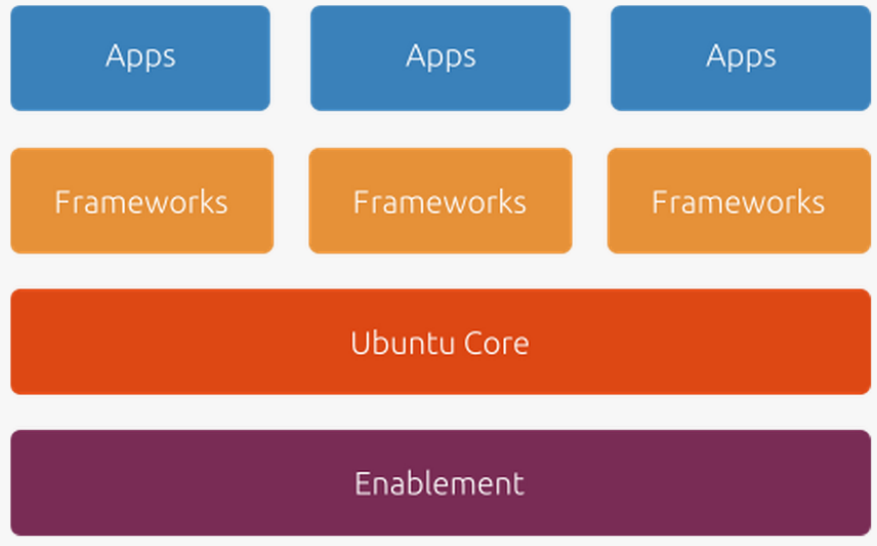



Figura 2.4: Architettura di Snappy Ubuntu Core[19]

codice, runtime, librerie di sistema, strumenti di sistema, qualsiasi cosa sia installabile in un server. Questo garantisce che verrà sempre eseguita allo stesso modo, indipendentemente dall'ambiente in cui si trova. I container condividono il kernel tra di loro, ma eseguono un processo isolato in *userspace* nel sistema operativo ospite. Non essendo legati a una specifica architettura possono essere eseguiti su qualsiasi computer, infrastruttura o cloud.[2] Docker è diventato velocemente uno standard *de facto* per i container e recentemente ha deciso di collaborare con altre aziende per creare un vero standard per i container software, chiamato Open Container Project[1]. In questo sforzo per creare uno standard si sono unite numerose aziende, tra cui: Amazon, Google, Microsoft, Oracle, VMware, IBM, Cicso e HP.

Snappy Ubuntu Core è disponibile su varie piattaforme e dispositivi: sono disponibili immagini per Virtual Machine su piattaforme cloud come Amazon EC2, Microsoft Azure e Google Compute Engine; tra i dispositivi supportati invece si trovano per esempio Raspberry Pi 2, Beaglebone Black e Odroid-C1. Canonical prevede di creare un *marketplace* dove i fornitori di framework e gli sviluppatori di applicazioni possono pubblicare i loro prodotti. Un tipico flusso per lanciare un'applicazione su Snappy include l'avvio di un'istanza di Snappy

[1]http://www.opencontainers.org/

Ubuntu Core, l'installazione di un framework pubblicato sul marketplace e infine la distribuzione di Snapps. L'azienda ha come obiettivo quello di creare un marketplace universale con Snapps che possono essere mirate per il *deployment* di applicazioni mobili, IoT e cloud.[19]

Linux ha visto una crescita enorme negli ultimi due decenni e grazie agli sforzi di vendor come Red Hat e Canonical, sta diventando un sistema operativo valido per eseguire e supportare carichi di lavoro aziendali. Linux è al momento il sistema operativo preferito per il cloud: supportando quest'ultimo e molteplici device, Canonical ha ideato Snappy per poter fornire un sistema semplice, simile a quanto si ha oggi per l'installazione di applicazione nel mondo mobile e che ha il potenziale per diventare uno dei punti principali dell'aumento e sviluppo dell'Internet of Things.[1] È riuscita persino a collaborare con aziende come Microsoft e Amazon per il rilascio delle loro API mirate all'IoT e di produttori di hardware come Intel nel progetto *Intel IoT Gateway*.[20][21]

## 2.4 Intel IoT Platform

Intel IoT Platform è sia un modello di riferimento *end-to-end* sia una famiglia di prodotti di Intel che grazie anche all'integrazione con soluzioni di terze parti fornisce strumenti fondamentali per connettere, senza interruzioni e in modo sicuro, i dispositivi a Internet o al cloud attraverso una connessione sicura e creando inoltre valore aggiunto attraverso l'analisi dei dati. Intel inoltre sostiene che grazie al suo developer kit è possibile portare innovazione sul mercato in modo più veloce, grazie a un processo semplificato e all'aiuto di maggiori componenti per soddisfare il vasto spettro di casi d'uso dell'IoT. I punti centrali della piattaforma sono:

- **Sicurezza**: è possibile avere dati affidabili grazie a una forte integrazione tra sicurezza hardware e software che inizia dove i dati sono più resistenti agli attacchi.
- **Interoperabilità**: è possibile utilizzare tecnologie che comunicano tra loro senza interruzioni e aiutano ad accelerare i tempi di entrata sul mercato e a ridurre i costi di *deployment* e di mantenimento delle soluzioni IoT.

- **Scalabilità**: possibilità di ottenere una computazione scalabile dal piccolo device fino al cloud, grazie a processori che vanno da Intel Quark[1] fino a Intel Xeon e dispositivi basati su processori Intel, gateway e datacenter.
- **Gestibilità**: possibilità di ottenere una gestione dei dati avanzata dai sensori fino ai datacenter.

Uno dei punti centrai di questa architettura è appunto l'IoT Gateway. Questo permette di connettere dispositivi datati e infrastrutture *next-generation* all'IoT, integrando tecnologie e protocolli di rete, controllori embedded, sicurezza e gestibilità sui quali possono essere eseguite applicazioni di terze parti. Rappresenta quindi è un componente critico all'interno di questo framework ed è il risultato della collaborazione tra diverse aziende per permettere il trasferimento di dati in modo continuativo e senza interruzioni dagli oggetti al cloud, che vede come elementi fondamentali l'utilizzo di dispositivi e software pre-approvato. Intel IoT Gateways inoltre permette agli sviluppatori di creare e distribuire in modo flessibile, sicuro ed economico soluzioni IoT per una vasta gamma di segmenti business. Intel IoT Gateways offre una scelta di processori Intel per le diverse esigente applicative, il supporto per diversi sistemi operativi come Ubuntu Linux e Microsoft Windows 10 e una gestione robusta delle funzionalità dei dispositivi.[9]

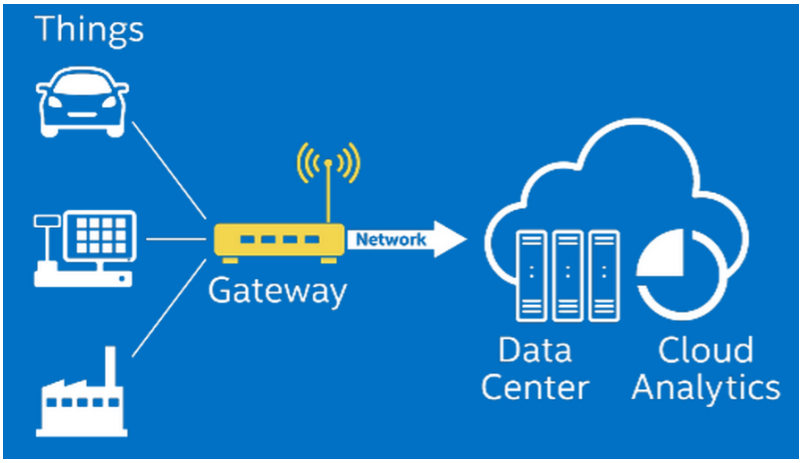



Figura 2.5: Intel IoT Platform[9]

[1]Intel Quark è una linea di System on Chip (SoC) a 32-bit di Intel, ideata per dispositivi di piccole dimensioni e dal basso consumo energetico, utilizzata in nuovi settori di mercato come per esempio nei dispositivi indossabili.

## 2.5 Project Brillo

Google ha creato un nuovo progetto con il nome in codice di "Brillo"[1]. Annunciato al Google I/O 2015, l'evento annuale tenuta da Google per presentare nuovi prodotti o tecnologie, sarà disponibile come anteprima per gli sviluppatori alla fine del 2015, si pone come obiettivo quello di governare gli oggetti dell'Internet of Things all'interno della casa. Brillo sarà un sistema operativo minimale derivato da Android, in modo da potersi adattare alla risorse limitate degli oggetti dell'IoT, ma comunque garantendo la compatibilità con un vasta gamma di piattaforme hardware e facendo leva sui numerosi hardware supportati da Android. Google inoltre afferma che in questo modo gli sviluppatori si potranno concentrare solo sullo scrivere applicazioni per Brillo o a costruire nuovo hardware, dato che il resto è già stato implementato. Per garantire la compatibilità e interoperabilità tra i vari componenti, i produttori di hardware dovranno passare attraverso un programma di certificazione. Oltre a Brillo è stato presentato anche un linguaggio per far comunicare i dispositivi tra di loro, chiamato Weave. Questo è indipendente da Brillo, probabilmente per spingerne l'adozione come standard. I dispositivi Android saranno in grado di rilevare automaticamente e interagire con i device Brillo grazie a Weave e all'integrazione all'interno del *Google Play services*[2] e il supporto sarà facilmente disponibile anche in iOS. Con questo protocollo sarà possibile dialogare sia con i dispositivi direttamente o attraverso il cloud come possibile vedere in Figura 2.6. Ovviamente Weave sarà compatibile con Nest[3] di Google, che comprende il termostato "intelligente", il rilevatore di fumo e un sistema di video sorveglianza. Questo supporto è allargato anche a tutti i dispositivi dell'ecosistema Nest, denominata *Works with Nest*[4], ovvero che possono interagire con il termostato al momento punto centrale della piattaforma. In passato Google aveva già provato a proporre una soluzione per la domotica con "Android@Home", che non prevedeva l'utilizzo dell'Internet of Things, ma il progetto non è andato a buon fine.

[1] https://developers.google.com/brillo/

[2] Google Play Services è un servizio proprietario installato nei dispositivi Android che fornisce accesso a API per accesso ai servizi offerti da Google, come la localizzazione.

[3] https://nest.com/

[4] https://nest.com/works-with-nest/

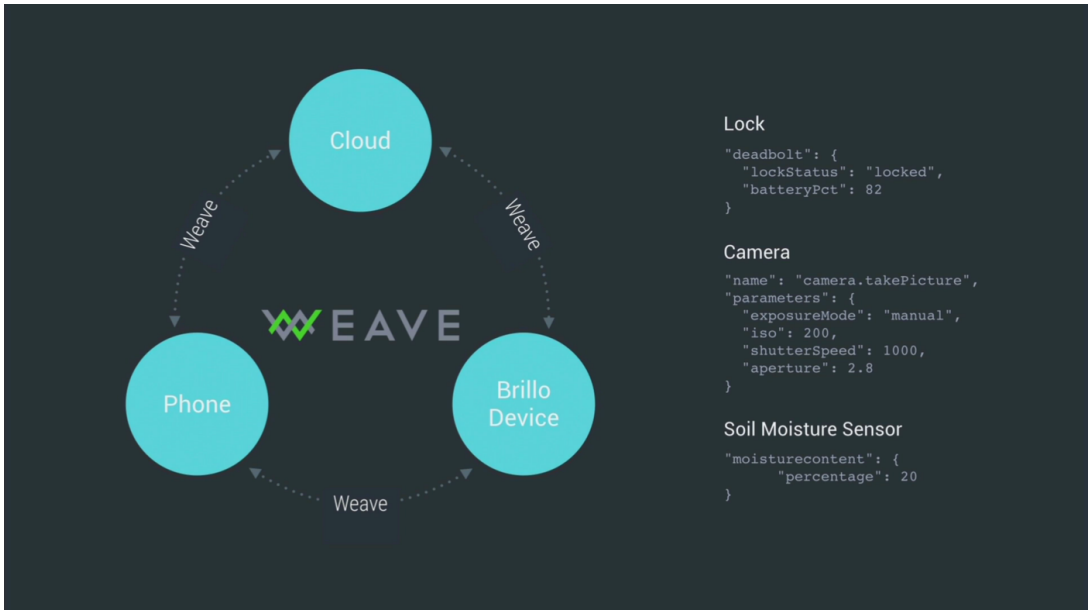



Figura 2.6: Piattaforma Weave di Google[1]

## 2.6 IPv6 e possibilità di indirizzamento univoco

Internet Protocol version 6 (IPv6) è la versione dell'Internet Protocol designata come successore dell'IPv4, la versione attualmente in uso. Uno dei motivi principali sottostante l'adozione di questa nuova versione è l'insufficienza di spazio per l'indirizzamento di dispositivi in rete che si ha con la versione 4. Quindi la caratteristica più importante è quella di un più ampio spazio di indirizzamento a disposizione:

- **IPv6** riserva 128 bit per gli indirizzi IP e gestisce $2^{128}$ (circa $3,4 \times 10^{38}$) indirizzi;
- **IPv4** riserva 32 bit per l'indirizzamento e gestisce $2^{32}$ (circa $4,3 \times 10^{9}$) indirizzi.

In altri termini, con l'IPv4 sono disponibili solamente 4.294.967.296 indirizzi, mentre con l'IPv6 questo numero aumenta in maniera considerevole arrivando a 340.282.366.920.938.463.463.374.607.431.768.211.456 indirizzi che si traducono

[1] http://www.theverge.com/2015/5/28/8683147/google-brillo-weave-internet-of-things-solution

in 655.570.793.348.866.943.898.599 indirizzi IPv6 unici per ogni metro quadrato di superficie terrestre, mentre per l'IPv4 sono solamente 7 per ogni chilometro quadrato. Questi numeri enormi fanno capire che sarà possibile avere un indirizzo IP univoco per ogni singolo oggetto o addirittura ogni singolo componente ed essere in grado di raggiungerlo in qualsiasi momento da qualsiasi luogo. Proprio per questo motivo rappresenta uno dei punti centrali dell'Internet of Things.

## 2.7 Grande mole di dati

L'Internet of Things genera un grande volume di dati e al momento solo una piccola parte di questi raggiunge i data center. Sebbene il traffico dei data center è previsto raggiungerà 8.6 ZB[1] nel 2018, la quantità di dati generata dall'IoT nel 2018 sarà più di 400 ZB o in altre parole circa 50 volte la somma del traffico dei data center. Ad esempio:

- Un Boeing 787 genera 40 terabyte (TB) di dati per ogni ora di volo, di cui solo 0.5 TB di questi sono trasmessi a un data center per essere analizzati o conservati.
- Un grande negozio raccoglie approssimativamente 10 GB di dati ogni ora e soltanto 1 GB di questi viene tramesso a un data center.
- Un impianto di produzione automatizzato genera circa 1 TB di dati ogni ora e solo 5 GB sono trasmessi a un data center.
- Una operazione di estrazione mineraria di una grossa azienda può generare fino a 2.4 TB di dati al minuto

Con il migliorare della capacità dei data center molti più dati generati localmente dai dispositivi potrà essere inviata a quest'ultimi. Globalmente la quantità di dati creata dall'IoT passerà dai circa 110 ZB all'anno registrati nel 2013 a circa 400 ZB nel 2018. Inoltre questa quantità di dati sarà 277 volte superiore a quella trasmessa dai device degli utenti e 47 volte superiore al traffico inviato ai data center, previsto per il 2018.[5] Quindi una delle sfide più importanti sarà quella

[1] Zettabyte. 1 ZB equivale a 1 miliardo di terabyte

di interpretare questa grande mole di dati e di "dare un senso" a tutte le letture dei vari sensori.

## 2.8 Adozione

L'Internet of Things è ancora nelle prime fasi di adozione, ma ha già una vasta varietà d'usi in costante espansione. Con i proliferare delle tecnologie applicate all'Internet of Things aumenta il potenziale per soddisfare alcuni dei più grossi bisogni, tra cui il miglioramento della produttività delle risorse e la gestione delle infrastrutture. *Smard grid* per l'elettricità, l'acqua e la rete dei trasporti ne sono un esempio. Le aziende elettriche e idriche sono state tra le prime a sfruttare queste tecnologie. I sensori sono essenziali per i sistemi basati su smart grid perchè danno agli operatori una modalità per monitorare l'utilizzo e lo stato della rete in tempo reale. Questo significa che anziché attendere una chiamata del cliente che si ritrova senza luce, la compagnia elettrica può individuare malfunzionamenti mentre stanno accadendo e in certe circostanze persino ripristinare il servizio reindirizzando la corrente per evitare il componente danneggiato. Una compagnia di servizi americana ha installato una sistema di smart grid e una rete elettrica aggiornata, che permette di offrire agli utenti un servizio senza interruzioni. Questi sensori, connessi a Internet, vengono anche utilizzati per fare rilevazioni sismiche sotterranee e monitorare il flusso d'acqua all'interno dei tubi di fornitura. Nell'industria energetica i sensori sono utilizzati per mappare aree ricche di combustibili fossili inesplorate e per individuarne l'esatta posizione.

Le tecnologie dell'Internet of Things posso anche avere un impatto diretto nelle vite umane e nella salute. Il concetto chiamato "Quantified Self", che involve l'uso di sensori per tracciare le prestazioni degli esercizi o monitorare la salute, è un trend sempre più popolare alimentato dalle tecnologie IoT. Per esempio diverse aziende commercializzano *wearable device*, sensori indossabili che permetto all'utente di tracciare il numero di chilometri che ha corso, il battito cardiaco e altri dati generati durante un esercizio fisico, che possono poi essere utilizzati per controllare la salute. I medici eseguono "endoscopie capsulari" utilizzando delle micro-camere a forma di pillola, con sensori wireless capaci di comunicare dati

mentre viaggiano all'interno del sistema digestivo di un paziente, trasmettendo le immagini a un computer.

Numerosi avanzamenti tecnologici stanno migliorando l'efficacia delle applicazioni dell'Internet of Things e contemporaneamente riducendo i costi. Il prezzo dei tag RFID e dei sensori sta precipitando e nuovi sviluppi come i MEMS stanno consentendo nuovi usi. La vendita dei sensori è cresciuta annualmente del 70% dal 2010 e gli avanzamenti tecnologici stanno rendendo più accessibili sensori più sofisticati. Sempre più tipologie di sensori vengono integrati nei dispositivi e una migliorata gestione dell'energia permette a questi device di eseguire operazioni per periodi più lunghi senza la necessità di interventi. Le tecniche di miniaturizzazione e i grossi volume di produzione rendono possibile installare i sensori persino nei device più piccoli. Per esempio uno smartphone può avere un singolo chip che include un sensore di posizione, un termometro e un rilevatore di movimento. Inoltre la diffusione di connessioni wireless ad alta velocità sta ampliando l'area coperta dall'Internet mobile, aiutando ad aprire la strada a maggiori usi dell'Internet of Things.[12]

L'azienda americana Cisco, nel 2013, stimava che il 99.4% degli oggetti fisici sono ancora scollegati. Questo significa che solo circa 10 miliardi dei 1500 miliardi di oggetti a livello globale, sono collegati. A un livello più individuale, ci sono ad oggi circa 200 oggetti collegabili per persona. Questi dati mettono in luce il vasto potenziale del collegamento tra tutti gli oggetti non ancora collegati.

Ad ogni modo, la crescita di Internet è senza precedenti. È stato stimato che nell'anno 2000 ci fossero circa 200 milioni di dispositivi connessi ad Internet. Spinti dai progressi della tecnologia mobile e da tendenze come "*bring your own device*"[1] (BYOD), tra le altre cose, questo numero è aumentato fino ad arrivare ai circa 10 miliardi di oggi, facendoci entrare nell'era dell'Internet of Things. La prossima ondata di crescita significativa di Internet avverrà grazie alla confluenza di persone, processi, dati e cose, quella che Cisco chiama Internet of Everything.[10] In Figura 2.7 è possibile vedere questo trend di crescita che porterà ai 50 miliardi di dispositivi connessi nel 2020. Sono presenti quattro fasi: la

[1] "Porta il tuo dispositivo" è un'espressione utilizzata per riferirsi alle politiche aziendali che permettono di portare i propri dispositivi personali nel posto di lavoro, e utilizzarli per avere gli accessi privilegiati alle informazioni aziendali e alle loro applicazioni

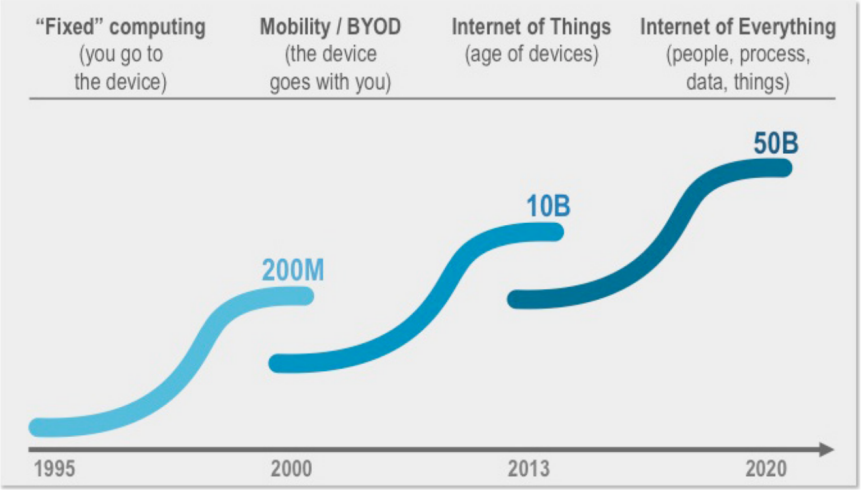



Figura 2.7: Rapida crescita degli oggetti connessi a Internet[1]

prima che inizia nel 1995, "Fixed computing" dove è l'utente ad andare al dispositivo; la seconda nel 2000, "Mobility" dove il dispositivo si muove con l'utente; la terza nel 2013, "Internet of Things" l'era dei dispositivi; l'ultima nel 2020, "Internet of Everything" dove qualsiasi cosa è connessa, persone, cose, processi e dati.

L'IoT è spinto ulteriormente da molti altri fattori. In primo luogo dai trend tecnologici, incluso il sostanziale aumento della potenza di elaborazione, della capacità di *storage* e della larghezza di banda a prezzi sempre minori; la rapida crescita del cloud, dei social media, e del mobile computing; la capacità di analizzare *Big Data*[2] e trasformali in informazioni utili; infine una migliore capacità di combinare tecnologie (sia hardware che software) in modi più significativi, permettendo di ricavare maggior valore dalla loro connessione.

In secondo luogo, le barriere alla connettività continuano a diminuire. Per esempio, come affrontato nella Sezione 2.6, l'IPv6 supera i limiti dell'IPv4 permettendo di connettere un numero altissimo di dispositivi, persone, processi dati e cose a Internet. Per l'IPv4 il numero di indirizzi era limitato a 4.294.967.296. Incredibilmente, l'IPv6 permette di crearne abbastanza per fornire 4.800 miliardi

[1]Fonte: Cisco IBSG, 2013 [10]

[2]Big data è il termine utilizzato per descrivere una raccolta di dati così estesa in termini di volume, velocità e varietà da richiedere tecnologie e metodi analitici specifici per l'estrazione di valore.

di indirizzi per ogni stella conosciuta nell'universo.

Inoltre, i *form factor* continuano a ridursi. Ad oggi un computer della grandezza di un granello di sale (1x1x1 mm) comprende una cella solare, una batteria dello spessore di una pellicola, la memoria, un sensore di pressione e un'antenna wireless e radio. Una telecamera del medesimo spessore ha una risoluzione di 250x250 pixel. Un sensore delle dimensioni di un granello di polvere (0.05x0.005 mm) può rilevare e comunicare la temperatura, pressione e movimento. Questi sviluppi sono importanti perché in futuro gli oggetti connessi ad Internet potranno essere talmente piccoli da essere persino difficili da vedere ad occhio nudo.

Infine, l'Internet of Everything riflette che il valore delle creazioni si è spostato verso la potenza dei collegamenti e più nello specifico verso l'abilità di creare informazioni per quelle connessioni. Le aziende non possono più fare affidamento esclusivamente sulle competenze chiave interne e sulla conoscenza dei loro dipendenti. Hanno bisogno anche di catturare velocemente informazioni da svariate sorgenti esterne. Questo avverrà tramite le connessioni rese possibili dall'Internet of Thing (Everything).[10]

Il *Value at Stake*, secondo Cisco, è il valore dei potenziali profitti (maggiori entrate e minori costi), che può essere creato per le aziende che saranno capaci di sfruttare l'IoT. Cisco prevede che il Value at Stake dell'Internet of Things sarà di 14.4 mila miliardi di dollari per le aziende e le industrie di tutto il mondo, nel prossimo decennio. Nello specifico, nei prossimi dieci anni, questo rappresenta una opportunità per incrementare i profitti globali delle aziende di circa il 21%.

In altre parole, da oggi al 2022, un valore di 14.4 mila miliardi di dollari generato dall'IoT, sarà disponibile per le imprese a livello globale. L'IoT creerà sia nuovo valore, sia ridistribuirà valore tra *early adopter* e i ritardatari, in base a quanto sapranno sfruttare le opportunità presentate dall'Internet of Things.

A livello globale, l'analisi di Cisco dimostra che la maggior parte del potenziale Value at Stake (66%, o 9.5 mila miliardi di dollari) si ha dalla trasformazione basata su casi d'uso specifici per le industrie, come smart grid, smart factories, smart building[1]. Il restante 34% è prodotto dai tre casi d'uso *cross-industry*, come la riduzione del time to market[2] o l'*outsourcing* dei processi aziendali. In

[1] reti, fabbriche e edifici intelligenti

[2] il tempo che intercorre dall'ideazione di un prodotto alla sua effettiva commercializzazione

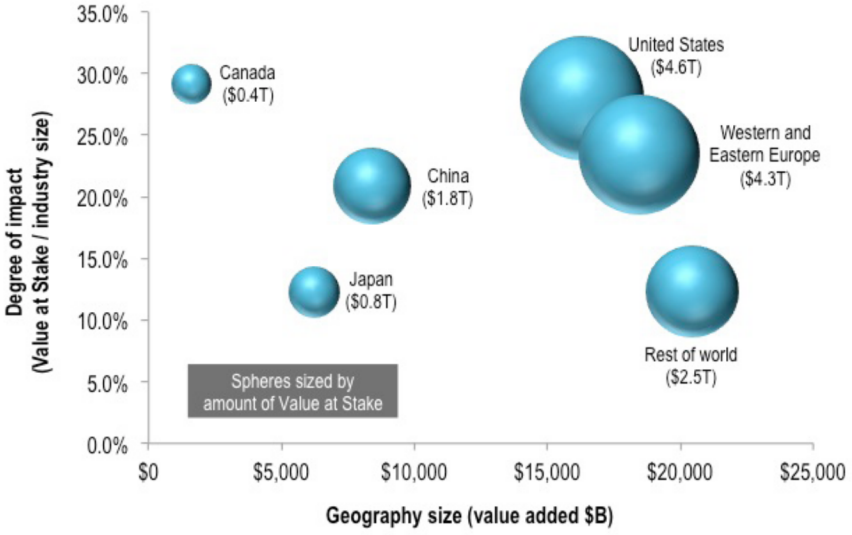



Figura 2.8: Il Value at Stake dell'Europa corrisponde al 30% del totale[1]

Europa 2.6 mila miliardi del potenziale Value at Stake verranno dai casi d'uso specifici, mentre 1.7 mila miliardi saranno dati dai casi d'uso cross-industry.

Cisco ha calcolato il Value at Stake adottando un approccio bottom-up considerando il valore creato da più di 50 casi d'uso solo nel settore privato, sia specifici che cross-industry, e li ha accorpati in 21 esempi significativi sia dal punto di vista del materiale che del valore. È stata inoltre eseguita un'analisi top-down per verificare la completezza e l'ordine di grandezza del più specifico approccio bottom-up. Infine è stato controllato che i valori dei casi d'uso fossero unici.[10]

La quota Europa del Value at Stake è di 4.3 mila milardi di dollari nei prossimi 10 anni. Questo rappresenta il 30% dei 14.4 mila miliardi di dollari a livello globale. Nella Figura 2.8 è inoltre possibile vedere l'impatto benefico per regione, ottenuto dividendo il Value at Stake per la grandezza dell'output economico di ogni regione.

Le aziende europee che sfrutteranno al meglio l'IoT raccoglieranno i frutti del valore in uno dei due seguenti modi:

- Catturando il nuovo valore creato dall'innovazione tecnologica

[1]Fonte: Cisco IBSG, 2013 [10]

- Ottenendo un vantaggio competitivo e guadagnando quote di mercato a discapito di altre aziende che sono state meno capaci di capitalizzare nella transizione di mercato dell'IoT.

A livello mondiale ci sono cinque principali fattori trainanti del Value at Stake. Questi sono illustrati nella Figura 2.9 e sono gli stessi anche in Europa:

- **Utilizzo delle risorse (1.85 mila miliardi di dollari)**
  L'IoT riduce le spese generali, di vendita e amministrative e i costi dei beni venduti migliorando l'esecuzione dei processi di business e l'efficienza del capitale.

- **Produttività dei dipendenti (1.06 mila miliardi di dollari)**
  L'Internet of Things crea una maggiore efficienza sul lavoro che risulta in minori ore di lavoro rendendole più produttive.

- **Innovazione, inclusa la riduzione del time to market (766 miliardi di dollari)**
  L'IoT aumenta il ritorno sugli investimenti in Ricerca e Sviluppo, riduce i tempi del time to market e crea ulteriori flussi di entrate grazie ai nuovi business model e opportunità.

- **Catene di distribuzione e logistica (440 miliardi di dollari)**
  L'Internet of Things elimina gli sprechi e migliora l'efficienza dei processi

- **Esperienza utente (206 miliardi di dollari)**
  L'IoT aumenta il valore della durata dei cliente e incrementa la quota di mercato aggiungendone altri.

Come mostrato in Figura 2.10, l'Internet of Things include tre tipi di connessioni: Machine to Machine (M2M), Person to Machine (P2M) e Person to Person (P2P). Combinate, le connessioni P2M e P2P contribuiranno per il 58% del valore totale dell'IoT entro il 2022, mentre le connessioni M2M contribuiranno per il rimanente 42%. È importante notare che mentre le connessioni M2M stanno rapidamente diventando una fonte considerabile di valore, il risultato finale di

[1]Fonte: Cisco IBSG, 2013 [10]

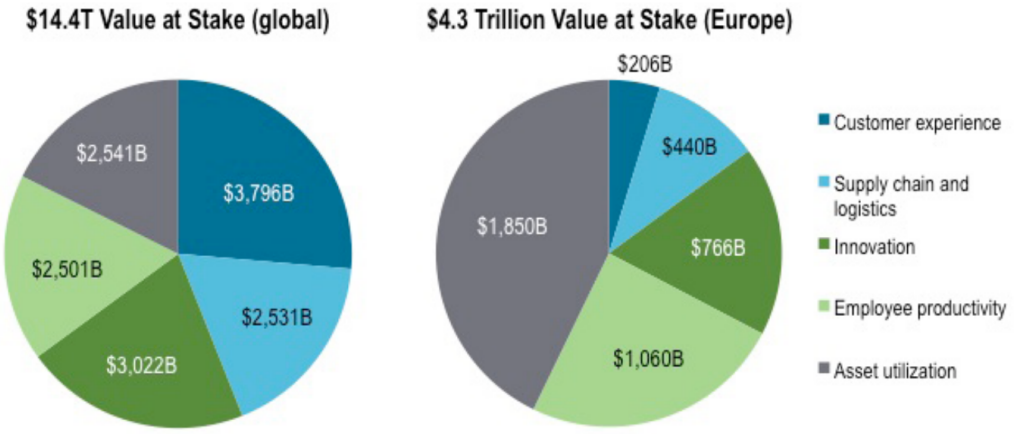



Figura 2.9: Suddivisione del Value at Stake per l'Europa[1]

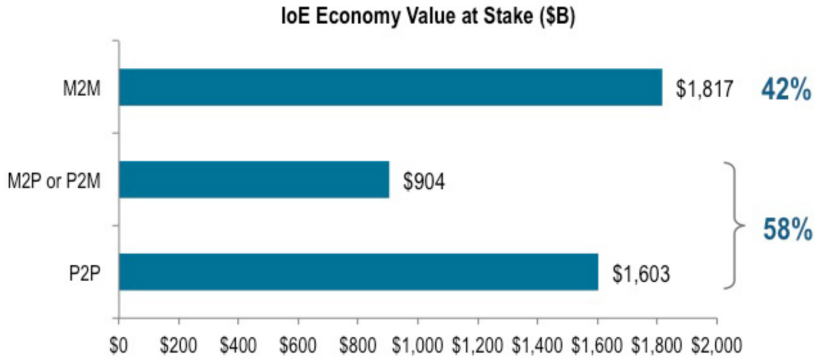



Figura 2.10: Economia dell'Internet of Things[2]

queste connessioni è sempre quello di portare benefici alla persone. In definitiva, l'economia dell'Internet of Things riguarda la possibilità di permettere alle persone di essere più produttive ed efficaci, prendendo decisioni migliori e di godere di una migliore qualità della vita.

[2]Fonte: Cisco IBSG, 2013 [10]

# Capitolo 3

# Physical Web

## 3.1 Physical Web

Come illustrato in precedenza, grazie all'Internet of Things sempre più dispositivi stanno diventando "intelligenti" e connessi a Internet. Tra le nuove esigenze che si vengono a creare troviamo quella di individuare questi oggetti, scoprendo così la loro esistenza e la possibilità di interagire con essi in modo semplice e immediato. Il progetto "Physical Web" si pone come obbiettivi, da una parte di pubblicizzare la presenza degli oggetti *smart* attraverso dei beacon Bluetooth, chiamati Eddystone-URL e dall'altra quello di intercettare e interpretare questi segnali. Questo progetto, pur essendo proposto da Google, è open source e non brandizzato, in modo da incentivarne l'adozione come standard.

L'idea alla base del Physical Web è quella di estendere le potenzialità del Web al mondo fisico intorno a noi. Questo comprende quindi la creazione di un ecosistema dove gli oggetti smart possono trasmettere URL nell'area intorno a loro. Ogni oggetto che dispone di un display, come un tablet o uno smartphone nelle vicinanze può intercettare questo URL, interpretarlo e mostrarlo all'utente.

Si rispecchia sostanzialmente il comportamento presente oggi in un motore di ricerca:

- L'utente richiede un lista delle cose nelle vicinanze
- Viene mostrata una lista ordinata

- L'utente sceglie un elemento
- L'URL viene aperto e mostrato all'interno di un browser

La parte che si occupa di pubblicizzare l'URL si chiama Eddystone-URL.

## 3.2 Eddystone

Eddystone è la specifica di un protocollo che definisce il formato dei messaggi inviati da dispositivi Bluetooth Low Energy, chiamati messaggi beacon. Descrive diversi tipi di frame che possono essere utilizzati individualmente o combinati per creare dei beacon che possono essere utilizzati per una varietà di applicazioni. La progettazione di Eddystone si è concentrata principalmente su alcuni aspetti chiave:

- Funzionamento con le API Bluetooth di Android e iOS
- Semplice implementazione per una vasta gamma di dispositivi Bluetooth Low Energy esistenti
- Compatibilità con le specifiche del *Bluetooth Core Specification*[1]
- Architettura flessibile che consente lo sviluppo di nuovi tipi di frame

Al momento sono disponibile tre tipi di frame: Eddystone-URL, Eddystone-UID e Eddystone-TLM

### 3.2.1 Elementi comuni

Ogni frame Eddystone deve contenere i seguenti tipi di dati PDU:

- Il tipo di dato *Service Solicitation* come definito nel *Bluetooth Core Specification Supplement* (CSS) v5. Il *Service Solicitation UUIDs*, lista di 16 bit, deve contenere il Service UUID di Eddystone: 0xFEAA. Questo viene incluso per permettere lo scan in background da parte dei dispositivi iOS.[16]

[1] https://www.bluetooth.org/en-us/specification/adopted-specifications

- Il tipo di dato *Service Data*, lista di 16 bit, deve essere il Service UUID di Eddystone: 0xFEAA [16]

Il tipo specifico di frame è codificato nei quattro bit di ordine superiore del primo ottetto nel Service Data associato al Service UUID.[8] I possibili valori sono mostrati in Figura 3.1. I quattro bit di ordine inferiore sono riservati per usi futuri

| Frame Type | High-Order 4 bits | Byte Value |
|---|---|---|
| UID | `0000` | `0x00` |
| URL | `0001` | `0x10` |
| TLM | `0010` | `0x20` |

Figura 3.1: Lista dei tipi di frame con i corrispettivi valori in bit[8]

e devono essere 0000.

### 3.2.2 Eddystone-URL

Il frame Eddystone-URL fa il broadcast di un URL utilizzando un formato di codifica compressa in modo da adattarsi agli spazi limitati del pacchetto di advertisment. Una volta codificato, l'URL può essere utilizzato da qualsiasi client con acceso a Internet. Per esempio, un client nelle vicinanze di un beacon Eddystone-URL che trasmette l'URL https://goo.gl/WVBu4W, può decidere di visitarlo, in questo caso risolvendo l'indirizzo di uno dei distributori automatici della simulazione.

Il frame Eddystone-URL, incorpora tutte le caratteristiche di UriBeacon, del quale ne è l'evoluzione. Questo frame è la colonna portante del Pyhsical Web, un progetto con lo scopo di facilitare la scoperta di contenuti web nelle vicinanze. Le specifiche del frame sono mostrate in Figura 3.2. Il campo *Frame Type*, come detto in precedenza, specifica il tipo di frame.

| Byte offset | Field | Description |
|---|---|---|
| 0 | Frame Type | Value = `0x10` |
| 1 | TX Power | Calibrated Tx power at 0 m |
| 2 | URL Scheme | Encoded Scheme Prefix |
| 3+ | Encoded URL | Length 0-17 |

Figura 3.2: Specifiche del tipo di frame Eddystone-URL[8]

### Tx Power Level

*Tx power* è la potenza ricevuta a 0 metri, in dBm. Questo valore può variare da -100dBm a +20 dBm con granularità di 1 dBm.

Il modo migliore per determinare un valore preciso da inserire in questo campo è quello di misurare l'output reale a 1 metro di distanza dal beacon e quindi aggiungere 41dBm a questo valore. 41dBm è la perdita di segnale che si ha per ogni metro di distanza Il valore di questo campo è un numero intero con segno a 8 bit, come specificato da *TX Power Level Bluetooth Characteristic*.[15][8]

### URL Scheme Prefix

Il byte *URL Scheme Prefix* definisce lo schema di identificazione, un prefisso opzionale che come il resto dell'URL è codificato.

### HTTP URL encoding

Lo schema degli URL HTTP, definito dal documento RFC 1738 [4], può essere ad esempio "https://goo.gl/WVBu4W", ed è utilizzato per indicare una risorsa Internet accessibile tramite HTTP (HyperText Transfer Protocol).

La codifica consiste in una sequenza di caratteri. I codici carattere esclusi dall'encoding dell'URL, sono utilizzati come codici di espansione del testo.

| Decimal | Hex | Expansion |
|---|---|---|
| 0 | 0x00 | `http://www.` |
| 1 | 0x01 | `https://www.` |
| 2 | 0x02 | `http://` |
| 3 | 0x03 | `https://` |

Figura 3.3: Schema dei prefissi per gli URL nel frame Eddystone-URL[8]

Quando uno *user agent* riceve un Eddystone-URL, i byte codificati nell'URL sono sostituiti tramite l'espansione del testo seguendo lo schema in Figura 3.4.

### 3.2.3 Eddystone-UID

Il frame Eddystone-UID fa il broadcast di un Beacon ID di 16-byte, opaco e unico. Questo si suddivide in un *namespace ID* di 10 byte e di un *instance ID* di 6 byte, rispettivamente una stringa univoca, identificativa del beacon e un codice di istanza scelto in base all'applicazione. Il Beacon ID può essere utile per mappare un dispositivo in un database. Il namespace ID può essere utilizzato per raggruppare una particolare serie di beacon, mentre l'instance ID identifica il singolo dispositivo all'interno del gruppo. La suddivisione del namespace e instanza può essere anche utilizzata per ottimizzare le strategie di scansione dei dispositivi BLE, per esempio filtrando solo in base al namespace.[8]

### 3.2.4 Eddystone-TLM

Il frame Eddystone-TLM fa il broadcast di informazioni di telemetria riguardanti il beacon stesso come valori di voltaggio della batteria, temperatura del dispositivo, conteggio dei pacchetti di broadcast inviati. Dato che il frame Eddystone-TLM non contiene un ID del beacon, deve essere accoppiato con un frame che

| Decimal | Hex | Expansion |
|---|---|---|
| 0 | 0x00 | .com/ |
| 1 | 0x01 | .org/ |
| 2 | 0x02 | .edu/ |
| 3 | 0x03 | .net/ |
| 4 | 0x04 | .info/ |
| 5 | 0x05 | .biz/ |
| 6 | 0x06 | .gov/ |
| 7 | 0x07 | .com |
| 8 | 0x08 | .org |
| 9 | 0x09 | .edu |
| 10 | 0x0a | .net |
| 11 | 0x0b | .info |
| 12 | 0x0c | .biz |
| 13 | 0x0d | .gov |
| 14..32 | 0x0e..0x20 | Reserved for Future Use |
| 127..255 | 0x7F..0xFF | Reserved for Future Use |

Figura 3.4: Encoding per gli URL nel frame Eddystone-URL[8]

fornisca un ID, cioè un Eddystone-UID o Eddystone-URL e la trasmissione dovrà essere intervallata a uno di questi due.[8]

# Capitolo 4

# Tecnologie alternative

## 4.1 Beacon

Sono presenti due principali alternative agli Eddystone: AltBeacon e iBeacon. I motivi principali per cui non sono stati utilizzati in questa tesi derivano dal fatto che gli iBeacon sono proprietari e non ufficialmente supportati in Android, mentre gli AltBeacon richiedono un'ulteriore libreria esterna per l'utilizzo. Inoltre, non meno importante, nessuno dei due offre l'elasticità offerta dagli Eddystone, in quanto consentono un solo tipo di advertisement, utilizzabile solo all'interno di applicazioni native. Anche nel caso dell'utilizzo degli Eddystone-UID all'interno dell'applicazione nativa, questi permettono una maggiore flessibilità grazie alla possibilità di poter associare ad essi dei dati, in qualsiasi momento.

### 4.1.1 iBeacon

iBeacon è un protocollo proprietario di Apple introdotto nel 2013 e richiede una licenza per poter sviluppare hardware compatibile.[3] Come è possibile vedere in Figura 4.1, l'identificativo del beacon si suddivide in tre parti:

- **UUID**: un identificativo univoco specifico per l'applicazione o il caso d'uso.
- **Major**: un ulteriore identificativo che può indicare per esempio una sotto-regione della più grande regione specificata dall'UUID.

| | Eddystone | AltBeacon | iBeacon | Proprietary |
|---|---|---|---|---|
| **Range** | ~50 meters | ~50 meters | ~50 meters | ~50 meters |
| **Official Android Support** | YES | YES | Unofficial | YES |
| **Official iOS Support** | YES | YES | YES | YES |
| **Open standard?** | YES | YES | NO | NO |
| **Multiple Vendors** | YES | YES | YES | NO |
| **Identifiers** | • 10 byte namespace<br>• 6 byte instance | • 16 byte id1<br>• 2 byte id2<br>• 2 byte id3 | • 16 byte UUID<br>• 2 byte major<br>• 2 byte minor | single |
| **Interoperable with iBeacon?** | NO | YES | YES | NO |
| **Introduced** | July 2015 | July 2014 | June 2013 | Various |

Figura 4.1: Confronto tra Eddystone, AltBeacon, iBeacon e altri[2]

- **Minor**: un ultimo identificativo che permette un'ulteriore suddivisione specifica per una regione.

Nella Figura 4.2 è possibile vedere un esempio di questa suddivisione per dei negozi in tre diverse città: l'UUID è sempre lo stesso e viene utilizzato per identificare la catena, il Major cambia per ognuna di queste città, mentre il Minor viene utilizzato per indicare il reparto di ciascun negozio. Sono presenti due modalità di utilizzo:

- **Monitoraggio della regione**: In modo simile alle geofence, un'applicazione può richiedere di essere notificata, anche quando si trova in background, se un utente entra o esce da una regione definita da un beacon. Quando l'applicazione fa questa richiesta di iniziare a monitorare una regione, deve specificare l'UUID. Un'applicazione può monitorare fino a 20 regioni, ma utilizzando un singolo UUID per diverse posizioni, come nell'esempio dei negozi fatto prima, una singola applicazione può arrivare a monitorare molte più posizioni fisiche.

- **Ranging**: Al contrario del monitoraggio che consente di monitorare entrate/uscite dal range di azione del beacon, il *ranging* fornisce una lista di tutti

| Store Location | | San Francisco | Paris | London |
|---|---|---|---|---|
| UUID | | D9B9EC1F-3925-43D0-80A9-1E39D4CEA95C | | |
| Major | | 1 | 2 | 3 |
| Minor | Clothing | 10 | 10 | 10 |
| | Housewares | 20 | 20 | 20 |
| | Automotive | 30 | 30 | 30 |

Figura 4.2: Esempio di utilizzo di un iBeacon[3]

i beacon individuati in una data regione contenente anche la distanza dal dispositivo a questi. Il ranging funziona solamente quando l'applicazione è aperta, in primo piano. La distanza ricevuta dai beacon viene categorizzata in quattro diversi stati: *immediate*, indica con un alto livello di fiducia che il beacon è fisicamente molto vicino al dispositivo ed molto probabile che sia praticamente sopra; *near*, se non è presente nessun ostacolo tra il beacon e il dispositivo, questo si trova a circa 1-3 metri di distanza, ma se sono presenti degli ostacoli che causano un'attenuazione del segnale, questo stato potrebbe anche non comparire mai; *far*, indica che è stato individuato un beacon ma che il livello di fiducia non è abbastanza elevato per determinare se sia *near* o *immediate*, ma va tenuto in considerazione che questo non vuol dire necessariamente che il beacon non sia vicino; *unknown*, la prossimità del beacon non può essere determinata e questo può indicare che non ci sono abbastanza dati presenti per determinare uno stato o che il ranging è appena iniziato.[3]

### 4.1.2 AltBeacon

AltBeacon è una specifica di protocollo per il broadcast di beacon *open* ideata da Radius Networks[3] nel 2014, per sopperire alla mancanza di uno standard aperto e alla disponibilità di beacon per la piattaforma Android. Inizialmente

[2]http://developer.radiusnetworks.com/
[3]http://www.radiusnetworks.com/

era stata sviluppata una libreria per interagire con gli iBeacon, ma Apple non ne permette l'utilizzo all'interno di sistemi Android ed è quindi stata ritirata.[18] Successivamente è quindi stata ideata una piattaforma compatibile con quella di Apple, ma aperta e disponibile per qualsiasi sistema operativo, così da poter garantire un supporto anche se in maniera non ufficiale. È possibile vedere un confronto tra queste tecnologie e quella Eddystone nella Figura 4.1 dove viene specificata l'interoperabilità tra AltBeacon e iBeacon grazie a una struttura degli identificativi simile. Si può notare come l'identificativo diviso in tre parti rispecchi quello degli iBeacon e ne garantisca quindi la compatibilità:

- **id1**: 16 byte per l'identificazione univoca dell'applicazione
- **id2**: 2 byte per una suddivisione dei casi d'uso
- **id3**: ulteriori 2 byte per una maggiore specializzazione del caso d'uso

Anche per quanto riguarda gli strumenti per individuare i beacon, sono gli stessi degli iBeacon, descritti nella sezione precedente: Monitoring e Ranging. Radius Networks, oltre ad aver ideato un protocollo, vende diversi tipi dei beacon di cui viene garantita la compatibilità con i tre diversi tipi di protocollo. In questi è possibile utilizzare contemporaneamente iBeacon e AltBeacon, alternandoli, mentre se si vuole fare il broadcast di Eddystone non sarà possibile utilizzare gli altri due.

### 4.1.3 Android Beacon Library

Rappresenta la libreria ufficiale per individuare AltBeacon, ma che permette di poter utilizzare anche qualsiasi altro tipo di beacon all'interno di Android. Come accennato precedentemente sono supportati ufficialmente solo gli AltBeacon e gli Eddystone, ma vista la similarità dei primi con gli iBeacon, vengono supportati anche quest'ultimi. La libreria viene caricata all'avvio del sistema operativo ed è in grado di lanciare applicazioni anche se l'utente non le ha ancora aperte (è sufficiente che le abbia avviate almeno una volta dopo l'installazione) e di gestirne la loro esecuzione. Infatti implementando un'opportuna classe all'interno dell'applicazione, questa può essere risvegliata dalla libreria all'avvistamento di

un beacon e verrà lanciata una normale attività, che potrebbe risultare addirittura nell'apertura dell'applicazione senza nessuna richiesta dell'utente.[14] Questo oltre a risultare in un forte utilizzo della batteria del dispositivo, visto che la libreria in automatico effettuerà una scansione dei beacon ogni 5 minuti, se non impostata diversamente, può anche risultare in una esperienza utente poco gradevole. Visto che sono presenti all'interno di Android delle API per la piattaforma beacon di Google e quindi degli Eddystone, mi è sembrato opportuno utilizzarle, rispetto ad una libreria di terze parti, che oltretutto va contro ad alcune delle *best practice* specificate da Google, come la ricerca in background di beacon e l'invio attivo di notifiche d una volta individuati, anche se non richiesto espressamente dall'utente.

# Capitolo 5

# Implementazione

In questo capitolo verrà trattata l'applicazione dell'Internet of Things e del Physical Web alle vending machines.

È stata creata una Web Application, tramite l'utilizzo di HTML5, CSS, JavaScript e jQuery e una applicazione per Android utilizzando Java. All'interno dell'applicazione web, utilizzando jQuery vengono effettuate chiamate asincrone GET e POST al WebServer, che si comporta come un proxy, per richiedere e trasmettere dati al database endpoint di backend. Questo endpoint, sviluppato in PHP, dopo aver cercato le informazioni richieste nel database, implementato in SQLite, risponde con il JSON appropriato. Essendo le risposte in JSON, lo stesso processo di richiesta e interpretazione è utilizzato con i meccanismi appropriati anche all'interno dell'applicazione Android. La Web Application e l'applicazione Android inoltre comunicano direttamente con il distributore automatico, mediante l'utilizzo di un WebSocket.

Di seguito verranno viste nel dettaglio le varie parti del sistema.

## 5.1 Server

L'implementazione lato server è suddivisa in due parti: **Database** e **Web Server**.

### 5.1.1 Database

Il database si occupa di immagazzinare tutti i dati relativi ai distributori automatici. Sono disponibili due tipi di informazione principali.

Informazioni relative al distributore:

- **Produttore**: il costruttore del distributore automatico
- **Rifornitore**: l'azienda che si occupa di rifornire di prodotti il distributore
- **Posizione**: l'indirizzo in cui è posizionato il distributore
- **Piano**: il piano dell'edificio, se rilevante, in cui è situato
- **Nome della zona**: nominativo della zona in cui sono raggruppati i distributori (es. dipartimento di matematica)

Informazioni relative ai prodotti:

- **Nome**: il nome del prodotto
- **Produttore**: l'azienda produttrice del prodotto
- **Tipologia**: il tipo di prodotto (es. snack, bevanda)
- **Immagine**: la rappresentazione del prodotto da mostrare all'utente

Queste informazioni sono poi poste in relazione tra di loro per conoscere l'effettiva disponibilità dei prodotti all'interno dei singoli distributori. Come risultato finale, in aggiunta alle informazioni precedenti, si possono quindi conoscere:

- **Posizione**: la posizione all'interno del distributore, informazione necessaria per erogare il prodotto
- **Quantità**: la disponibilità relativa al prodotto
- **Prezzo**: il prezzo del prodotto
- **Scadenza**: la data di scadenza del prodotto

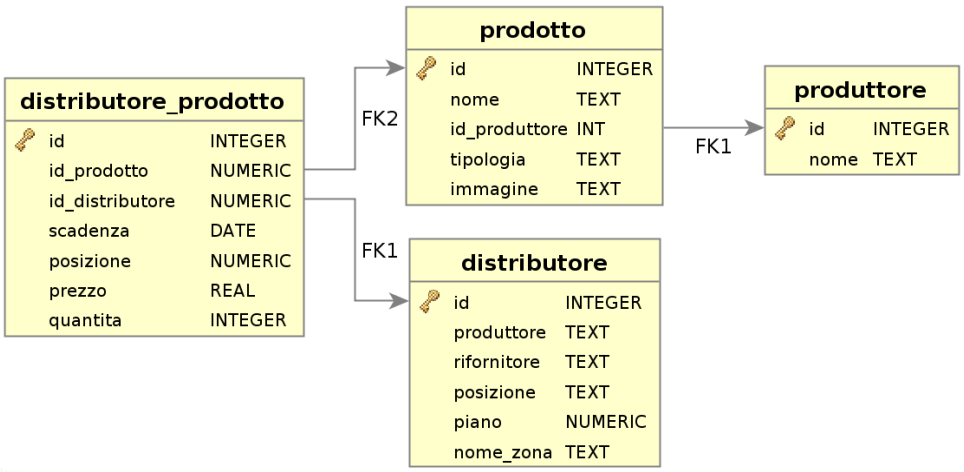



Figura 5.1: Schema del database

```php
function get_all_products($id, $quantita) {
    $query = "SELECT prodotto.nome,
    prodotto.immagine,
    distributore_prodotto.prezzo,
    distributore_prodotto.quantita,
    prodotto.tipologia,
    distributore_prodotto.posizione,
    distributore_prodotto.id_prodotto";
    $query = $query . " FROM prodotto join distributore_prodotto on
    prodotto.id = distributore_prodotto.id_prodotto join
    distributore on distributore_prodotto.id_distributore =
    distributore.id";
    $query = $query . " WHERE distributore.id = " . $id . " and
    quantita >= ". $quantita .";";

    $results = fetch_data($query);

    return $results;
  };
```

Listing 5.1: Codice PHP di una richiesta al database di tutti i prodotti in un distributore

Nella Figura 5.1 è possibile vedere lo schema del database con i relativi vincoli di integrità mentre nella Listing 5.1 è possibile vedere parte del codice PHP che viene eseguito lato server facendo una richiesta di tutti i prodotti all'interno del distributore, passato come parametro alla funzione.

### 5.1.2 Web Server

Il Web Server è ospitato su Google App Engine, principalmente per far fronte alla necessità di tutta la piattaforma di avere un certificato SSL valido. Infatti, per poter utilizzare i *Service Worker*, che verranno illustrati nella Sezione 5.2.3, tutte le pagine devono essere realizzate in HTTPS e di conseguenza anche tutte le comunicazioni tra le varie parti del sistema e le richieste a risorse esterne devono rispettare tale protocollo. All'interno del web server sono ospitate tutte le pagine HTML, con i relativi CSS, le immagini dei prodotti e i file JavaScript. Inizialmente il database e il web server erano collocati sulla stessa macchina, ma in seguito alla necessità di passare a HTTPS, è stata spostata solo la parte web su GAE per due motivi: in primo luogo perché la parte database era già stata implementata in SQLite, non supportato dalla piattaforma, che prevede soltanto un piano a pagamento per poter utilizzare i database proprietari; in secondo luogo perché in questo modo è stato aggiunto un ulteriore strato a protezione del database.

#### Google App Engine

Gogle App Engine (GAE), rappresenta una piattaforma di cloud computing, offerta come *Platform as a Service* (PaaS), che permette di sviluppare e ospitare applicazioni nei data center gestiti da Google. Le applicazioni vengono eseguite in delle *sandbox* e distribuite su più server. Google App Engine offre un ridimensionamento automatico delle risorse per le applicazioni. All'aumentare delle richieste vengono allocate automaticamente maggiori risorse, in modo da fronteggiare la domanda aggiuntiva. Questo processo viene gestito interamente dalla piattaforma e l'utente non può impostare nessun parametro. Nella piattaforma Google App Engine è possibile utilizzare quattro linguaggi per gestire il progetto: Python, Java, PHP e Go. Per questo progetto è stato scelto Python.

La struttura del progetto è gestita da un file YAML, *app.yaml*, in cui sono descritti tutti i path raggiungibili e le operazioni da eseguire quando viene richiesto quel percorso. Un esempio in merito è mostrato in un estratto del codice nella Listing 5.2. All'inizio del file vengono date alcune direttive tra cui il nome dell'applicazione, la versione e il linguaggio da utilizzare, in questo caso Python 2.7. Successivamente sono mostrati gli *handlers*, ovvero le direttive che si occupano di gestire le chiamate, composti da una lista di URL validi per l'applicazione e le operazioni da effettuare quando questi vengono richiesti. Per qualsiasi altro URL al di fuori di quelli elencati, viene mostrato un messaggio di errore.

All'interno del file è possibile incontrare tre diversi tipi di handler, di seguito verranno mostrati alcuni esempi:

- **Singolo file statico**: richiedendo la radice, ovvero "/", viene restituita la Home Page, cioè il file "index.html".
- **Cartella di file statici**: attraverso una *regular expression*, qualsiasi file con estensione .gif, .png o .jpg, richiesto all'interno della cartella "img", contenente le immagini, viene restituito al browser.
- **Script**: questo tipo di handler esegue lo script "main.py" per gestire la richiesta, in questo esempio a "/postdata". Verrà chiamata l'applicazione "app" all'interno dello script, mostrata nella Listing 5.3; per questo motivo viene indicato come "main.app".

```
1  application: vending-machine
2  version: 1
3  runtime: python27
4  api_version: 1
5  threadsafe: yes
6
7  handlers:
8  - url: /
9    static_files: index.html
10   upload: index.html
11 - url: /img/(.*\.(gif|png|jpg))$
12   static_files: img/\1
```

```
13   upload: img/.*\.(gif|png|jpg)$
14 - url: /postdata
15   script: main.app
```

Listing 5.2: Estratto del codice YAML di gestione del progetto

Lo script Python si occupa di gestire le richieste ai database di backend, sia quello contenente i dati relativi ai distributori, sia quello delle iscrizioni alle notifiche push. Sempre per quanto riguarda le notifiche push è presente anche una componente che si occupa dell'invio al client, tramite il Google Cloud Messaging. Al momento le notifiche possono essere inviate solo su Chrome OS, Chrome per dispositivi mobili Android e per qualsiasi Sistema Operativo desktop.

Nella Listing 5.3 viene mostrato il codice che si occupa di gestire gli handler associati allo script e due esempi di richieste. Il primo esempio riguarda una chiamata POST all'endpoint del database delle sottoscrizioni push, invocata tramite */push*. Viene creata la richiesta con l'identificatore univoco in possesso del client, associata al campo "subscription" e invocata la chiamata POST tramite la "urlfetch". In questo modo viene salvata nel database, per poi poter inviare notifiche push successivamente al client. Nella Sezione 5.2.3 verranno illustrate con maggior dettaglio le notifiche push. Nel secondo esempio vengono presi i parametri della URL della chiamata GET, che ha invocato la funzione */getdata*, e vengono composti per creare la chiamata GET all'endpoint del database dei distributori automatici.

```
class PushPage(webapp2.RequestHandler):
  def post(self):
    url = serverUrl+"push.php"
    req = self.request.get("subscription")
    data = {"subscription" : req}
    form_data = urllib.urlencode(data)
    result = urlfetch.fetch(url=url,
                            payload=form_data,
                            method=urlfetch.POST)
    logging.info(req)

class getDataPage(webapp2.RequestHandler):
  def get(self):
    url = serverUrl+"getdata.php?"
    args = self.request.arguments()
    data = {}
    for a in args:
      data[a] = self.request.get(a)

    form_data = urllib.urlencode(data)
    logging.info(form_data)

    result = urlfetch.fetch(url+"%s" % form_data)

    logging.info(result.status_code)
    self.response.write(result.content)

                                    ⋮

app = webapp2.WSGIApplication([
  ("/send_push", SendPushHandler),
  ("/push", PushPage),
  ("/getdata", getDataPage),
  ("/response", getRespJson),
  ("/postdata", postDataPage)
], debug=True)
```

Listing 5.3: Estratto del codice Python dello script principale main.app

## 5.2 Web Application - Client

Nella componente client sono state impiegate diverse tecnologie e framework, tra cui jQuery, Materialize e WebSocket.

### 5.2.1 JavaScript, jQuery e jQuery Mobile

La componente principale dell'applicazione web è stata sviluppata in JavaScript, facendo ampio uso di una delle sue più famose librerie, jQuery. Sebbene sia fruibile anche tramite desktop, lo sviluppo si è concentrato maggiormente sulla versione mobile e quindi è stato utilizzata anche la libreria jQuery Mobile che offre funzionalità aggiuntive mirate all'ambiente mobile. La scelta di privilegiare il mobile è dettata dal fatto che attualmente non è ancora disponibile la ricerca di beacon Bluetooth in ambiente desktop; inoltre l'utilizzo dei distributori automatici, per la natura stessa del servizio, si applica maggiormente ai dispositivi mobili. Inoltre *Chrome OS* è l'unico sistema operativo desktop, in versione stabile, in cui è possibile utilizzare le funzioni offerte da Web Bluetooth che verranno affrontate nella Sezione 5.6.2

Nella Listing 5.4 è possibile vedere un estratto del codice JavaScript principale. In questo caso si vede una chiamata HTTP asincrona effettuata con la funzione ajax di jQuery all'indirizzo "/getdata". In questa richiesta vengono passati tre parametri: il tipo di dati che si vuole chiedere, in questo caso i prodotti; l'identificativo del distributore automatico per cui si stanno richiedendo; la quantità minima che deve avere un prodotto per comparire nella risposta. Questo ultimo parametro è stato aggiunto per semplificare la pagina di rifornimento del distributore, visualizzando anche i prodotti che sono esauriti e che sono solitamente presenti in quel distributore. Il Server risponderà a questa richiesta, e dopo aver interrogato il database, ritornerà un JSON contenente tutti i dati relativi ai prodotti. Nell'esempio è possibile vedere come ad ogni iterazione del ciclo *each* relativa ad ogni prodotto, verrà creato il codice HTML contenente i dati di quest'ultimo, come *result.nome* per il nome del prodotto, *result.immagine* per visualizzare l'immagine relativa o per esempio *result.posizione* che conterrà la posizione del prodotto all'interno del distributore.

```
$.ajax({
            type: 'GET',
            url: '/getdata',
            data: {
                type: 'prodotti',
                machine: theId,
                quantita: quantity
            },
            dataType: 'json'
        }).done(function(response) {
            $(response).each(function(index, result) {
                resultId.push(result.id_prodotto);
                $('#result').append('<div class="col l2 col s4"> \
                <div class="card"> \
                <a class="prodotto" \
                data-posizione=' + result.posizione + '\
                data-id=' + result.id_prodotto + ' \
                data-name="' + result.nome + '"> \
                <div class="card-image valign-wrapper"> \
                <div id="shield" data-id='+result.id_prodotto+'>\
                </div> \
                <img class="valign" src="' + result.immagine + '"> \
                </div> \
                </a> \
                                    ⋮
                </div>');
            });
                                    ⋮


        }).fail(function() {
            alert(this.responseText);
        });
```

Listing 5.4: Esempio di funzione ajax di jQuery del file JavaScript principale, adattato per questioni di spazio.

### Materialize

Materialize è un framework di front-end, basato sul Material Design di Google[1], che fornisce funzioni e strumenti per creare pagine web secondo questo stile. Questo framework, che a sua volta fa largo uso di jQuery, permette di gestire l'interfaccia da mostrare all'utente mediante l'uso di classi CSS e di funzioni.

## 5.2.2 WebSocket

Una funzione fondamentale per la piattaforma è svolta dal WebSocket, che si occupa di far dialogare tra loro, in maniera diretta, il distributore automatico e l'applicazione. WebSocket è una tecnologia web che fornisce canali di comunicazione full-duplex attraverso una singola connessione TCP. L'API del WebSocket è stata standardizzata dal W3C[2] e il protocollo WebSocket è stato standardizzato dall'IETF[3]. Nella Listing 5.5 vengono mostrate le tre funzioni che si occupano di spedire e ricevere dati dal distributore. Al momento dell'acquisto tramite l'applicazione, viene aperto un WebSocket e attraverso la funzione *send* vengono inviati al distributore i prodotti selezionati nella pagina. Quest'ultimo rimane poi in attesa di una risposta, che può essere di errore oppure può contenere il messaggio inviato dal distributore automatico. Se tutto è andato a buon fine questi dati arriveranno nella *onmessage* e verranno letti e processati per poi venire mostrati nella pagina.

```
1  var ws = new WebSocket("wss://vm.beacon-machine.eu:8080");
2
3  dataToSend = JSON.stringify({
4      prodotti: id_prodotti,
5      posizione: posizione_prodotti
6  });
7
8  ws.onopen = function() {
9      // Web Socket is connected, send data using send()
10     ws.send(dataToSend);
```

[1]Descrizione del Material Design alla pagina: http://www.google.it/design/spec/material-design/introduction.html
[2]World Wide Web Consortium
[3]Internet Engineering Task Force

```
11 };
12
13 ws.onerror = function(evt){
14     alert("Errore di comunicazione con il WebSocket del distributore
    ");
15 };
16
17 ws.onmessage = function(evt) {
18     var received_msg = evt.data;
19     product_list = JSON.parse(received_msg);
20     //process the received data
21                                     ⋮
22 }
```

Listing 5.5: Estratto del codice JavaScript che si occupa di comunicare con il distributore.

### 5.2.3 Service Workers

Un Service Worker è uno script eseguito in *background* dal browser, separatamente rispetto a una pagina web, che offre funzionalità che non richiedono la visualizzazione di una pagina o l'interazione da parte dell'utente. È un *Working Draft* del W3C[1], quindi ancora in una prima fase di specifiche e ancora lontano dall'essere una raccomandazione. Per questo le funzionalità disponibili sono in continuo aumento, ma dipendono anche dalla specifica adozione di ogni singolo browser utilizzato. Essendo questo Working Draft promosso da Google, ovviamente Chrome è il browser con la maggiore adozione delle API. Fra queste si possono trovare funzioni per intercettare le richieste di rete di una pagina e rispondere a seconda delle esigenze, per esempio con contenuti presenti in una cache o per inviare notifiche push. In futuro è prevista l'aggiunta di metodi per il *geofencing* di cui si tratterà nella Sezione 5.6.1 e per sincronizzare dati in background.

Alcune caratteristiche dei Service Worker sono:

- Sono dei *JavaScript Worker*, quindi non possono accedere direttamente al DOM[2], però possono comunicare con le pagine che li controllano attra-

[1] World Wide Web Consortium: http://www.w3.org/
[2] Document Object Model

verso dei messaggi inviati tramite l'interfaccia *postMessage* e quelle pagine possono modificare il DOM all'occorrenza.

- Sono dei proxy di rete programmabili e quindi permettono di controllare come vengono gestite le richieste di rete per quella pagina

- Vengono terminati quando non in uso e riavviati all'occorrenza, quindi non si può fare affidamento ad uno stato globale all'interno delle funzioni *onfetch* e *onmessage*.

Un Service Worker ha un ciclo di vita completamente separato rispetto alla pagina web e richiede di essere installato. Come primo passo quindi è necessario procedere alla sua registrazione, chiamando la funzione *register*, tipicamente all'interno del file JavaScript principale. Come conseguenza di questa chiamata, il browser inizierà la fase di installazione in background, durante la quale alcuni file statici vengono salvati in cache. Se tutto è andato a buon fine il Service Worker sarà installato e si passerà alla fase di attivazione dove solitamente si controllano i file in cache e vengono cancellati quelli non aggiornati. Dopo questi step il Service Worker controllerà tutte le pagine all'interno del suo *scope*, ad esclusione della pagina che ha richiesto l'installazione, che sarà controllata solo dal caricamento successivo. Una volta che il Service Worker prende il controllo, può trovarsi in due stati: può essere terminato per risparmiare memoria oppure si occuperà di gestire tutti gli eventi di richieste o messaggi che vengono fatte dalla pagina.
In questa implementazione, all'installazione del Service Worker, vengono aggiunti nella cache: la home page, i file JavaScript di jQuery, jQuery Mobile e di Materialize, i loro rispettivi CSS e le icone in stile Material Design di Google[1]. In questo modo verranno direttamente intercettate le richieste di rete per questi file e restituiti direttamente dal Service Worker. Questo processo può però esporre a rischi: se si dovessero verificare malfunzionamenti durante la richiesta di questi file, infatti, l'installazione del SW fallirebbe, senza nessun avviso o notifica, né direttamente all'utente, né nei log. Nel processo di installazione è quindi opportuno richiedere un numero ristretto di file e procedere a popolare la cache successivamente. Il Service Worker proverà comunque ad installarsi al

[1] https://www.google.com/design/icons/

successivo caricamento della pagina. Dopo il primo caricamento, salverà dinamicamente tutte le immagini dei prodotti e le restanti pagine html, in questo modo l'applicazione risulterà da subito più fluida e immediata.

### Notifiche push

All'interno del Service Worker oltre a registrare un *Listener* per le richieste di rete è possibile associarne uno ulteriore per la rilevazione di notifiche push e della loro apertura. Al momento, pure essendo l'API delle notifiche push uno standard, non è ancora del tutto completa e adottata da tutti i browser. Questa API è disponibile solo su Chrome per Android e per i vari Sistemi Operativi desktop, per il momento attraverso il Google Cloud Messaging[1], in attesa che lo standard venga ampiamente adottato. Anche Mozilla sta implementando le notifiche push all'interno di Firefox, ma è prevista la disponibilità per l'utilizzo nei Service Worker in Firefox 42, che dovrebbe essere presentato nella versione stabile a novembre 2015.

La notifica che viene inviata dal server è un JSON in cui è possibile inserire qualsiasi tipo di dato che sarà disponibile nell'evento di ricezione, ma quando si creerà la notifica nel Service Worker per visualizzarla, sarà possibile disporre di cinque soli campi:

- **Titolo**: il titolo da mostrare all'utente per la notifica
- **Corpo**: il testo del messaggio all'interno della notifica mostrato all'utente
- **Icona**: l'icona che verrà visualizzata
- **Vibrazione**: è possibile specificare un pattern di vibrazioni del dispositivo mobile da effettuare alla ricezione della notifica.
- **Tag**: un tag con cui è possibile identificare la notifica o il gruppo di notifiche. Per esempio, inviando più notifiche con lo stesso tag, come comportamento predefinito verrà visualizzata solo l'ultima e quindi tutte le altre andranno perse, perché sovrascritte. Recentemente è stata aggiunta la possibilità di richiedere le notifiche presenti attraverso la funzione *getNotifications()* e

[1] GCM: https://developers.google.com/cloud-messaging/

sarà quindi possibile per esempio raggruppare le notifiche con lo stesso tag e visualizzare un contatore.

- **Dati**: dalla versione 44 di Chrome (diventata stabile a fine luglio 2015) è stata aggiunta la possibilità di associare un campo *data* alla notifica. In questo modo è possibile passare qualsiasi tipo di stringa alla notifica e per esempio decidere quale URL mostrare al momento dell'apertura. Per motivi di sicurezza comunque sarà possibile aprire soltanto URL all'interno dello scope.

Attualmente nell'implementazione sono presenti tutti i campi presentati e all'interno del campo *data* viene passata la URL da aprire. Inoltre sono stati aggiunti nel JSON dei campi per poter inviare notifiche solo se l'utente è in determinate zone, attraverso il Geofencing, tuttavia questa funzionalità non è ancora completamente implementata (Sezione 5.6.1).

## 5.3 Applicazione Android

Allo stato attuale, le tecnologie Web utilizzate nella web application non sono ancora del tutto mature, soprattutto per quanto riguarda la componente di ricerca dei beacon, che al momento è disponibile solo in Chrome per iOS. Per quanto riguarda Android, gli sviluppatori hanno rilasciato un'applicazione di test per la ricerca dei beacon, che simula quello che verrà a breve integrato nel sistema operativo. Dal momento che è necessario installare un'applicazione per simulare una scoperta del beacon "senza app" è stato deciso di implementare un'applicazione nativa per Android così da aver maggiore controllo su questa fase e per confrontare il diverso approccio che si ha in una piattaforma più matura come quella Android. Per sfruttare tutte le API della piattaforma beacon di Google è stato utilizzato il secondo tipo di frame offerto dalla famiglia Eddystone, l'Eddystone-UID ideato proprio per l'utilizzo all'interno di un'applicazione mobile. Di seguito verranno illustrate le varie componenti della piattaforma che sono state utilizzate per l'applicazione.

### 5.3.1 Proximity Beacon API

L'API Proxymity Beacon fa parte della piattaforma per i beacon Bluetooth Low Energy (BLE) di Google. Questa API consente di gestire dei dati associati ai beacon BLE utilizzando un'architettura REST, è quindi possibile effettuare chiamate *GET*, *POST*, *PUT* e *DELETE* in base all'operazione che si vuole effettuare. Una volta registrato il beacon è possibile associare dati che verranno salvati nel cloud. In questo modo è possibile gestire e aggiornare le informazioni associate ad ogni beacon anche dopo che questi sono stati posizionati, senza bisogno di accedervi fisicamente. L'API consente inoltre di eseguire aggiornamenti batch in modo da fornire sempre le informazioni più aggiornate senza dover aggiornare manualmente ogni singolo beacon e dato che queste informazioni sono salvate nel cloud questo metodo è scalabile e a bassa latenza. Un *attachment* che è associato al beacon può essere utilizzato come segnale contestuale dall'applicazione utilizzando l'API Nearby. Nella Figura 5.2 è possibile vedere la struttura della piattaforma beacon. Nella parte destra si vede come dopo la scoperta di un beacon viene fatta la richiesta al cloud e la risposta con i dati riguardanti il beacon viene utilizzata nell'applicazione tramite l'API Nearby Messages. Nella parte sinistra si vede la registrazione e la successiva gestione del beacon. Questa ultima parte è stata implementata ad hoc tramite uno script Python.
Durante la fase di registrazione o di update del beacon è possibile associare le seguenti informazioni:

- **Advertised ID**: un oggetto contenente il tipo di beacon utilizzato e una codifica dell'identificativo effettivamente emesso del beacon, obbligatorio solo nella fase di registrazione. Il campo contenente il tipo di beacon può essere: *EDDYSTONE*: quello utilizzato in questo progetto, un formato aperto sviluppato da Google supportato sia da Android che iOS; *IBEACON*: un beacon compatibile con le specifiche di Apple iBeacon; *ALTBEACON*: un altro tipo di beacon aperto, sviluppato da Radius Networks[1]. Nella fase di update visto che verrà aggiornata direttamente la risorsa, questo campo sarà ignorato.

[2]Fonte: https://developers.google.com/beacons/overview
[1]http://www.radiusnetworks.com/

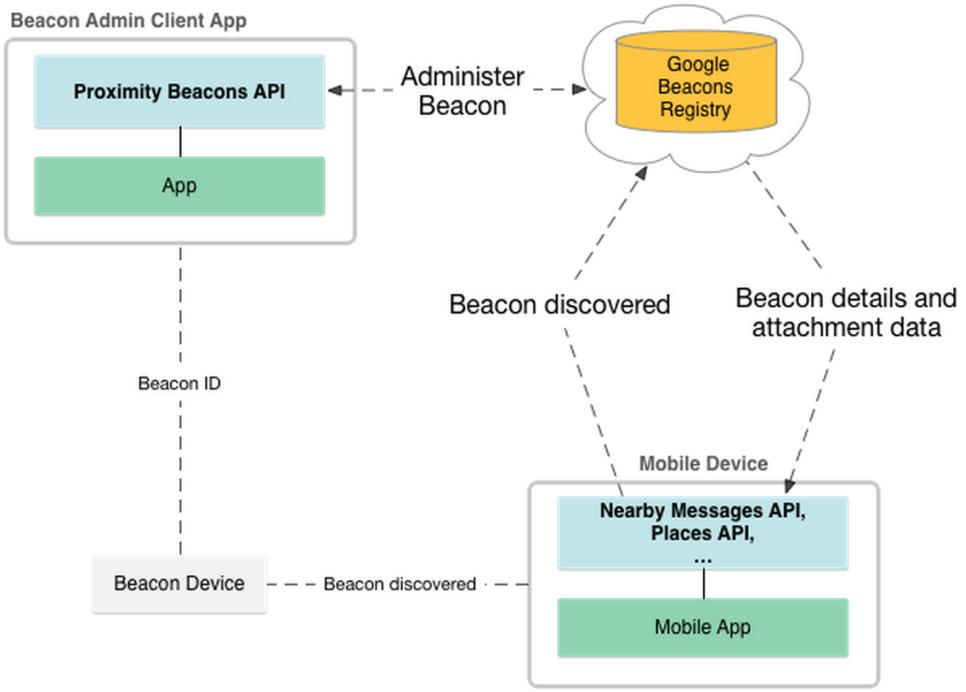



Figura 5.2: Struttura della piattaforma beacon di Google[2]

- **Status**: lo stato attuale del beacon, parametro obbligatorio. Può essere: *ACTIVE*, lo stato normale del beacon, ovvero attivo, che sta trasmettendo; *DECOMISSIONED*, il beacon non è più utilizzato; *INACTIVE*, il beacon non è più visibile a nessun dispositivo. Gli ultimi due stati sono irreversibili.

- **Expected stability**: campo opzionale in cui inserire la previsione della "stabilità" della posizione del beacon. Può essere: *STABLE*, non è previsto che si muova, per esempio all'entrata di un negozio; *PORTABLE*, indica che solitamente è stabile ma potrebbe muoversi, per esempio all'interno di un piccolo negozio; *MOBILE*, si muove frequentemente, per esempio un oggetto personale o un *Food Truck*; *ROVING*, in continuo movimento, per esempio su un autobus o un treno.

- **Latitude and longitude**: la posizione del beacon espresso con coordinate di latitudine e longitudine. È un campo facoltativo.

- **Indoor floor level**: campo opzionale, indica in quale piano di un edificio

si trova il beacon. Deve essere una stringa come restituita dalle API di Google Maps.

- **Google Places API Place ID**: anche questo un campo facoltativo dove è possibile specificare la stringa identificativa del luogo in cui è posizionato il beacon. È generata chiamando le API Google Places e richiedendo il Place ID.

- **Freeform text description**: Un campo di testo libero in cui è possibile inserire una descrizione del beacon. Sono disponibili 140 caratteri.

- **Properties**: un oggetto contenente una lista di chiavi e valori, come può essere un JSON, in cui è possibile inserire le proprietà e caratteristiche del beacon quali il livello della batteria o la versione del firmware.

Per poter utilizzare questa API è necessario abilitarla nella *Google Developers Console* e utilizzare le giuste credenziali ogni volta che si vuole registrare, aggiornare o aggiungere un *attachment* ai beacon.

### Registrazione

Nella Listing 5.6 è possibile vedere il codice utilizzato per registrare un beacon nel cloud. Nelle prime righe viene fatta una richiesta di autenticazione tramite OAuth 2.0 in cui viene fornito uno *scope*, ovvero il set limitato di permessi che vengono richiesti dallo script. La risposta con le credenziali e il token necessario per effettuare la richiesta di registrazione vengono salvati nella variabile *credentials* e successivamente il campo *access_token* viene inserito nell'header alla voce *Authorization*. Questo passaggio si ripete immutato per tutte le operazioni e quindi in seguito verrà omesso. Dopo la fase di autenticazione, viene mostrato il JSON contenente i dati relativi a un ipotetico beacon posizionato nel distributore automatico del Dipartimento di Informatica. All'interno vi sono: il campo obbligatorio che identifica il beacon, lo stato impostato come attivo, la latitudine e longitudine, la stringa del Google Place ID, la stabilità prevista e una breve descrizione. Infine viene fatta una richiesta POST contenente gli header creati e il JSON dei dati all'endpoint di registrazione salvato nella variabile URL.

```
1                                       ⋮
2 flow = OAuth2WebServerFlow(client_id=CLIENT_ID,
3                            client_secret=CLIENT_SECRET,
4                            scope="https://www.googleapis.com/auth/
    userlocation.beacon.registry",
5                            redirect_uri="http://example.com/
    auth_return")
6
7 storage = Storage("creds.data")
8 credentials = run(flow, storage)
9
10 headers = {"Content-Type": "application/http","Authorization": "
    Bearer "+credentials.access_token}
11
12 data ={
13     "advertisedId": {
14         "type": "EDDYSTONE",
15         "id": "NmVuZGluZy1tYRI0VnibrA=="
16     },
17     "status": "ACTIVE",
18     "placeId": "ChIJSdIWg7rUf0cRjRXgU18xUgw",
19     "latLng": {
20         "latitude": 44.497088,
21         "longitude": 11.356054
22     },
23     "expectedStability": "STABLE",
24     "description": "VendingMachine Dipartimento Informatica"
25 }
26
27 url = "https://proximitybeacon.googleapis.com/v1beta1/beacons:
    register''
28 r = requests.post(url, data=json.dumps(data), headers=headers)
```

Listing 5.6: Estratto del codice per la registrazione di un beacon.

In risposta a una richiesta di registrazione effettuata con successo si riceverà un JSON contenente gli stessi dati inviati in fase di registrazione e un parametro aggiuntivo *beaconName* contenente il nome assegnato al beacon. Questo parametro sarà necessario per le altre fasi di gestione.

### Aggiornamento

Tramite una richiesta di *update*, si può aggiornare un beacon precedentemente registrato, effettuando una richiesta PUT specificando come URL quello identificativo del beacon, come può essere https://proximitybeacon.googleapis.com/v1beta1/beacons/3!36656e64696e672d6d61123456789bac/. Come nel caso della registrazione si possono specificare i parametri desiderati ad eccezione del campo *status*, che anche se presente verrà ignorato perché per modificare questo parametro sono presenti endpoint specifici. Se un campo era presente nella fase di registrazione o in un update precedente e non viene specificato, verrà eliminato.

### Attivazione, disattivazione, dismissione

Come anticipato nella sezione precedente sono presenti tre endpoint specifici per attivare, disattivare o dismettere un beacon registrato. Per esempio per attivare un beacon è necessario fare una richiesta POST all'URL https://proximitybeacon.googleapis.com/v1beta1/beacons/3!36656e64696e672d6d61123456789bac:activate. Per effettuare le altre due operazioni basterà sostituire *activate* con *deactivate* o *decommission* in base al tipo di richiesta che si vuole effettuare. Se viene utilizzato decommission, il beacon non sarà più utilizzabile in futuro con quell'identificativo.

### Aggiunta Attachment

Una richiesta per creare un attachment deve contenere due campi:

- *namespacedType*: composto da il *namespace* ovvero il nome assegnato al progetto sul cloud, non modificabile; *type* che identifica il tipo di dati dell'attachment e può essere una stringa di caratteri scelta liberamente ad eccezione del carattere "/" e essere lungo fino a 100 caratteri.

- *data*: il contenuto dell'attachment, ovvero i dati che si vogliono associare al beacon. Deve essere codificato in *base64*[1] e può essere fino a 1024 bytes. È possibile associare qualsiasi tipo di dato purchè rispetti queste specifiche.

[1] http://tools.ietf.org/html/rfc4648#section-4

Nella Listing 5.7 è possibile vedere il codice utilizzato per creare un attachment per il beacon registrato nella sezione precedente 5.3.1. Dopo aver effettuato l'autenticazione qui omessa, viene creato un JSON contenente l'URL del distributore, un titolo, una descrizione, la posizione descritta a parole e quella tramite coordinate GPS e infine un ipotetico stato delle promozioni per quel distributore. Successivamente viene codificato il dato e inserito nell'apposito campo del JSON finale. Questa volta nell'URL di richiesta è inserito il nome del beacon ricevuto come risposta in fase di registrazione *beacons/3!36656e64696e672d6d61123456789bac* e fatta una richiesta POST all'endpoint attachment.

```
1                                   ⋮
2  dataToEncode = {
3    "url": "https://vending-machine.appspot.com/?id=3",
4    "title": "Vending Machine",
5    "description": "Distributore automatico",
6    "position": "Dipartimento di Informatica, piano -1",
7    "latLng": {"latitude":44.497088,"longitude":11.356054,
8    "promotion": "Promozione 2x1 in corso!"}
9    }
10
11 dataEncoded = base64.b64encode(str(dataen))
12
13 data =    {
14   "namespacedType":"vending-machine/id",
15   "data": dataEncoded
16 }
17 url = "https://proximitybeacon.googleapis.com/v1beta1/beacons
       /3!36656e64696e672d6d61123456789bac/attachments"
18 r = requests.post(url, data=json.dumps(data), headers=headers)
```

Listing 5.7: Estratto del codice per la creazione di un attachment di un beacon.

Se la richiesta è andata a buon fine si riceverà come risposta un JSON contenente gli stessi dati inviati con in aggiunta il campo *attachmentName* contenente il nome assegnato al beacon.

#### Eliminazione Attachment

Per quanto riguarda gli attachment non esiste la possibilità di modifica; di conseguenza nel caso si volessero aggiornare i dati associati a un determinato *type* (descritto nella sezione precedente) oppure eliminarli, è possibile effettuare una richiesta DELETE all'URL contenente l'identificativo dell'attachment restituito in fase di creazione per eliminarlo e eventualmente ricrearlo successivamente con i nuovi dati. Per esempio per eliminare l'attachment creato nella sezione precedente sarà necessario effettuare una richiesta DELETE a https://proximitybeacon.googleapis.com/v1beta1/beacons/3!36656e64696e672d6d61123456789bac/attachments/ddb71ff1−5d9f−491d−b3c1−69bb108d915a.

#### Lista beacon e attachment

È possibile richiedere sia una lista dei beacon registrati che la lista dei singoli attachment per ogni beacon. Per effettuare queste due richieste è necessario fare una richiesta GET a https://proximitybeacon.googleapis.com/v1beta1/beacons e https://proximitybeacon.googleapis.com/v1beta1/beacons/3!36656e64696e672d6d61123456789bac/attachments/ rispettivamente. Nella seconda richiesta viene specificato il beacon, in questo caso restituirà gli attachment rispettivi al beacon registrato nelle sezioni precedenti.

Per tutte queste operazioni, anche dove non è stato specificato espressamente, sarà sempre necessario autenticarsi. Nel caso dell'applicazione mobile l'autenticazione avviene tramite una *API Key* inserita all'interno del codice.

### 5.3.2 Nearby Messages API

L'API Nearby permette di far interagire tra loro dispositivi diversi che si trovano nelle vicinanze. Questi dispositivi devono essere connessi a Internet, ma non necessariamente nella stessa rete. Neaby sfrutta una combinazione di Bluetooth, Bluetooth Low Energy, Wi-Fi e ultrasuoni per comunicare un codice di accoppiamento univoco fra due dispositivi. Il server facilita lo scambio di messaggi tra i dispositivi che rilevano lo stesso codice. La sequenza seguente mostra gli eventi che portano a uno scambio di messaggi:

1. Un'applicazione pubblica una richiesta di associare un payload binario (il messaggio) con un codice univoco (token). Di conseguenza il server crea un'associazione temporanea tra il messaggio e il token.

2. Il dispositivo che pubblica questo messaggio, utilizza una combinazione di Bluetooth, Bluetooth Low Energy, Wi-Fi e ultrasuoni per rendere il token rilevabile dai dispositivi nelle vicinanze. Inoltre, questo dispositivo, utilizza le stesse tecnologie per fare una scansione di token in arrivo da altri dispositivi.

3. Un'applicazione che ha fatto una sottoscrizione, la associa con un token e utilizza le stesse tecnologie appena descritte per inviare il proprio token al dispositivo che pubblica e per rilevare il token inviato dall'altro dispositivo.

4. Quando una delle due parti rileva il token dell'altro dispositivo, lo invia al server.

5. Il server facilita lo scambio di messaggi fra i due dispositivi quando sono entrambi associati con un token comune e le chiavi dell'API utilizzata dalle applicazioni sono associate allo stesso progetto nella Google Developers Console.

Si tratta quindi di un tipo di API *publish-subscribe* che permette di trasferire piccoli *payload* binari tra dispositivi connessi a Internet. Questo vuol dire che un dispositivo pubblica un messaggio (publish) e l'altro effettua una sottoscrizione (subscribe) dicendo che è in attesa di ricevere messaggi. I beacon sono un caso particolare di questo meccanismo ed è possibile specificare la strategia Strategy.BLE_ONLY, al momento della sottoscrizione nell'app come mostrato nella Listing 5.8, per indicare che si vuole effettuare solo una scansione di dispositivi BLE. In questo modo l'API non attiverà una scansione Wi-Fi o Bluetooth normale, ma solo quella Bluetooth Low Energy e questo permette di ridurre notevolmente i tempi di scansione ed il consumo di batteria. La parte di *publish* viene effettuata attraverso la Proximity Beacon API, come descritto nella Sezione 5.3.1, mentre è possibile vedere la parte di *subscribe* nella Listing 5.8. Per motivi di spazio in questa Listing viene mostrato solo lo scheletro delle funzioni e in seguito nella

Listing 5.9 verranno illustrate le azioni intraprese al momento della ricezione di un messaggio, ovvero le operazioni all'interno della funzione onFound. La prima riga mostra la creazione di un filtro, in cui viene specificato che si vogliono ricevere solamente i messaggi provenienti dal namespace "vending-machine" con tipo "id". Visto che è possibile associare diversi attachment con tipi diversi, in questo modo è possibile ricevere solo quelli rilevanti per l'applicazione. Nelle righe successive viene creato il messageListener in cui è possibile compiere operazioni al momento della ricezione di un messaggio o quando questo verrà perso, ovvero non ci si troverà più all'interno del campo di emissione del beacon. Come ultima cosa viene mostrata la sottoscrizione in cui sono specificati il filtro e il listener appena creati e la strategia da utilizzare.

```
1                                   ⋮
2 mMessageFilter = new MessageFilter.Builder().includeNamespacedType("
    vending-machine", "id").build();
3 messageListener = new MessageListener() {
4     // Called each time a new message is discovered nearby.
5     @Override
6     public void onFound(Message message) {
7         //message found
8         //read it
9     }
10    public void onLost(Message message) {
11        //message lost
12    }
13
14                                  ⋮
15
16 Nearby.Messages.subscribe(mGoogleApiClient, messageListener,
    Strategy.BLE_ONLY, mMessageFilter)
17 .setResultCallback(new ErrorCheckingCallback("subscribe()"));
```

Listing 5.8: Estratto del codice che effettua la sottoscrizione ai messaggi Neraby

La Listing 5.9 mostra parte del contenuto della funzione onFound che viene chiamata al momento della ricezione di un messaggio. Dopo aver effettuato un controllo per accertarsi di leggere solamente la corretta tipologia di dati, viene copiato il contenuto del messaggio in un oggetto JSON *jMess*. Successivamente vengo-

no letti i vari valori contenuti nel JSON ricevuto e inseriti negli appositi campi dell'oggetto "beacon" creato appositamente per gestire le informazioni relative a questi. Per esempio sono presenti la posizione del distributore, la sua descrizione, il titolo del messaggio da visualizzare, la promozione in corso e l'URL relativa al distributore. Questo oggetto beacon verrà utilizzato dalle varie componenti dell'applicazione, per esempio per visualizzare i dati relativi al distributore automatico da cui si è ricevuto il messaggio. I dati relativi alla posizione del distributore non vengono al momento utilizzati, si veda la Sezione 5.7.

```
1  if(message.getType().equals("id")) {
2      byte[] data = message.getContent();
3      jMess = new JSONObject(new String(data));
4      jLatLng = new JSONObject();
5      BeaconData beacon = new BeaconData();
6
7      if (jMess != null) {
8          try {
9              beacon.setPosition(jMess.getString("position"));
10             beacon.setDescription(jMess.getString("description"));
11             beacon.setPromotion(jMess.getString("promotion"));
12             beacon.setURL(jMess.getString("url"));
13             beacon.setTitle(jMess.getString("title"));
14             jLatLng = jMess.getJSONObject("latLng");
15             beacon.setLatitude(jLatLng.getDouble("latitude"));
16             beacon.setLongitude(jLatLng.getDouble("longitude"));
17         }
18     }
19 }
```

Listing 5.9: Estratto del codice che effettua la sottoscrizione ai messaggi Neraby

La prima volta che viene aperta l'applicazione, Nearby chiederà il permesso di poter attivare il Wi-Fi e Bluetooth automaticamente per la fase di ricerca, come mostrato in Figura 5.3. In questo modo anche se l'utente avrà il Bluetooth disattivato, quando inizierà una scansione, questo verrà attivato automaticamente, senza effettuare nessuna richiesta aggiuntiva, per tutta la durata di questa fase e disattivato non appena si passerà a una diversa attività all'interno dell'applicazione o se questa verrà messa in background.

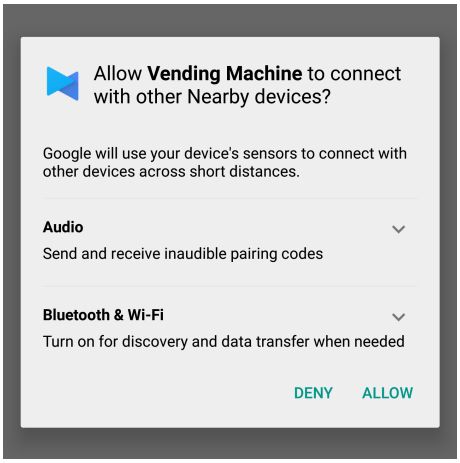



Figura 5.3: Neaby: richiesta di utilizzo Bluetooth

### 5.3.3 Geofencing API

Per facilitare l'individuazione dei distributori automatici, all'apertura dell'applicazione viene effettuata una richiesta al server di tutti i distributori e viene aggiornata una lista di *Geofence*. In questo modo è possibile impostare delle aree attorno al distributore automatico, individuare automaticamente quando un utente entra in una di queste tramite l'API Geofence ed effettuare operazioni di conseguenza. La richiesta della lista delle geofence viene fatta all'endpoint "/position", che effettua una chiamata al database e risponde con un JSON. Nella Listing 5.10 è possibile vedere le operazioni che vengono effettuate una volta ricevuta questa lista, contenuta nell'oggetto resp. Nel database dei distributori, le posizioni vengono salvate con l'indirizzo e quindi viene utilizzata l'API Geocoding per trasformare questi indirizzi in coordinate GPS, necessarie per impostare una geofence. È possibile rilevare quest'ultimo aspetto nelle prime righe del codice, dove viene creata una nuova istanza Geocoder e viene utilizzata per richiedere le coordinate GPS attraverso la sua funzione getFromLocationName a cui viene passato l'indirizzo contenuto nell'oggetto appena ricevuto resp, restituito tramite la richiesta della stringa "posizione". Questa stringa è ad esempio "Mura Anteo Zamboni, 7, 40126 Bologna" per il distributore relativo al Dipartimento di Informatica. Se sono state ricevute delle coordinate GPS corrispondenti, si può procedere alla creazione della geofence e all'inserimento di questa nella lista comprendente tutte

le geofence che si vogliono creare. Per ognuna di queste viene creato un nuovo oggetto Geofence in cui vengono impostati:

- **RequestId**: utilizzato per identificare la geofence, utile per quando non sarà più necessaria e si vorrà procedere all'eliminazione
- **CircularRegion**: attraverso i parametri *latitue*, *longitude* e *GEOFENCE_RADIUS_IN_METERS*, viene impostata l'area da "osservare". I primi due valori sono quelli ricevuti attraverso la risoluzione dell'indirizzo, mentre l'ultimo è definito all'interno dell'applicazione e ha un valore di 100 metri. Questo valore è stato impostato secondo le linee guida dell'API che suggeriscono di utilizzare un'area di queste dimensioni perché la localizzazione essendo calcolata da più sorgenti, ovvero Wi-Fi, rete cellulare e GPS in base alle situazioni, potrebbe non essere sempre altamente precisa e quindi con questo valore si evitano la maggior parte degli errori.
- **ExpirationDuration**: con questo valore si indica per quanto tempo la geofence deve rimanere attiva. Nell'applicazione è stato impostato un valore di 48 ore, perché comunque queste verranno aggiornate ad ogni apertura dell'applicazione.
- **TransitionTypes**: è possibile impostare a quali eventi di "transizione" si è interessati, in questo caso l'entrata e l'uscita da una geofence.

Nell'ultima riga gli identificativi delle geofence vengono salvati permanentemente, in questo modo è possibile gestirle anche se l'applicazione viene chiusa e se necessario eliminarle alla riapertura dell'applicazione o aggiornarle con i valori nuovi ricevuti dal server.

```
1                                   ⋮
2 gcoder = new Geocoder(getActivity(), Locale.getDefault());
3 List<Address> address;
4
5 address = gcoder.getFromLocationName(resp.getString("posizione"),1);
6
7 if(address.size() > 0) {
8     double latitude = addresses.get(0).getLatitude();
9     double longitude = addresses.get(0).getLongitude();
```

```
10
11      mGeofenceList.add(new Geofence.Builder()
12              .setRequestId(obj.getString("nome_zona"))
13              .setCircularRegion(
14                      latitude,
15                      longitude,
16                      GEOFENCE_RADIUS_IN_METERS
17              )
18              .setExpirationDuration(GEOFENCE_EXPIRATION_IN_MILLISECONDS)
19              .setTransitionTypes(Geofence.GEOFENCE_TRANSITION_ENTER |
20                      Geofence.GEOFENCE_TRANSITION_EXIT)
21              .build());
22      mGeofences.put(obj.getString("nome_zona"));
23  }
24                                          ⋮
```

Listing 5.10: Estratto del codice che crea una Geofence

## 5.4 Distributore Automatico

Per simulare l'attività del distributore automatico, è stato utilizzato un Raspberry Pi nel quale è inserita una chiavetta Bluetooth 4.0 che si occupa di trasmettere i due tipi di beacon: Eddystone-URL e Eddystone-UID. Nel dettaglio sono attivi due servizi: uno script in Python che implementa un WebSocket e uno script in Node.js che si occupa dell'emissione dei beacon.

### 5.4.1 Raspberry Pi

Il Raspberry Pi è un *single-board* computer economico, ma completo, ideale per lo sviluppo. Per questo progetto è stato utilizzato un Raspberry Pi 2, versione B, che ha un processore ARM Cortex-A7. Dato che l'architettura ARMv7 è supportata ufficialmente da Linux, sono disponibili diverse distribuzioni Linux, al contrario del primo modello del Raspberry Pi che doveva adottare una versione modificata di Debian. È stata quindi installata una versione di Debian minimale, con solo le librerie necessarie per il WebSocket e lo stack Bluetooth, per renderlo il più performante possibile.

### 5.4.2 WebSocket

Il WebSocket, una volta ricevuta una richiesta dal client, decide in base a una probabilità prefissata se erogare il prodotto o rifiutare la transazione. Questo è stato previsto per simulare un guasto nel motore del distributore che eroga il prodotto o per simulare un prodotto che si incastra e non cade. Come per tutto il resto della piattaforma, anche in questo caso, il WebSocket deve avere un certificato SSL valido per poter essere chiamato dal client, quindi è stato creato un certificato personale attraverso il servizio StartSSL.[11] Nella Listing 5.11 viene mostrata una prima implementazione in cui è presente la funzione che si occupa di decidere se erogare o meno il prodotto, *decision()* e quella principale che risponde alle richieste inviate dal client, *vending()*, che in base alla decisione risponde con "1", se il prodotto è stato erogato o "0" altrimenti. In questo caso la complessità del software eseguito dal distributore è minima e quindi si limitano le possibilità di errori e anche la quantità di dati che deve trasferire è molto ridotta. Con questa implementazione però si lascia il compito di aggiornare il database sul server al client che sta effettuando l'acquisto in quel momento e quindi si ha un controllo limitato sull'esito dell'operazione, in quanto si potrebbero verificare, ad esempio, problemi di connessione oppure l'utente potrebbe effettuare una chiusura forzata dell'applicazione e lasciare il database non sincronizzato con l'effettiva disponibilità del distributore.

```
1  def decision():
2    num = random.random()
3    return num < 0.85
4
5  @asyncio.coroutine
6  def vending(websocket, path):
7    response = {}
8    name = yield from websocket.recv()
9    product_list = json.loads(name)
10   prodotti = product_list["prodotti"]
11   posizioni = product_list["posizione"]
12
13   for key, item in enumerate(prodotti):
14     if decision():
15       #prodotto erogato
```

```
16          response.setdefault(str(item),1)
17      else:
18        #prodotto NON erogato
19          response.setdefault(str(item),0)
20
21    yield from websocket.send(json.dumps(response))
```

Listing 5.11: Estratto del codice Python del servizio WebSocket nel distributore

Per questo lo script è stato rivisto aggiungendo la chiamata diretta al server da parte del distributore. Nella Listing 5.12 si possono vedere le differenze rispetto alla prima implementazione. Come prima cosa viene creata una lista di prodotti che verranno inviati al server e successivamente viene popolata dai prodotti erogati. Nel caso il prodotto non venga erogato viene creato un dizionario che sarà restituito al client. Se sono stati erogati dei prodotti, viene creato il JSON da inviare al database sul server, contenente tre parametri:

- **upload_type**: il tipo di operazione che si vuole effettuare, in questo caso un aggiornamento delle quantità dei prodotti
- **machine**: l'identificativo del distributore sul quale si vuole effettuare l'operazione
- **id_prodotto**: una lista dei prodotti da aggiornare

Viene quindi effettuata una chiamata POST che ha come parametri l'URL dell'endpoint, specificato all'inizio di questa Listing e la struttura dati appena creata. Come risposta si riceverà un JSON contenente la lista dei prodotti appena inviati e le rispettive quantità aggiornate. Questa lista viene unita a quella dei prodotti non erogati, che per essere identificati avranno una quantità di "-1". Se invece sono presenti solo prodotti non erogati verranno copiati direttamente nella variabile da inviare al client. L'ultima riga, come nella Listing precedente mostra la risposta che viene inviata al client alla fine di tutto questo processo.

```
1 prodottiServer = []
2 url = "https://vending-machine.appspot.com/postdata"
3
4 for key, item in enumerate(prodotti):
```

```
5    if decision():
6      #prodotto erogato
7      prodottiServer.append(item)
8    else:
9      #prodotto NON erogato
10     response.setdefault(str(item),-1)
11
12 if prodottiServer:
13   data = {"upload_type": "update",
14           "machine": idMachine,
15           "id_prodotto[]": prodottiServer}
16   serverResp = requests.post(url, data=data)
17   ServerJson = json.loads(serverResp.text)
18   finalResult = serverJson.copy()
19   finalResult.update(response)
20 else:
21   finalResult = response
22
23 yield from websocket.send(json.dumps(finalResult))
```

Listing 5.12: Estratto del codice Python del servizio WebSocket nel distributore, con aggiornamento diretto del server

Andrebbe valutato, soprattutto in base alla connessione a Internet presente sul distributore, ma anche all'hardware disponibile, quale soluzione è opportuno adottare.

### 5.4.3 Broadcast del beacon

Per la componente che si occupa di creare i beacon è stato scelto di realizzare lo script in Node.js in quanto sono disponibili delle librerie per interagire con il Bluetooth e creare i vari parametri di advertising dell'Eddystone-URL. Queste sono facilmente utilizzabili nel Raspberry Pi, infatti è sufficiente utilizzare il gestore dei pacchetti *npm*, attraverso il comando npm install eddystone−beacon per installare la libreria necessaria per trasmettere il segnale, che a sua volta si occuperà di installare tutte le dipendenze necessarie.[6] Come si può vedere nella Listing 5.13, nello script sarà necessario richiedere la libreria e poi utilizzare la funzione per trasmettere l'URL.

```
var eddystoneBeacon = require('eddystone-beacon');

var options = {
  txPowerLevel: -22  // override TX Power Level, default value is -21,
  tlmCount: 2,       // 2 TLM frames
  tlmPeriod: 10      // every 10 advertisements
};

url = 'https://goo.gl/WVBu4W'
eddystoneBeacon.advertiseUrl(url, [options]);
```

Listing 5.13: Codice di esempio che trasmette il beacon URL

```
var namespaceId = '36656e64696e672d6d61';
var instanceId = '123456789bac';

eddystoneBeacon.advertiseUid(namespaceId, instanceId, [options]);
```

Listing 5.14: Codice di esempio che trasmette il beacon UID

La funzione *advertiseUrl* accetta come input due parametri: l'URL da trasmettere e le opzioni. Il primo parametro delle opzioni riguarda il livello di potenza della trasmissione, di default a -21. Il secondo e il terzo parametro servono per impostare rispettivamente quanti frame TLM inviare e ogni quanto. Questi frame, che come già accennato sono di diagnostica del beacon, vengono intervallati a quelli che trasmettono l'URL e in questo caso si hanno 2 frame TLM ogni 10 frame URL. Nell'implementazione sono stati utilizzati i valori di default, quindi l'ultima riga diventerà eddystoneBeacon.advertiseUrl("https://goo.gl/WVBu4W");. In modo simile è possibile creare un broadcast per l'Eddystone-UID utilizzato all'interno dell'applicazione Android. Nella Listing 5.14 è possibile vedere la parte di codice che si occupa di effettuare questa operazioni. A differenza del beacon Eddystone-URL, per quello UID è necessario specificare degli identificativi per il namespace e per l'istanza, che in questo caso sono rispettivamente un *hash* del dominio utilizzato per il sito web e una stringa casuale. Questi vengono passati come parametri alla funzione adverdiseUid,che si occuperà di creare il broadcast dell'UID, aggiungendo se necessario delle opzioni come per la Listing 5.13.

## 5.5 Utilizzo delle applicazioni - Casi d'uso

In questa sezione verranno illustrati i casi d'uso per l'applicazione web e quella nativa e descritte le loro varie componenti. Le motivazioni principali per lo sviluppo di queste applicazioni sono quelle di poter scoprire gli oggetti connessi a Internet e di poter interagire con esse.

### 5.5.1 Scoperta di un beacon

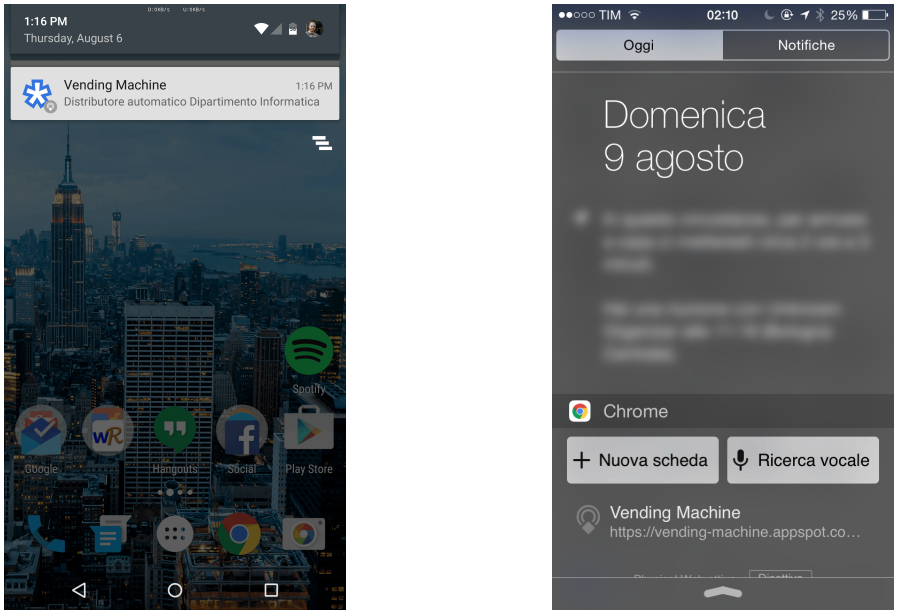
Questa fase utilizza due approcci diversi. L'applicazione nativa deve essere aperta e iniziata una fase di scansione, dal momento che non è possibile utilizzare le API Nearby Message quando l'applicazione è in background, ma è stato dato per scontato che se l'utente ha installato l'applicazione è a conoscenza della possibilità di individuare distributori automatici e quindi la aprirà nel momento del bisogno. Per l'applicazione web, sia utilizzando l'applicazione di test per Android, che la funzionalità già integrata di iOS, si vedrà comparire un messaggio all'interno dell'area notifiche.

#### Applicazione web

L'utente riceve una notifica nel menu a tendina del proprio dispositivo mobile solo quando lo schermo viene acceso e viene aperto il centro notifiche nelle vicinanze di un distributore, ovvero di un beacon Bluetooth, come mostrato in Figura 5.4a. Nel caso di iOS verrà visualizzato come mostrato in Figura 5.4b all'interno della sezione Chrome. Ogni beacon ha la URL che punta al contenuto di quel distributore, quindi aprendo la notifica si viene indirizzati alla pagina con la lista di tutti i prodotti presenti (Figura 5.8a).

#### Applicazione Android

All'apertura dell'applicazione, come mostrato in Figura 5.5a, viene richiesto di effettuare uno *swipe* per iniziare lo scan dei beacon; questo perché una delle *best practice* suggerite per l'uso dell'API è quello di non iniziare operazioni di ricerca se non espressamente richiesto dall'utente. Effettuando lo swipe si inizia la ricerca, mostrata in Figura 5.5b e non appena viene individuato il primo beacon si



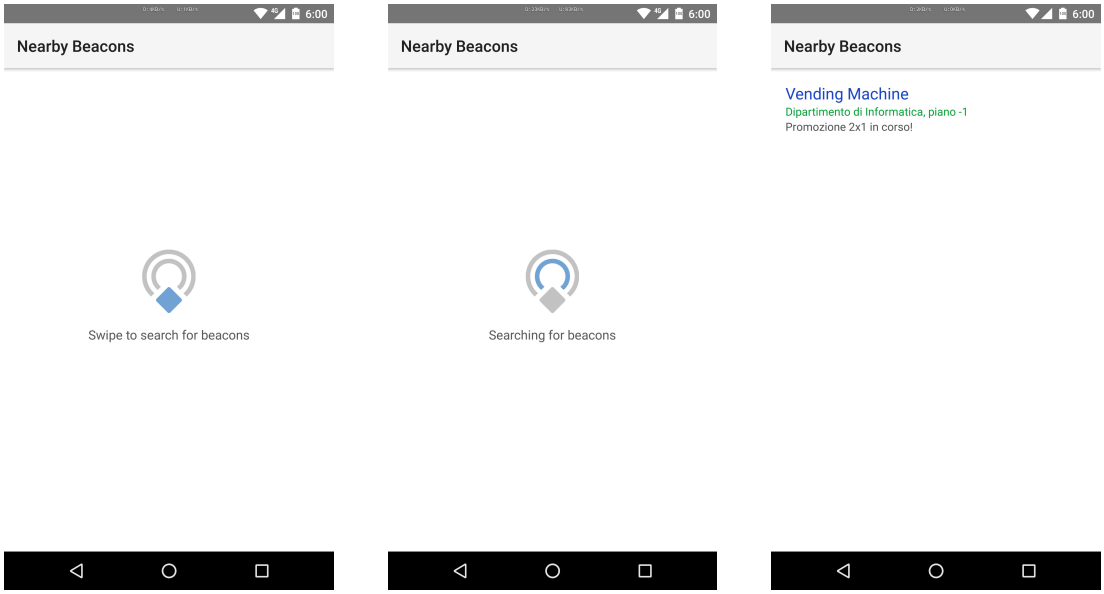

(a) Web app Android (b) Web app iPhone

Figura 5.4: Scoperta di un beacon in Android e iOS

passa alla schermata in Figura 5.5c, dove vengono popolati i vari campi presenti nel messaggio ricevuto. In questo caso il messaggio era quello dell'attachment registrato nella Listing 5.7. Nell'applicazione Android si ha un maggior controllo su quali elementi visualizzare quando si trova un beacon rispetto alla soluzione dell'applicazione web, che richiede automaticamente solo la stringa contenuta nel *manifest* del sito web. Infatti è possibile associare qualsiasi tipo di dato al beacon e la visualizzazione è completamente gestita dall'applicazione.

### 5.5.2 Ricezione di una notifica push

Un altro caso d'uso si ha quando l'utente riceve una notifica push. Nella Figura 5.6 è possibile vederne la ricezione per la web app. All'utente verranno mostrati i campi Titolo e Corpo, come descritti in 5.2.3, completamente personalizzabili. In questo caso non sarà possibile avere dei criteri in base a cui riceverle, in quanto non sono ancora disponibili meccanismi per creare delle geofence (Sezione 5.6.1) e quindi non è possibile sapere se l'utente si trova nelle vicinanze di un distributore. Sarà quindi necessario utilizzare un'apposita pagina da far visualizzare all'utente




(a) Schermata iniziale (b) Ricerca del beacon (c) Beacon trovato

Figura 5.5: Scoperta di un beacon nell'applicazione Android

quando apre la notifica, contenente per esempio la lista delle promozioni in corso e in cui si specifica che si deve ricercare il distributore. In futuro come spiegato nella Sezione 5.6.2 sarà possibile inoltre controllare che l'utente sia effettivamente davanti al distributore facendo una scansione direttamente all'interno della pagina.

### 5.5.3 Entrata in una Geofence

L'utilizzo delle geofence è al momento possibile soltanto nell'applicazione Android. L'informazione dell'entrata in una geofence viene utilizzata per visualizzare una notifica sulla presenza di un distributore nelle vicinanze e l'utente viene invitato ad aprire l'applicazione per avere maggiori informazioni. Nella Figura 5.7 viene mostrato un esempio di notifica. Aprendo la notifica si arriverà alla vista della ricerca del beacon così da essere sicuri che l'utente sia abbastanza vicino da individuare il distributore automatico.

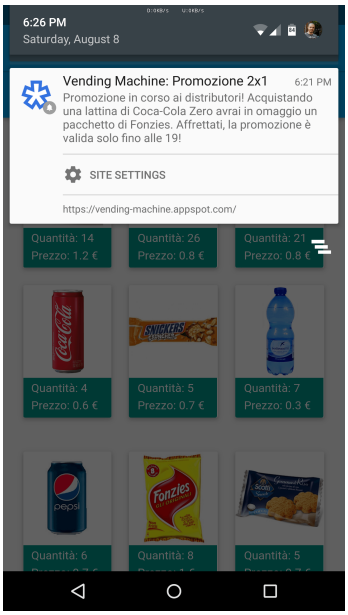



Figura 5.6: Notifica push Web Application

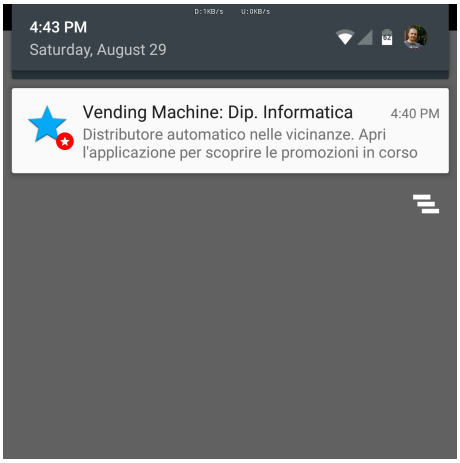
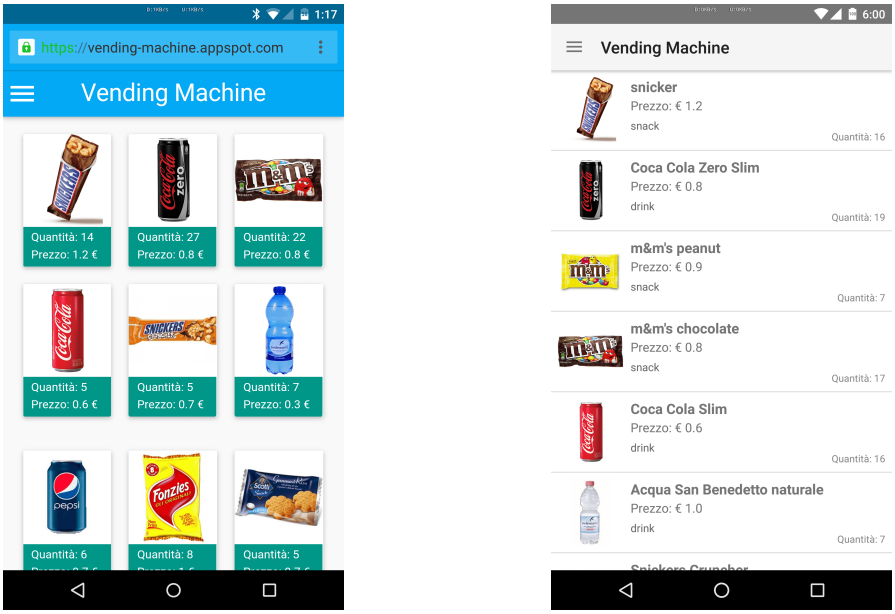



Figura 5.7: Esempio di notifica all'entrata in una Geofence




(a) Vista principale Web Application (b) Vista principale in Android

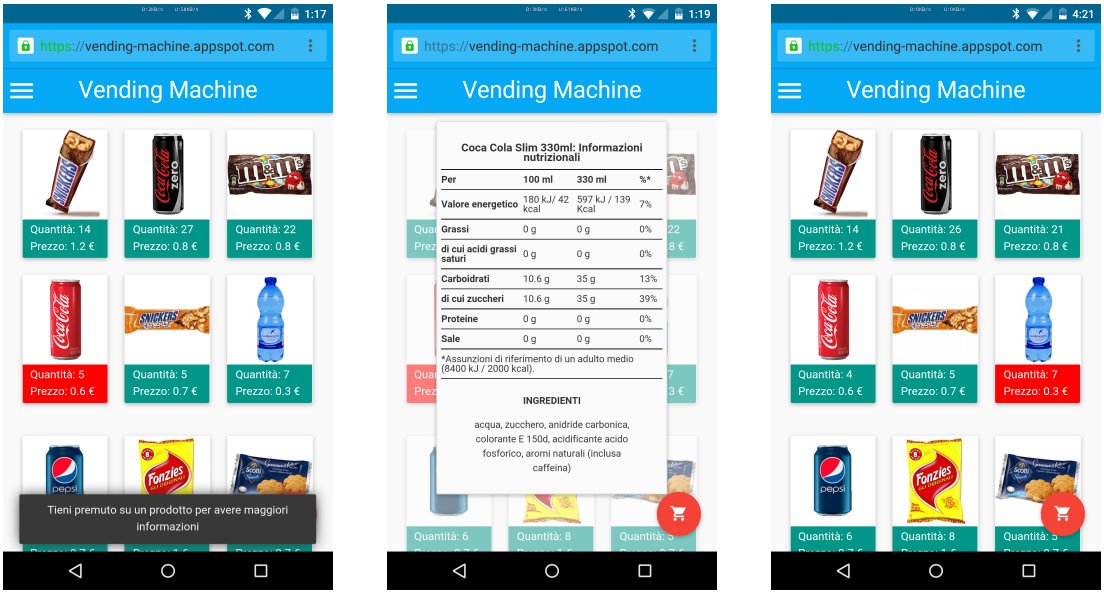
Figura 5.8: Home per Android e Web Application

### 5.5.4 Apertura dell'applicazione

In tutti i casi precedentemente descritti, ad eccezione del caso delle notifiche push per la web application, come risultato si arriva all'apertura dell'applicazione in cui vengono mostrati i prodotti disponibili per quel distributore. Quindi dopo aver trovato un beacon, aver ricevuto una notifica o essere entrati in una geofence, si vedrà la pagina principale mostrata in Figura 5.8a per la web application e 5.8b per l'applicazione Android.

#### Applicazione web

Per quanto riguarda l'applicazione web, nella Figura 5.9a viene visualizzato un esempio di messaggio che può essere mostrato all'utente, un toast, un piccolo pop-up per dare un feedback o fornire informazioni. In questo caso viene informato l'utente che tenendo premuto su un prodotto potrà visualizzare le informazioni nutrizionali e gli ingredienti di quest'ultimo. Questa azione viene mostrata in Figura 5.9b, dove viene creato un pop-up contenente tutti i dati relativi al prodotto. In Figura 5.9c è possibile invece vedere come facendo un "tap" su uno o più




(a) Esempio di toast alla prima apertura

(b) Informazioni relative al prodotto

(c) Bottone carrello per comprare

Figura 5.9: Insieme di immagini dell'applicazione web

prodotti vengono selezionati e compare un bottone con cui è possibile finalizzare l'acquisto.

Nella Figura 5.11e viene visualizzata la richiesta di aggiungere la pagina alla homescreen. Questa richiesta viene gestita automaticamente dal browser, se sono presenti i campi all'interno del *manifest* della pagina, e viene proposta dopo che la pagina viene visitata per almeno due volte in un certo intervallo di tempo. Ovviamente sarà sempre possibile aggiungere la pagina manualmente alla homescreen dal menu di Chrome, con lo stesso risultato, ma in questo modo si pone l'utente a conoscenza della possibilità ottenendo un maggior coinvolgimento. Se si decide di aggiungere la pagina alla homescreen, come mostrato in 5.11f viene creata un'icona come se fosse un'applicazione nativa e una volta aperta si ha l'impressione che questa lo sia, in quanto non compare neanche più la barra di navigazione del browser. La pagina che verrà aperta da quell'icona è impostata lato server all'interno del manifest, normalmente si apre la home page, ma non essendo rilevante in questo contesto perché dipendente dal beacon nelle vicinanze, è possibile aprire per esempio una pagina delle promozioni o in futuro una ricer-

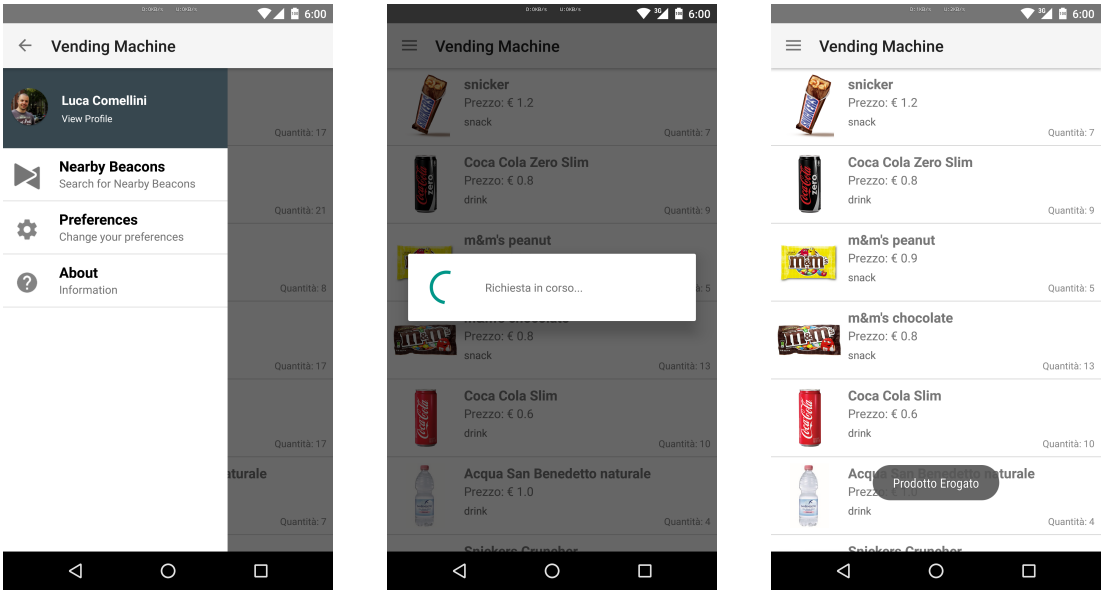
ca dei beacon, quando le API del Web Bluetooth saranno completate (Sezione 5.6.2).

#### Applicazione Android

In modo simile all'applicazione web, è presente un menu che si può aprire sia effettuando uno *swipe* dall'estremità sinistra del display oppure premendo l'*hamburger* menu in alto a sinistra. Nella Figura 5.10a è possibile vedere il menu dell'applicazione aperto. La prima parte mostra un ipotetico profilo con cui si è autenticati. La seconda opzione "Nearby Beacons" riporta alla schermata iniziale di ricerca del beacon vista in 5.5a, dove è possibile cercare i distributori automatici nelle vicinanze. La voce preferenze non è al momento utilizzata, le possibili implementazioni verranno affrontate nella Sezione 5.7.2. L'ultima voce porta a una pagina di informazioni generali sull'applicazione. Successivamente viene mostrato un esempio di come vengono visualizzate le varie richieste mentre vengono eseguite nella Figura 5.10b e di messaggio di avvenuta erogazione del prodotto nella Figura 5.10c. In modo simile questi messaggi *toast* vengono utilizzati anche per gli eventuali errori, per esempio alla mancata erogazione del prodotto. Inoltre effettuando uno swipe dall'alto verso il basso è possibile aggiornare manualmente la lista dei prodotti.

### 5.5.5 Rifornimento distributore

Questa parte è stata implementata solamente nella web application, principalmente perché non rilevante per la *user experience* degli utenti, che viene paragonata fra le due implementazioni. Nella 5.11a è possibile vedere il menu laterale che compare toccando l'*hamburger* menu oppure facendo uno *swipe* dall'estremità sinistra del display. Qui è possibile vedere le varie pagine presenti per rifornire il distributore o caricare nuovi prodotti nel database, un *toggle* che mostra se le notifiche push sono attive o meno e un link per il login (si veda 5.6.3). Nella Figura 5.11b viene mostrata la prima pagina per rifornire il distributore, che mostra i prodotti che erano presenti ma esauriti in modo da poter rifornire velocemente, visto che raramente vengono cambiati i prodotti all'interno di un distributore, e i prodotti con ancora una disponibilità per poterne aumentare il numero. Premen-




(a) Menu delle opzioni per Android

(b) Esempio di richiesta in corso

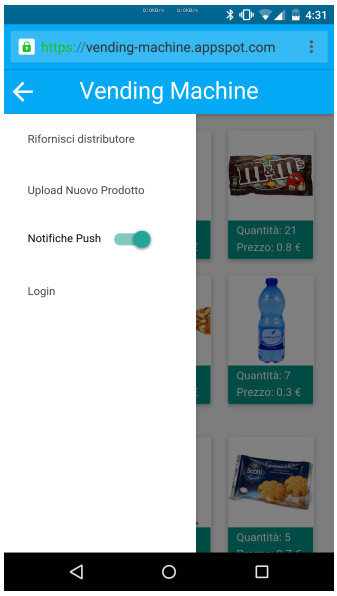
(c) Esempio di toast all'erogazione

Figura 5.10: Insieme di immagini dell'applicazione Android

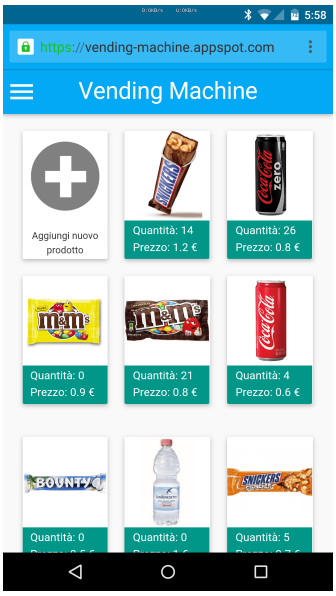
do l'immagine a forma di "+" è possibile aggiungere prodotti che non sono mai stati inseriti all'interno di quel distributore e viene aperta una pagina come quella della Figura 5.11c. Infine l'ultima pagina presente è quella per aggiungere un prodotto nuovo all'interno del database, come mostrato in Figura 5.11d. In questo caso quindi verranno chiesti il nome del prodotto, il produttore, la tipologia di prodotto e un'immagine da associarvi.

## 5.6 Sviluppi futuri - Web Application

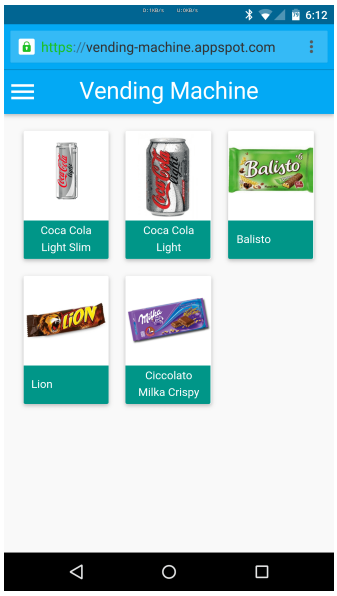
Come già detto tutte queste tecnologie sono nel pieno del loro sviluppo e quindi vengono aggiunte continuamente nuove API e funzionalità. Tra le più interessanti e rilevanti per questa applicazione troviamo: *Gefencing* e *Web Bluetooth*. Un'ulteriore possibile estensione potrebbe essere effettuata aggiungendo i profili utente.



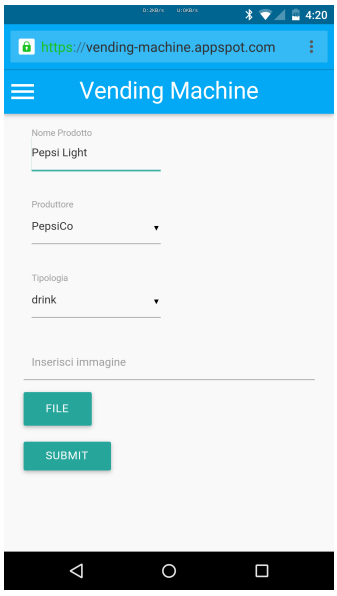

(a) Menu delle opzioni disponibili



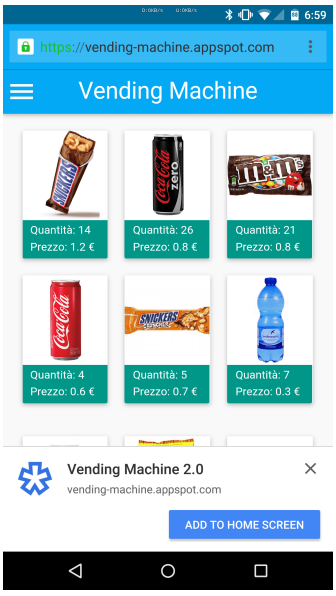

(b) Pagina di rifornimento con prodotti già utilizzati




(c) Pagina di rifornimento con prodotti nuovi




(d) Pagina per caricare un nuovo prodotto




(e) Richiesta di aggiunta alla homescreen


(f) Icona come applicazione nativa

Figura 5.11: Secondo insieme di immagini dell'applicazione web

### 5.6.1 Geofencing

Un'estensione dell'API *geolocation* già presente all'interno del browser, permette di creare delle zone da "osservare" e di scatenare eventi nel momento in cui l'utente entra in una di queste. Questa API, che sarà disponibile attraverso i Service Worker, permetterà allo sviluppatore di accedere alla posizione anche dopo che la Web Application verrà chiusa, per questo ci sono importanti problemi di privacy che devono essere presi in considerazione e sono ancora in fase di decisione le precauzioni da intraprendere. Quando questa API sarà disponibile, per esempio, sarà possibile inviare notifiche agli utenti solo se si trovano in determinate zone e questo risulta molto utile per questa applicazione, in quanto sarà possibile inviare notifiche mirate solo agli utenti vicino a un distributore, senza disturbare gli altri.

### 5.6.2 Web Bluetooth

L'API Web Bluetooth permette di interagire con dispositivi Bluetooth direttamente da una pagina web. Per motivi di sicurezza questa connessione può essere iniziata solo attraverso l'azione dell'utente, come per esempio la pressione di un bottone. Questa API è al momento disponibile solo in Chrome 45 su Chrome OS (canale *beta*) e in Chrome 46 per Android, attualmente nel canale *dev*, ma comunque non è attiva di default e va impostato un flag. Attualmente sono state implementate funzionalità per ricevere informazioni base, ma quella più interessante per questa applicazione, cioè di poter leggere i dati di Advertising di un beacon, non è ancora presente. In futuro inoltre, a specifica ultimata, sarà possibile connettersi direttamente con i dispositivi e si potranno quindi scambiare dati direttamente con il distributore.

### 5.6.3 Profilo utente

Come sviluppo futuro dell'applicazione si potrà aggiungere la possibilità di effettuare un Login, permettendo quindi di avere un proprio profilo, attraverso il quale vengono proposte offerte personalizzate o ordinati i prodotti all'interno del distributore in base alle preferenze. In questa area inoltre sarà possibile salvare le proprie preferenze di pagamento.

## 5.7 Sviluppi futuri applicazione Android

Come per l'applicazione web si potrebbe prevedere l'implementazione di un profilo utente dove salvare le preferenze dei prodotti, per esempio per proporre promozioni mirate oppure programmi di fidelizzazione. Inoltre grazie soprattutto alle API presenti in Android, si potrebbero prevedere funzionalità aggiuntive per esempio grazie al posizionamento.

### 5.7.1 Posizionamento

Attraverso l'utilizzo degli attachment nei beacon Eddystone-UID è possibile ricevere all'interno dell'applicazione qualsiasi tipo di informazione. Al momento vengono associate le coordinate GPS del distributore e con questa informazione si potrebbe visualizzare una mappa che mostri la posizione del distributore e dell'utente e le indicazioni per raggiungerlo, così da poterlo indirizzare al punto esatto. In modo simile, visto che l'applicazione scarica la lista delle geofence, e quindi delle posizioni dei distributori, si potrebbe mostrare una mappa visualizzando tutti i distributori nella zona. Queste funzionalità possono essere sfruttate dalla Google Places API, vista anche la presenza di un identificativo del luogo all'interno del beacon.

### 5.7.2 Preferenze

Nella sezione preferenze, oltre alle impostazioni generali dell'applicazione, potrebbero essere presenti delle impostazioni per le modalità di ricezione delle notifiche, per esempio per filtrare solamente i distributori da cui si vogliono ricevere promozioni. Inoltre qui troverebbero luogo le impostazioni riguardati il profilo e le modalità di pagamento ad esso associate.

# Capitolo 6

# Conclusioni

La costante evoluzione tecnologica sta portando alla luce nuove possibilità, che un tempo non erano nemmeno immaginabili. Uno dei settori che si è rivoluzionato maggiormente negli ultimi anni è quello delle telecomunicazioni e questo ha portato sia a una riduzione dei costi per le connessioni a Internet, sia ad un aumento delle loro capacità e prestazioni; di conseguenza queste connessioni sono diventate più accessibili e affidabili permettendo alle persone e alle cose di accedere a Internet quasi ininterrottamente. Grazie agli sviluppi attualmente in corso, principalmente legati all'introduzione del protocollo IPv6 per l'assegnazione degli indirizzi su Internet, a breve sarà inoltre possibile raggiungere direttamente ognuno di questi dispositivi connessi, in qualsiasi parte del mondo esso sia. In questo modo si potrà collegare qualsiasi tipo di oggetto a Internet e trarne i benefici che questo comporta, ad esempio: la possibilità di ricevere aggiornamenti costanti sullo stato dei sensori all'interno di un oggetto, aggregare e interpretare i dati per avere una comprensione migliore delle dinamiche all'interno di un'azienda o ancora far eseguire ad essi delle azioni in base ad eventi, siano essi innescati da delle persone o da altri oggetti.

Nel presente elaborato, dopo aver analizzato lo stato dell'arte, i benefici, le problematiche e gli attuali interessi da parte di alcuni dei maggiori *player* verso l'Internet of Things è stata analizzata in particolare la necessità di individuare e a controllare la miriade di dispositivi connessi, destinati ad aumentare esponenzialmente nei prossimi anni. È stato quindi individuato e analizzato il progetto "Physical Web", che si pone come obbiettivi proprio quelli di risolvere queste

problematiche. Questo progetto ancora in via sperimentale, sebbene guidato da Google, è open source e non brandizzato e viene proposto come standard, nella speranza venga adottato dai diversi produttori di software e hardware. La soluzione proposta dal Physical Web prevede la creazione di un ecosistema dove gli oggetti smart possano trasmettere URL nell'area intorno a loro. Questo può essere effettuato ad esempio mediante l'utilizzo di beacon Bluetooth che trasmettono un identificativo dell'oggetto (l'URL) che è possibile aprire con un normale browser. Infatti questo progetto si ripropone di sfruttare le potenzialità offerte dal web e di estenderle al modo fisico attorno a noi, permettendo ad ogni oggetto che dispone di un display, come un tablet o uno smartphone nelle vicinanze di intercettare questo URL, interpretarlo e mostrarlo all'utente.

Sono state quindi realizzate ad hoc tutte le componenti necessarie per creare questo ecosistema e analizzate nell'utilizzo applicate a delle vending machines. Dopo aver implementato la componente che si occupa di inviare l'URL tramite il protocollo Eddystone-URL, è stata gestita e creata tutta la parte di interazione con l'utente, che avviene in seguito all'individuazione del beacon, attraverso un'interfaccia web.

Infine, essendo queste tecnologie ancora nel pieno dello sviluppo, sono state paragonate ad altre più mature, sempre facenti parte della piattaforma becon di Google. È stata implementata un'applicazione Android, che attraverso l'utilizzo degli Eddystone-UID individua i beacon e presenta le informazioni del distributore automatico associate. Questa applicazione è stata infine paragonata nelle sue funzionalità all'applicazione web.

Da questo confronto si è potuto constatare come le tecnologie web, anche se molto promettenti, non siano ancora completamente pronte per un utilizzo nel prossimo futuro, soprattutto per la mancanza di integrazione di componenti per la ricerca dei beacon. Al contrario l'applicazione Android risulta sfruttare tecnologie più mature e quindi su cui è possibile fare maggiore affidamento.

In virtù dell'immediatezza di queste tecnologie e degli innumerevoli sbocchi potenziali si ipotizza un'adesione crescente, visto anche l'interesse dimostrato da sempre più soggetti e sviluppatori. Si può inoltre immaginare che questi ostacoli si supereranno nel breve periodo, grazie anche alla rapidità di evoluzione delle tecnologie, portando ad una diffusione nei settori più disparati.

# Bibliografia


[1] The cloud market. URL http://thecloudmarket.com/. 18

[2] Docker. URL https://www.docker.com/. 17

[3] Apple. ibeacon for developers. URL https://developer.apple.com/ibeacon/. 39, 41

[4] Masinter & McCahill Berners-Lee. Rfc1738. URL https://www.ietf.org/rfc/rfc1738.txt. 34

[5] Cisco. Cisco global cloud index: Forecast and methodology, 2013–2018. URL http://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/Cloud_Index_White_Paper.pdf. 22

[6] Don Coleman. Create an eddystone beacon using node.js. URL https://github.com/don/node-eddystone-beacon. 74

[7] Barb Edson. Creating the internet of your things. URL http://download.microsoft.com/download/C/F/7/CF78575B-711E-4E1B-8BAB-3ED1657DFA82/Creating_the_Internet_of_Your_Things.pdf. 8

[8] Google. Specification for eddystone, an open beacon format from google. URL https://github.com/google/eddystone. 33, 34, 35, 36, 37

[9] Intel. The intel® iot platform: Secure, scalable, interoperable. URL https://www-ssl.intel.com/content/www/us/en/internet-of-things/iot-platform.html. 19

[10] Doug Handler Joseph Bradley, Joel Barbier. Embracing the internet of everything to capture europe's share of $14.4 trillion. 2013. 24, 25, 26, 27, 28, 29

[11] Start Commercial Limited. Startcom certificate policy & practice statements. URL https://www.startssl.com/policy.pdf. 72

[12] J Manyika, M Chui, J Bughin, R Dobbs, P Bisson, and A Marrs. Disruptive technologies: Advances that will transform life. *Business, and the Global Economy*, 2013. 5, 7, 8, 10, 11, 24

[13] Microsoft. Guide to universal windows platform (uwp) apps. URL https://msdn.microsoft.com/en-us/library/windows/apps/dn894631.aspx. 13

[14] Radius Networks. Android beacon library. URL http://altbeacon.github.io/android-beacon-library. 43

[15] Bluetooth Developer Portal. Tx power level. URL https://developer.bluetooth.org/gatt/characteristics/Pages/CharacteristicViewer.aspx?u=org.bluetooth.characteristic.tx_power_level.xml. 34

[16] Bluetooth SIG Proprietary. Supplement to the bluetooth core specification. URL https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=291904. 32, 33

[17] World Wide Web Technology Surveys. Usage statistics and market share of linux for websites. URL http://w3techs.com/technologies/details/os-linux/all/all. 14

[18] Doug Thompson. Apple cracks down on ibeacon for android. URL http://beekn.net/2014/07/ibeacon-for-android/. 42

[19] Ubuntu. Snappy, . URL https://developer.ubuntu.com/en/snappy/. 16, 17, 18

[20] Ubuntu. Intel and canonical collaborate around iot gateways, . URL https://insights.ubuntu.com/2015/06/02/intel-and-canonical-collaborate-around-iot-gateways/. 18

[21] Ubuntu. Ubuntu core on the internet of things, . URL http://www.ubuntu.com/internet-of-things/. 18